IBM

Creating Integrated IBM WebSphere Solutions using Application Lifecycle Management

- Creating an end to end solution using the WebSphere portfolio
- Delivering effective application lifecycle management using CLM
- Increasing control and agility using BPM and ODM

Emrah Barkana
Antonella Bertoletti
Stefano Bussaglia
Ernest Calalang
Sebastian Kapciak
Leonardo Olivera
Sergio Polastri
Fabio Silva

Redbooks

ibm.com/redbooks

International Technical Support Organization

Creating Integrated IBM WebSphere Solutions using Application Lifecycle Management

December 2014

SG24-8243-00

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

First Edition (December 2014)

This edition applies to IBM Rational Collaborative Lifecycle Management V4.0.6, WebSphere Application Server V8.5.5, WebSphere Liberty Core 8.5.5, IBM Worklight V6.2, and DB2 V10.5.3.

© Copyright International Business Machines Corporation 2014. All rights reserved.
Note to U.S. Government Users Restricted Rights -- Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices . . . vii
Trademarks . . . viii

IBM Redbooks promotions . . . ix

Preface . . . xi
Authors . . . xi
Now you can become a published author, too! . . . xiv
Comments welcome . . . xiv
Stay connected to IBM Redbooks . . . xv

Chapter 1. Introduction . . . 13
1.1 Business integration problems . . . 14
1.2 The business scenario . . . 14
1.3 Application Lifecycle Management . . . 15
1.4 Using this book . . . 16
1.4.1 Goals and objectives . . . 17
1.4.2 How this book is organized . . . 17

Chapter 2. Solution overview . . . 19
2.1 The solution architecture . . . 20
2.2 Middleware products used for the solution . . . 20
2.2.1 Additional solution artifacts . . . 21
2.2.2 Integration of the components . . . 22
2.3 Deployment topology . . . 23
2.4 Bringing the solution to the cloud . . . 25

Chapter 3. Application Lifecycle Management on a software development project . . . 27
3.1 Define the IBM Application Lifecycle Management methodology . . . 28
3.2 Define Collaborative Lifecycle Management . . . 29
3.2.1 Products and applications . . . 29
3.3 Configure a lifecycle project using Collaborative Lifecycle Management . . . 30
3.3.1 Create the lifecycle project . . . 31
3.4 Contribute to a Collaborative Lifecycle Management project . . . 34
3.4.1 Plan the project . . . 40
3.4.2 Define development lifecycle activities . . . 42
3.4.3 Developers working on a Collaborative Lifecycle Management project . . . 44

Chapter 4. Configuring IBM Business Process Manager integrated with IBM WebSphere Operational Decision Manager . . . 47
4.1 Decision management with WebSphere Operational Decision Manager . . . 48
4.1.1 Business drivers for decision management . . . 48
4.1.2 Decision management with BPM . . . 50
4.1.3 WebSphere Operational Decision Manager . . . 53
4.1.4 Overview of business rules . . . 56
4.1.5 Overview of event rules . . . 63
4.1.6 Business management of rules . . . 75
4.2 Managing business processes with Business Process Manager . . . 80
4.2.1 Overview of BPM . . . 81

© Copyright IBM Corp. 2014. All rights reserved.

4.2.2 Process automation, visibility, and control . . . 83
4.2.3 Overview of Business Process Manager . . . 83
4.2.4 Create a service claim sample . . . 89
4.2.5 Add a rule invocation for rules-related activities . . . 94
4.2.6 Add the user interface for human task activities . . . 100
4.2.7 Create an Event Rule to trigger the business process . . . 104
4.2.8 Add database integration . . . 108
4.2.9 Deploy the process to Process Center . . . 112

Chapter 5. Creating solution artifacts . . . 113
5.1 Prepare the integration environment . . . 114
5.1.1 Integration architecture . . . 114
5.1.2 Installation . . . 115
5.1.3 Configuration . . . 116
5.1.4 Working with integration flows . . . 118
5.1.5 Unit testing . . . 121
5.2 Create the web application for the scenario . . . 123
5.2.1 Install WebSphere Developer Tools in the Eclipse IDE . . . 123
5.2.2 Develop the web application . . . 123
5.2.3 Export the application archive . . . 132
5.3 Prepare the Liberty profile environment for your web application . . . 132
5.3.1 Configure the Java runtime environment . . . 133
5.3.2 Installation . . . 133
5.3.3 Create and configure the server . . . 134
5.3.4 Deploy the application manually . . . 135
5.4 Prepare WebSphere Application Server full profile environment for web application . 136
5.4.1 Install and create the server profile . . . 136
5.4.2 Configure the server . . . 136
5.4.3 Deploy the application manually . . . 138
5.5 Create the mobile application for our scenario . . . 139
5.5.1 Prepare the development environment . . . 139
5.5.2 Develop the mobile application . . . 140
5.5.3 Install the application on the server . . . 159
5.5.4 Updating the application . . . 174

Chapter 6. Rapid application deployment using IBM UrbanCode Deploy . . . 175
6.1 IBM UrbanCode Deploy . . . 176
6.1.1 Components . . . 177
6.1.2 Applications . . . 177
6.1.3 Resources . . . 178
6.2 Prepare the environment . . . 179
6.2.1 Create a deployment process . . . 179
6.3 Build and execute the deployment process . . . 190
6.3.1 Create a new component . . . 190
6.3.2 Create an application for deployment . . . 195
6.4 Deploy the application on Liberty . . . 198
6.4.1 Create a component . . . 198
6.4.2 Create the new application . . . 201
6.4.3 Create the resource . . . 203
6.4.4 Add a component to the application . . . 206
6.4.5 Create the component process . . . 207

Chapter 7. Quality management and the software development project . . . 211
7.1 Quality Management . . . 212

7.1.1 Create a test plan 213
7.1.2 Associate the requirements 215
7.1.3 Create a test case 215
7.1.4 Define the platforms and test environments for the test plan 216
7.1.5 Create a test script 217
7.2 Test Execution 218
7.3 Defect and change management 219
7.4 Test effort management 222

Appendix A. Integrating the Eclipse environment with IBM Rational Collaborative Lifecycle Management 223
Configure installation between Eclipse IDE environment and Rational Team Concert . . . 224
The integrated team 224
Install the Rational Team Concert Eclipse plug-in 225
Rational Team Concert and third-party products 225
Integrating an Eclipse Workbench with Rational Team Concert 225
Configure the repository connection 226
Create a Rational Team Concert Repository Workspace 226
Sharing the Project with Rational Team Concert 227
Associate the Change Set with a Task 229
Check in and accept changes 229
Rational Team Concert dashboards 231
Rational Team Concert and UrbanCode Deploy for continuous integration 232
Configure UrbanCode Deploy 233
Configure Rational Team Concert 233

Appendix B. Importing solution applications 237
Download and extract the web material 237
Import the integration projects 237
Import the web application 238

Appendix C. Database definitions 239
Create entities and attributes for the business scenario 240
Adding attributes to entities 240

Related publications 243
IBM Redbooks 243
Online resources 243
Help from IBM 244

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.

© Copyright IBM Corp. 2014. All rights reserved.

Trademarks

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

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

AIX®
CICS®
DataPower®
DB2®
Global Business Services®
Global Technology Services®
IBM®
IBM UrbanCode™
ILOG®
IMS™
Jazz™
PureApplication®
Rational®
Rational Team Concert™
Redbooks®
Redpaper™
Redbooks (logo) ®
System z®
TXSeries®
WebSphere®
Worklight®

The following terms are trademarks of other companies:

SoftLayer, and SoftLayer device are trademarks or registered trademarks of SoftLayer, Inc., an IBM Company.

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

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

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

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

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

Find and read thousands of IBM Redbooks publications

- Search, bookmark, save and organize favorites
- Get up-to-the-minute Redbooks news and announcements
- Link to the latest Redbooks blogs and videos

Get the latest version of the Redbooks Mobile App

iOS **Download Now** Android

Promote your business in an IBM Redbooks publication

Place a Sponsorship Promotion in an IBM® Redbooks® publication, featuring your business or solution with a link to your web site.

Qualified IBM Business Partners may place a full page promotion in the most popular Redbooks publications. Imagine the power of being seen by users who download millions of Redbooks publications each year!

ibm.com/Redbooks
About Redbooks → Business Partner Programs

THIS PAGE INTENTIONALLY LEFT BLANK

Preface

This IBM® Redbooks® publication demonstrates, through a practical solution and step-by-step implementation instructions, how customers can use the IBM Rational® Application Lifecycle Management (ALM) portfolio to build and manage an integrated IBM WebSphere® Application. Building a business application (mobile and desktop) that uses WebSphere Application Server, IBM MQ, IBM Integration Bus (IIB), Business Process Management (BPM), Operational Decision Management (ODM), and Mobile.

IBM Redpaper™ publication, *Rapid deployment of integrated WebSphere solutions in your cloud*, REDP-5132, is an extension to this IBM Redbooks publication. Using the same practical solution covered in this Redbooks publication, REDP-5132 demonstrates how the IBM PureApplication® System is a “logical extension” versus a “whole new world”, covering PureApplication Patterns and the new PureApplication as a service on Softlayer. Showing how PureApplication patterns can be used to speed up the installation process of the products as described can be found at the following website:

http://www.redbooks.ibm.com/abstracts/redp5132.html?Open

The intended audience for this book is architects, developers, administrators, and DevOps personnel.

Authors

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

Emrah Barkana is an IBM Certified IT Specialist working in Software Group in Turkey. He joined IBM in May 2008 as IBM Software Services for WebSphere (ISSW). Since then, Emrah worked on several industrial areas, specialized on WebSphere Application Server, WebSphere Process Server, WebSphere Portal Server, IBM ILOG®, Business Process Management, and Master Data Management. Before he joined IBM, he was working for ISBANK in Turkey as a business analyst and WebSphere Process Server developer.

© Copyright IBM Corp. 2014. All rights reserved.

Antonella Bertoletti is an Executive IT Specialist working for IBM Italy, in Milan. She joined IBM in 1987 and spent the first ten years of her career in the HW Division (now STG). She is specialized in both OLTP areas (IBM CICS® and IBM TXSeries®, WebSphere Application Server, CICS Transaction Gateway) and Cloud Computing and Virtualization (IBM PureApplication Systems, IBM Workload Deployer, WebSphere Application Server Hypervisor Edition, WebSphere Virtual Enterprise, WebSphere Compute Grid, WebSphere eXtreme Scale). She has been involved in many different projects using OLTP products, distributed computing transactional technology, and virtualization on several platforms, especially in the Insurance, Banking, and Manufacturing sectors. Antonella is member of the IBM Academy of Technology and she is the PureApplication System Technical Leader for Italy. She has written IBM Redbooks publications and white papers on TXSeries, CICS Transaction Gateway, and WebSphere products. She is currently working for the Software Group Division, in the WebSphere Technical Sales and Solutions - Europe IOT team. Antonella holds a degree in Economics from Pavia University.

Stefano Bussaglia is a WebSphere Client Technical Specialist at IBM Switzerland. Before joining IBM, he was working as a Middleware specialist in a bank, developing the backbone with IBM WebSphere software. Stefano has more than 30 years of IT experience mainly in developing client/server and web applications. The client experiences that he gained are coming from running his own company from more than 10 years. His area now is serving the clients from IBM with IBM PureApplication System and WebSphere Application Server.

Ernest Calalang is a Certified Advisory IT Specialist in the IBM Software Services for WebSphere in the Philippines. He is also an IBM Certified Instructor for the Application Integration and Middleware portfolio. Since joining IBM in 2008, he has six years of industry experience in the banking, telecommunications, and utility sector. His areas of expertise include enterprise integration using IBM Integration Bus, WebSphere Application Server, WebSphere IBM DataPower®, WebSphere Business Process Management, and IBM PureApplication Systems. He holds a degree of Bachelor of Science in Computer Science from the University of Santo Tomas, Philippines.

Sebastian Kapciak is an IBM Certified IT Specialist working for IBM Poland. He has more than 10 years of IT experience in delivering middleware, security, and open source solutions. His areas of expertise are system integration and Java Platform, Enterprise Edition technologies. Sebastian is a certified Open Group TOGAF9 architect and holds a Master's degree in Information Technology from the University of Technology of Warsaw.

Leonardo Olivera is an IT Specialist at IBM Global Services in Uruguay. He has 20 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, sensor and actuator solutions, and application security. During the last few years he has been working with mobile application platforms, including IBM Worklight®.

Sergio Polastri is an IBM IT Specialist at IBM Global Business Services® in Brazil. He joined IBM in August 2000 and since then has been working with Rational products and solutions. He is currently leading the Tools team of the Processes Method and Tools Team. His team supports Rational tools deployment for over more than 45 projects in the Global Delivery Center (Latin America). He holds a Bachelor's degree in Computer Systems Engineering from Centro Universitario da Cidade do Rio de Janeiro in Brazil and a Post-Graduate Degree in Computer Network Management from Centro Universitario da Cidade do Rio de Janeiro in Brazil.

Fabio Silva is an IT Specialist working for IBM Global Technology Services® in Brazil. He joined IBM in January 2007 and the first six years of his career, he supported WebSphere family and now PureApplication System. His main areas of expertise are automation for WebSphere administration, PureApplication Script Packages and Plug-ins, infrastructure design in Legacy and Cloud, implementation, maintenance, and problem determination of the Web environment. He has more than 15 years of IT expertise in fields, such as middleware, Java programmer, open source tools, UNIX operating systems, network security, and web hosting environments. He has designed, implemented, and supported various middleware infrastructure and web hosting environments in large public and private organizations. He holds a Bachelor's degree in Compute System Engineering from Centro Universitario da Cidade do Rio de Janeiro in Brazil.

This project was led by:

Margaret Ticknor, a Redbooks Project Leader in the Raleigh Center. She primarily leads projects about WebSphere products and IBM PureApplication System. Before joining the ITSO, Margaret worked as an IT specialist in Endicott, NY. Margaret attended the Computer Science program at State University of New York at Binghamton.

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

- Jim Amsden, Senior Technical Staff Member Solution Architect for Rational Government Industry Solutions
- Rajeev Gandhi, STSM, ISSW Tech Consultant
- Derrick W. Foley, Manager, WebSphere Cloud Initiatives
- Robbie John Minshall, IBM Software Group, Rational, DevOps, Continuous Delivery, and Cloud Adoption
- Scott Walden, IBM PureApplication System Enablement

- Brutus Zhou, Architect, Business Rules/Event/Process

Thanks to the following people for supporting this project:

- Deana Coble, IBM Redbooks Technical Writer and Video
- Richard Conway, IT Support
- Karen Lawrence, IBM Redbooks Technical Writer
- Tamikia Lee, IBM Redbooks Residency Administrator
- Mahesh Nagabhushan, IBM Redbooks Graphics Editor

Thanks to the authors of IBM Redbooks publication SG24-8065.

Thanks to the authors of IBM Redbooks publication SG24-7973.

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:

  https://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

1

Introduction

The modern IT world shows a rapid change in application integration and systems development. With this rapid change, organizations now try to:

- Shorten delivery times
- Reduce the costs associated with software and mobile application development
- Maintain the quality of product offerings
- Closing the demand and supply gap

Although these objectives are occasionally conflicting, each objective is a high priority.

Application integration and systems development now need a unified process for managing projects, from requirements gathering, to application development, to governance.

Application Lifecycle Management (ALM) is a methodology that enables you to track project development efforts, while maintaining low overall cost and high product quality. It is a methodology that encompasses the significant development and deployment processes, from requirements gathering to governance.

This IBM Redbooks publication documents the use of Application Lifecycle Management in an integrated WebSphere environment. It also shows you how to build your environment and track your progress by the use of governing processes in Application Lifecycle Management.

This chapter describes a business scenario that will be used throughout this book. The following topics are covered:

- 1.1, “Business integration problems” on page 14
- 1.2, “The business scenario” on page 14
- 1.3, “Application Lifecycle Management” on page 15
- 1.4, “Using this book” on page 16

© Copyright IBM Corp. 2014. All rights reserved.

1.1 Business integration problems

This is the age when all of our devices are connected, from computers, to mobile phones, and tablets, to cars. Even our homes are connected. Each system is intelligent enough to give you directions for accomplishing a task, for example, when you are driving, there are mobile applications that provide the best route to travel to avoid traffic jams.

While it may seem easy to invent and introduce new applications that help us in our daily lives, the term *integration* is quite different. Integration, as defined by Merriam-Webster, is "the combining and coordinating of separate parts or elements into a unified whole." To create a process in this era, you need a perfect blend of IT and business to assimilate a unified, intelligent process.

Most companies experience integration problems today because they have existing monolithic applications that only serve one purpose, to perform a single function, and are built on different architectures. A user can use one or more applications to perform a single task, thus slowing down productivity. There are existing applications that need to be connected to perform a single transaction.

1.2 The business scenario

In this section, we describe a fictitious business scenario. Throughout this book, we refer to this scenario, and how Application Lifecycle Management can assist with meeting current project requirements. The scenario takes place at the Redbooks Company Service Desk, where process automation has become problematic. Figure 1-1 depicts this scenario.

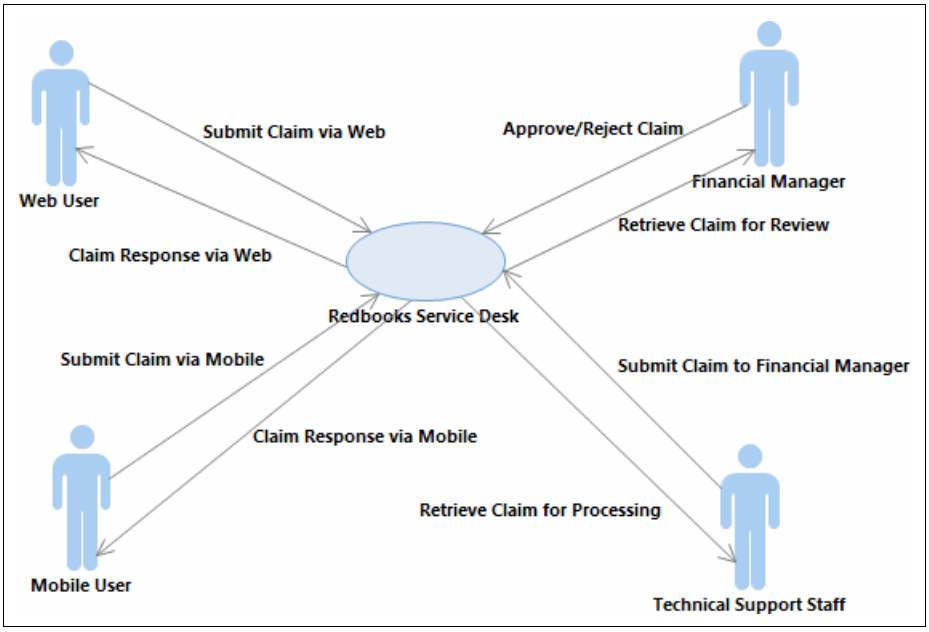


Figure 1-1 Business scenario for the Redbooks Company Service Desk

The Redbooks Company is having a problem with a lack of automation at their service desk. By repeating manual tasks, such as reviewing warranty claims and parts, the company becomes less productive.

In this business scenario, we describe a warranty claim system that is used to accept all of the claims that are entered by the users from either the web or mobile devices. The system can determine the appropriate course of action the business should take, given the clients' input, to track and resolve warranty claims. A business process is created to automate tasks, and business rules are created to determine which category the product falls into.

The following steps are part of the sample scenario in this book:

1. A client provides the serial number of a purchased product and submits it from either the web or a mobile application.
2. When a claim is received, the system launches a business process to initiate the warranty review.
3. If the system records multiple pending claims for the same serial number over a specified period, the system automatically rejects the claim. Otherwise, the system triggers an event to indicate that these claims with the same serial number need further investigation.
4. Multiple claims with the same serial number are be routed to technical support staff for further investigation, and they will assess if the claim is valid.
5. This process then invokes a business rule, wherein the serial number is assessed if the parts that are affected have low or high impact in terms of cost. A claim requires special handling if the assessment from the business rules produces a high impact to cost.
6. In cases for which there is a high impact to cost, the financial manager then approves the claim for special handling. Otherwise, the claim is processed automatically for implementation.
7. All claims are written to the database for logging and tracking purposes.

This scenario describes how project stakeholders can take advantage of events using well-defined processes to create rapid changes and flexible business decisions. This solution describes how the components fit together to become one system.

1.3 Application Lifecycle Management

In today's fast-paced environment, project management usually works in silos, and this can often cause project delays, thus adding to the overall cost of the project. The structure of teams today is tightly coupled. Teams cannot send a solution to the next team to integrate, test, or deploy. The business should be able to know the demands and needs of their users. Business analysts should be more involved with users, so they are aware of the users' pain points and where the system can be improved. Architects need to be involved by determining which architectural principle or pattern applies to the business. They should know the stakeholders' needs as well. These then must be cascaded to the software developers, who can understand the business well and identify assets that can be reused. For software testing, quality assurance testers need to be closely aligned with the business analysts to ensure that the applications delivered are high quality.

When complexity affects quality, it affects the entire business. This costs your organization real bottom-line cost, damage to your reputation and client relationships, and damage to your clients' loyalty. There is also the risk of managing multiple applications that can cause product recalls. This, too, is damaging to business.

Application Lifecycle Management is a methodology in which all of the activities can be tracked based on the following steps (see Figure 1-2 on page 16):

- Project management
- Asset Management

- Requirements gathering
- High-level design (architecture)
- Low-level design
- Build and software development
- Quality assurance
- Release management
- Change management

Application Lifecycle Management is a widely accepted methodology. It involves the coordination of activities and assets to produce, manage, and change software applications throughout their lifecycle. IBM, Hewlett-Packard, and Computer Associates are some examples of vendors that have products that implement Application Lifecycle Management. IBM uses Rational Collaborative Lifecycle Management as their implementation of Application Lifecycle Management. IBM has offerings to support each of the practices that are required for continuous delivery, that is, the develop, build, deploy, test, and release practices:

- Rational Collaborative Lifecycle Management offerings enable agile development to support collaborative development, enabling the output of agile development with continuous integration.
- IBM UrbanCode™ solutions enable application release automation and support the practice of continuous release and deploy.

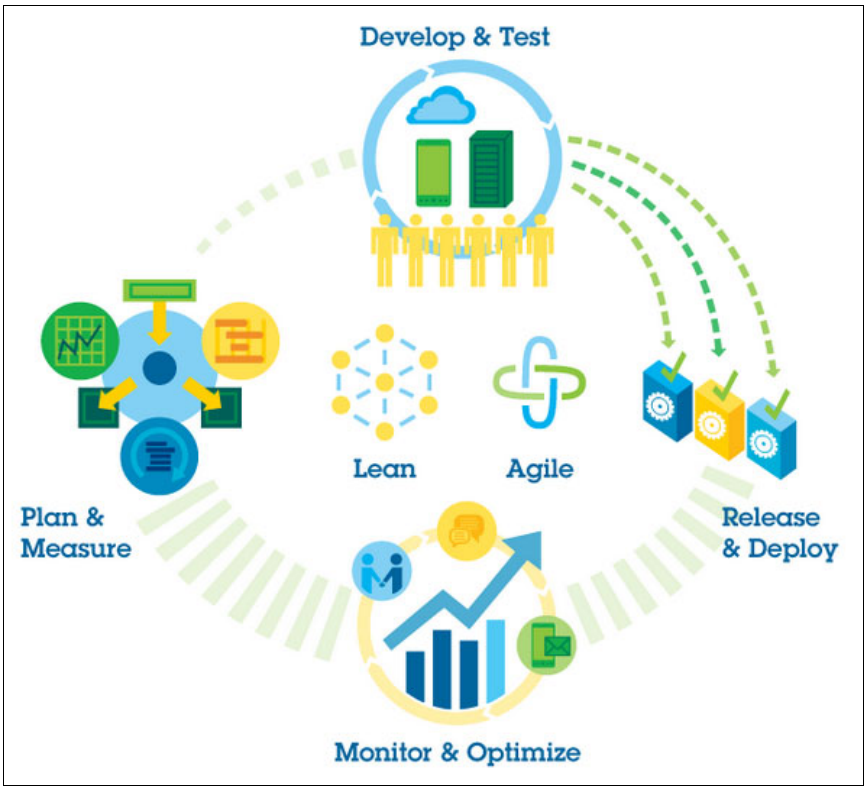


Figure 1-2 Picture illustrates a Software Development Lifecycle

1.4 Using this book

In this section, we list the goals and objectives of this book and describe how it is organized.

1.4.1 Goals and objectives

This book has the following goals and objectives:

- Use ALM to track the progress of the project
- Build your WebSphere environment by using ALM
- Identify change requests
- Create a package for deployment

1.4.2 How this book is organized

This book is organized so that users understand the use of Application Lifecycle Management and how to build a WebSphere environment:

Chapter 1, “Introduction” on page 13 describes the solution from a high-level perspective, outlines the business problem of the Redbooks Company Service Desk, and briefly identifies how the solution can alleviate the business problem.

Chapter 2, “Solution overview” on page 19 discusses the solution and how it addresses the business problem of the Redbooks Company Service Desk.

Chapter 3, “Application Lifecycle Management on a software development project” on page 27 discusses the roles and responsibilities of each project member in the Application Lifecycle Management process, and how to manage projects using this methodology.

Chapter 4, “Configuring IBM Business Process Manager integrated with IBM WebSphere Operational Decision Manager” on page 47 provides an in-depth discussion about how a WebSphere environment is integrated. We discuss the procedures for how it is built and how it is managed.

Chapter 5, “Creating solution artifacts” on page 113 describes the front-end application development (web and mobile) process, using IBM WebSphere Application Server and Worklight. It also provides information about how the integration flows are created in IBM Integration Bus.

Chapter 6, “Rapid application deployment using IBM UrbanCode Deploy” on page 175 shows how to automate your deployment process by creating components.

Chapter 7, “Quality management and the software development project” on page 211 discusses the testing methodologies and procedures that are used to assert the quality of this solution.

Note: IBM Collaborative Lifecycle Management is IBM’s implementation of Application Lifecycle Management. This chapter only discusses manual testing. To learn more about automated testing using IBM Rational Functional Tester, see the following link:

`http://www-03.ibm.com/software/products/en/functional`

2

Solution overview

The IBM WebSphere Portfolio offers software that can easily be integrated in your business or in your system, as discussed in 1.2, “The business scenario” on page 14. This chapter covers the solution for addressing the business problem.

The following topics are covered:

- 2.1, “The solution architecture” on page 20
- 2.2, “Middleware products used for the solution” on page 20
- 2.3, “Deployment topology” on page 23
- 2.4, “Bringing the solution to the cloud” on page 25

© Copyright IBM Corp. 2014. All rights reserved.

2.1 The solution architecture

To meet all of the requirements for the business problem described in 1.2, “The business scenario” on page 14, several software components have to be used.

To begin with, there is a need to provide two different access channels for the users, so web and mobile access will be provided by separate services. Both services trigger the Redbooks Company Service Desk business process, which handles the execution of business rules and approvals required from technical support and financial manager to complete the process. It is highly suggested to use a single integration point that handles the transportation and mediation of incoming messages. Such a component performs the enterprise service bus role. It translates the incoming messages so that process management can understand them. The last important component of the solution is the database where all claim records are kept and updated throughout the business process lifecycle.

Figure 2-1 shows a high-level solution overview for our business case.

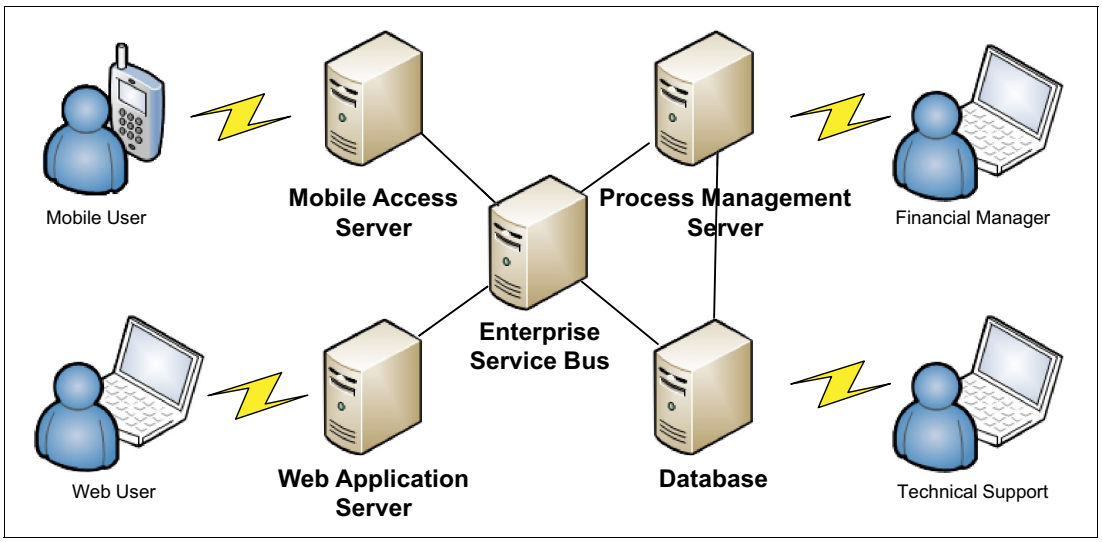


Figure 2-1 High-level solution outline for the Redbooks Company Service Desk system

The following outlines the key elements that implement the solution:

- Web application server
- Mobile access server
- Enterprise service bus
- Process management server
- Database

2.2 Middleware products used for the solution

The solution requires the implementation of each of the elements specified in Figure 2-1. Table 2-1 on page 21 maps the key solution elements to specific IBM software products that will be used for the implementation. These products will establish a solid technology baseline for the solution, upon which the customer-ready system can be created.

Note: Using all of the mentioned IBM products to create this simple scenario in real-life would be an exaggeration, however this simple case is perfect to illustrate how separate WebSphere components integrate with each other to create the solution.

Table 2-1 Map of the key solution elements to IBM product components

Solution component name	Product name	Function within this solution
Web Application Server	WebSphere Application Server	Provides a scalable Java Enterprise Edition platform that will run a custom developed web application and many advanced features and tools that can be used with little or no configuration to perform a fast integration with other components.
Mobile access server	IBM Worklight	Provides an advanced platform for developing, deploying, hosting, and managing mobile enterprise applications for smart devices. The Worklight server is also a scalable Java based gateway between applications and internal services.
Enterprise service bus	IBM Integration Bus	Provides a great variety of options for implementing a universal integration foundation, based on enterprise service bus (ESB). It enables the integration of data sources from both web and mobile channels, and provides the transformation of the messages running inside the solution.
Process management server	Business Process Manager Advanced	Provides a comprehensive business process management (BPM) platform. The Advanced edition supports high-volume automation and extensive system integration and human workflow. The main components of this product are the process server, process center, IBM Process Designer, and IBM Integration Designer.
	IBM Operational Decision Manager	Provides a comprehensive platform for the management and execution of business rules and business events: ► WebSphere Decision Center enables business users to govern business rules and business event-based decision logic. ► WebSphere Decision Server automates decision logic, enabling sense and respond actions based on context of an event.
Database	IBM DB2®	Provides a scalable, enterprise-wide solution for handling high-volume workloads and relational data structures.

2.2.1 Additional solution artifacts

The final solution consists of the elements specified in Table 2-1. Each of the components has to be configured and integrated with other components. There is also a need for custom-made artifacts, such as applications, business processes, rules, and integration flows.

This is a brief list of additional artifacts that complete the solution:

- The dynamic web application packaged as Web ARchive (WAR), which includes an HTML generated claim form for the user and controller logic for sending a message containing the user input to the Integration Bus.
- The mobile, integrated application that enables mobile users to file the same claim form as the web application and send it to the IBM Integration Bus from a smart device.
- Integration flows that will retrieve messages from both web (Extensible Markup Language (XML) message) and mobile (JavaScript Object Notation (JSON) message) sources, transform them, persist in the database, and invoke the business process.
- The business process and set of business rules that model the acceptance process of the claim form, including human tasks required for the business user to submit a claim for approval.
- The database containing tables and relations required to store claim requests from users.

2.2.2 Integration of the components

Now that we described the technology that will be used and what kind of artifacts have to be produced to create the solution, the next step is to design the integration setup of the system.

The main integration point of the solution is the Integration Bus, which is capable of handling a wide variety of different protocols. To demonstrate the protocols, we choose to use WebSphere message queuing (MQ) as an asynchronous method of delivering messages from the application server to the Integration Bus. The web application will use Java Message Service (JMS) to deliver an XML formed message to the WebSphere MQ queue. The Integration Bus will listen on this queue and pick up any message that arrives.

The mobile application will integrate with the Integration Bus through a Worklight HTTP adapter, which will use a Representational state transfer (REST) service to send a synchronous HTTP request using JSON data.

> **Note:** In an ideal world, systems and their services should be designed to use a common technology standard wherever possible to communicate. This reduces the number of different technology stacks and simplifies the administration. In this scenario, multiple integration standards were used to demonstrate several integration alternatives that can be used with WebSphere software.

When the user submits the claim form, both web and mobile applications generate a tracking number, enabling users to check the status of the request. This tracking number, along with a time stamp, will be included in data payload that is sent to the Integration Bus.

The Integration Bus will transform the incoming messages so that it will be able to persist them in a relational database, using the native open database connectivity (ODBC) drivers. Additionally, the Integration Bus will update a log file for each request, after it commits the record to the database. This log will be an additional solution feature that can be helpful for administrators in tracing whether the request came from the mobile or web applications. The last part of the integration flow that is executed by the Integration Bus will be the invocation of a business rule, which will be exposed as a web service by the Operational Decision Manager.

The purpose of the exposed business rule will be to verify if the user claim request is valid and whether it should initiate the business process. If so, the Operational Decision Manager will trigger the business process using a web service call to the Business Process Manager.

During the business process flow, the Business Process Manager will be the supervisor of the business process state, but it will ask the Operational Decision Manager for any decisions that are based on business rules (for example, to check if the serial number requires additional financial manager approval). Business Process Manager will handle the human tasks, such as providing approvals, and it will update the final status of the claim form in the database. Both Business Process Manager and Operational Decision Manager will use JDBC connections to communicate with the database where the claim form was persisted. For more details and a description of the business process used in this scenario, refer to Table 4-4 on page 93.

Figure 2-2 shows the described integration model for the solution.

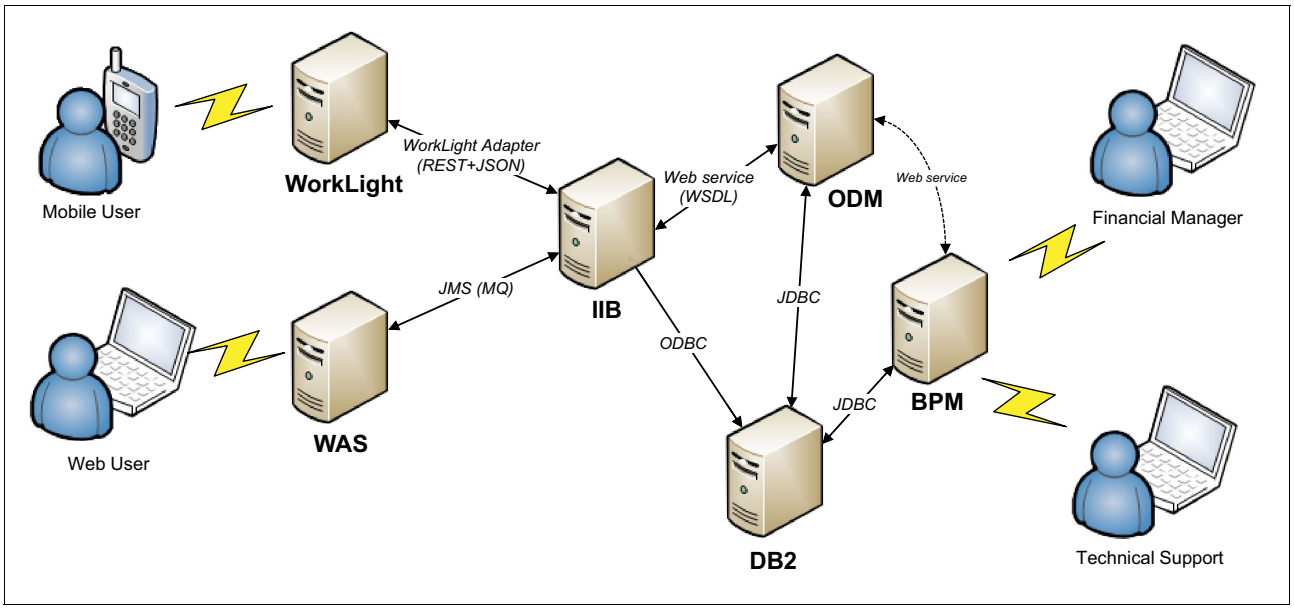


Figure 2-2 Solution integration model

2.3 Deployment topology

The final solution will by composed from the artifacts described in 2.2.1, “Additional solution artifacts” on page 21 that will be integrated accordingly to 2.2.2, “Integration of the components” on page 22. However, there is still a need to discuss the deployment topology and configuration aspects of each of the solution components.

Figure 2-3 on page 24 presents a detailed deployment diagram of the components in conjunction with the custom-made artifacts.

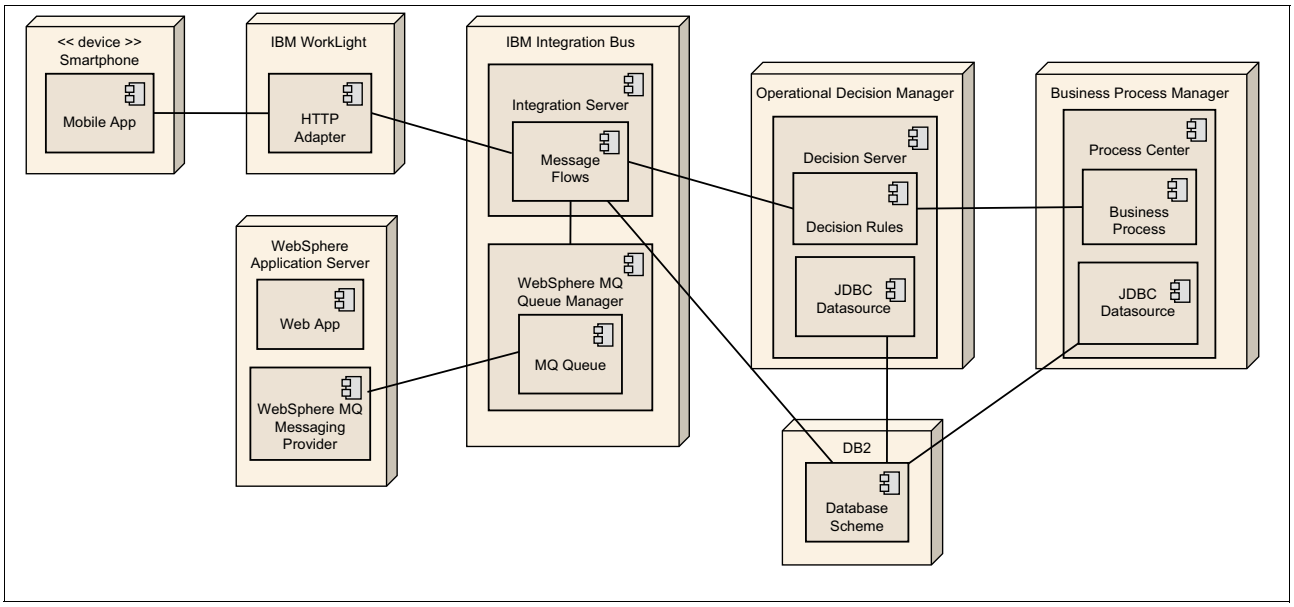


Figure 2-3 Deployment diagram of the components and the custom made artifacts

As you see from Figure 2-3, some of the integration between the solution components is done using functionalities that are provided by products, such as messaging providers, but some require further development to ensure proper integration, for example, custom integration flows.

Let us start with the Integration Bus. It consists of two integral components: the Integration Server and the WebSphere MQ Queue Manager. Both will be installed on the same system. All custom integration flows with the integration logic will be executed in the Integration Server runtime environment, which has access to the WebSphere MQ Queue Manager. There will also be a single queue defined on the WebSphere MQ Queue Manager, which will be exposed to the WebSphere Application Server. The web application will use the WebSphere MQ messaging provider from the WebSphere Application Server to communicate with the WebSphere MQ Queue Manager and send the messages to its WebSphere MQ queue.

The mobile application will be running on the business user's smart device. To communicate with the Integration Bus, it will use the IBM Worklight environment, which provides an HTTP adapter. This adapter will be configured to communicate with the Integration Bus message flow.

Note: This is a simplified scenario to demonstrate the integration capabilities of the IBM WebSphere components. In a real-life, production environment, both web and mobile users should not be able to communicate directly with the WebSphere Application Server or Worklight server. An additional demilitarized zone (DMZ) with HTTP proxy servers should be considered to secure this communication.

The business process part of the solution is running on two components installed separately:

- The first is the Decision Server, which is part of the Operational Decision Manager product. It will execute all decision rules used in the solution.
- The second is Process Center, which is the runtime environment for business processes, which are defined in the Business Process Manager.

Both products are implemented on the WebSphere Application Server runtime environment and will use its JDBC Provider features to connect to the DB2 database.

2.4 Bringing the solution to the cloud

Having a complete solution, the next step is to increase its adaptability, reliability, and scalability, and also optimizing operational costs with the expert integrated system.

The best way to achieve all these factors is using PaaS (Platform as a Service) on IBM cloud environment. Bringing the solution to the cloud, it can derive advantage of other benefits like automated creation from patterns or replication of the whole solution. It is an effective way for seamless integration of the software lifecycle like from development to production, which can be created in a matter of hours instead of months.

In case of increased workload, such approach enables to respond very quickly to the demand thanks to transparent horizontal and vertical scalability. Additionally, it can be a very efficient mechanism to control and optimize the number of licenses used for the solution. On this topic, follow *Rapid deployment of integrated WebSphere solutions in your cloud,* REDP-5132:

http://www.redbooks.ibm.com/abstracts/redp5132.html

With the IBM Redpaper publication, learn how to migrate from a classic, on-premises system design to a scalable cloud environment. It guides through an end to end migration process including preparation of automated scripts packages, configuring patterns, and many other considerations. It covers the IBM cloud offerings of IBM PureApplication System and IBM PureApplication Services on SoftLayer® and uses the business solution created and covered in this book.

3

Application Lifecycle Management on a software development project

This chapter describes how Application Lifecycle Management orchestrates the activities involved in the software development project, which is related to the business scenario described in 1.2, “The business scenario” on page 14.

The following topics are covered:

- 3.1, “Define the IBM Application Lifecycle Management methodology” on page 28
- 3.2, “Define Collaborative Lifecycle Management” on page 29
- 3.3, “Configure a lifecycle project using Collaborative Lifecycle Management” on page 30
- 3.4, “Contribute to a Collaborative Lifecycle Management project” on page 34

© Copyright IBM Corp. 2014. All rights reserved.

3.1 Define the IBM Application Lifecycle Management methodology

Playing in a band is like being on an ideal team, where each member has a different part to play. The ideal team that works together synergistically knows exactly what to do and when, and they get immediate feedback during their concerts (see Figure 3-1).

Figure 3-1 The ideal team works together synergistically

For a software development project, the efforts are more complicated. Teams are dispersed, frequently across widely separated time zones, with pressure to deliver more functions with fewer resources. The team needs to adhere to standards and follow processes that are difficult to apply consistently across the different disciplines.

Deploying a software project to production is a live performance. All rehearsals are in the past.

Application Lifecycle Management is the vehicle that brings the teams and disciplines together. It drives the transparency of their progress and governs how the team works. Today's software development teams can produce high-quality applications and deployments using Application Lifecycle Management.

Application Lifecycle Management coordinates a set of contributions among all team members. Coordination efforts involve people, processes, information, and tools, including the underlying transparency and shared responsibility for success. Stated simply, what one team member produces, another team member consumes, and this concept is employed during the entire development lifecycle. Understanding these dependencies helps a project team to:

- Organize their efforts
- Streamline their ability to produce software
- Produce a productive and efficient software development environment

Application Lifecycle Management provides relationships that help teams to successfully manage software development projects. The goal of Application Lifecycle Management is to help teams coordinate and manage their efforts and to deliver software projects successfully.

3.2 Define Collaborative Lifecycle Management

The IBM Collaborative Lifecycle Management is the IBM Rational solution for Application Lifecycle Management, as it addresses Application Lifecycle Management capabilities using IBM Rational products.

Collaborative Lifecycle Management coordinates software development activities across a number of components of software development:

- Systems and business requirements
- Design
- Development
- Build
- Test
- Delivery

Note: This book describes the IBM Rational products that apply to the business scenario discussed in 1.2, “The business scenario” on page 14.

3.2.1 Products and applications

Collaborative Lifecycle Management can be used to link artifacts for traceability in a web-like navigation to create reports and dashboards, and to do so across your Collaborative Lifecycle Management applications. Each Collaborative Lifecycle Management application consists of one or more capabilities. The following Rational products are part of Collaborative Lifecycle Management:

- *Rational Requirements Composer:* This product addresses the requirements management capability and uses the Requirements Management application. This capability provides tools to capture, organize, and collaboratively review, analyze, and report on requirements, especially in relation to their associated development work items and test artifacts.
- *IBM Rational Team Concert™:* is a team collaboration tool that is built on a scalable, extensible platform. Rational Team Concert uses the Change and Configuration Management (CCM) application to provide features that integrate development project tasks including iteration planning, process definition, change management, defect tracking, source control, build automation, and reporting. This product addresses the following capabilities:
  - *Change Management:* The main feature of change management is work items, which track and coordinate stories, defects, plan items, and ordinary tasks. The work items and the workflow process they follow can be customized to suit your project.
  - *Planning*: The planning capability provides tools to assist with the planning, tracking, and workload balancing for entire projects, for teams within those projects, and for individual developers. Plans are accessible to everyone on the team, and show the progress on releases and iterations at any point in time.
  - *Software configuration management*: The component-based source control system provides strong support for parallel development, agile development, and geographically distributed teams. It integrates tightly with defect tracking, builds, and process automation.
  - *Automation*: The automation capability provides build management control to the development and test teams. Team members can track build progress, view build alerts and results, request builds, and trace the relationship of builds to artifacts, such as change sets and work items.

– *Reports:* Customizable reports provide both real-time views and historical trends of artifacts across the entire lifecycle, including requirements, work items, builds, test cases, and test results. Team reports and dashboards help to keep tabs on the health of your project. Dashboards provide an at-a-glance view of work item queries, event feeds, reports, and other items critical to understanding the progress of the project.

- *Rational Quality Manager:* This product consists of Requirements Management, Change and Configuration Management, and Quality Management applications. Features include dynamic test plans, governed workflows, lab efficiency, test coverage analysis, and manual test authoring. These features integrate with other lifecycle artifacts, such as work items and requirements, and with reporting and dashboards. They provide detailed and highly customizable analytics to help you monitor the health and progress of a project.

These products are delivered as a set of applications together with an IBM *Jazz™ Team Server*.

The *Jazz Team Server* (JTS) serves as an integration hub for the applications. It provides the foundational services, such as user management and license management, that enable the Collaborative Lifecycle Management applications to work together as a single logical server (see Figure 3-2).

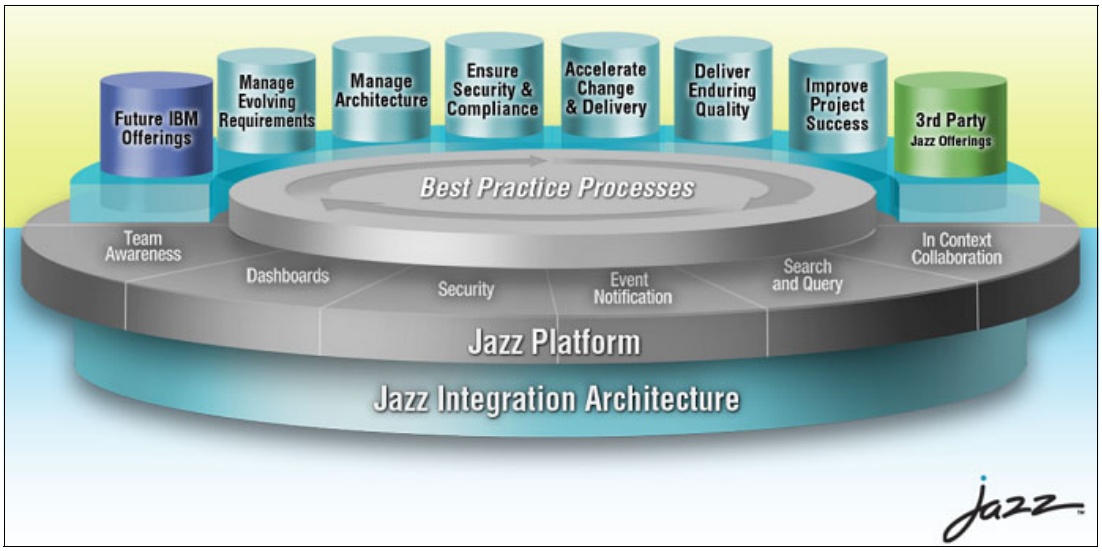


Figure 3-2 ALM capabilities covered by the Rational Collaborative Lifecycle Management Solution

Note: For more information about IBM Collaborative Lifecycle Management and other IBM Rational Products available, see the following website:

http://jazz.net

3.3 Configure a lifecycle project using Collaborative Lifecycle Management

This section describes how to create a lifecycle project and how to configure the application integration for the project.

Note: For Collaborative Lifecycle Management installation, setup, and support, see:

http://jazz.net

Table 3-1 shows the Rational products that are used in the scenario discussed in 1.2, “The business scenario” on page 14.

Table 3-1 Rational products included in the business scenario

IBM Rational product	Version
Rational Requirements Composer	4.0.6
Rational Team Concert	4.0.6
Rational Quality Manager	4.0.6

Table 3-2 lists the defined roles for a project that uses the Collaborative Lifecycle Management solution to orchestrate software development project activities:

Table 3-2 Defined roles in the Collaborative Lifecycle Management solution

Role	Definition	Role name in this book
Stakeholder	Customer	stakeholder
Project manager	Project manager	manager
Project leader	Project team leader	leader
Business analyst	Business requirements analyst	analyst
Developer	Developer	developer and developer_A
Tester	Tester	tester
Release Engineer	System Integrator	release_engineer

3.3.1 Create the lifecycle project

After the Collaborative Lifecycle Management products are installed, proceed with the following steps to create the lifecycle project:

Note: This book uses the Open Unified Process (OpenUP), which is a lifecycle software development process and industry standard that provides a disciplined approach for task and responsibility management. To learn more about OpenUp, see:

http://epf.eclipse.org/wikis/openup

1. Navigate to the Change and Configuration Management administration page, at https://<server_url>:9443/admin/web/projects (see Figure 3-3 on page 32).

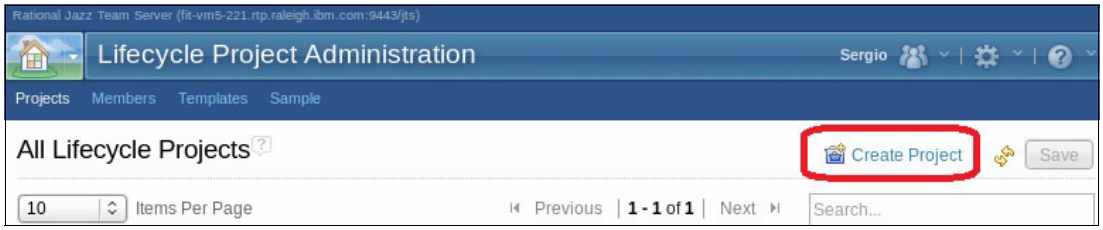


Figure 3-3 Create a new Collaborative Lifecycle Management project

2. Create a name for the project (see Figure 3-4).

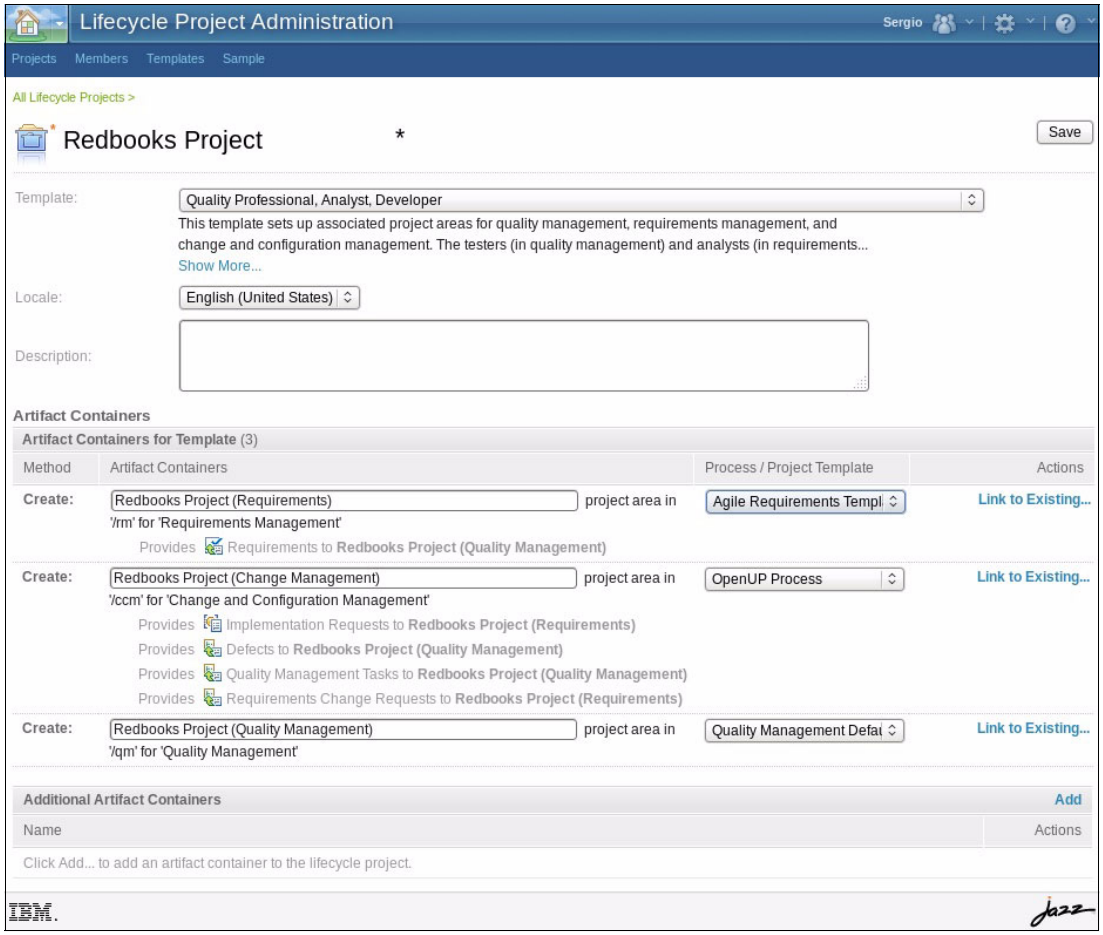


Figure 3-4 This book uses the OpenUP industry standard for requirements

3. Click **Save** to create your project.
4. Click an application of interest in the Configured Artifacts Containers section (see Figure 3-5 on page 33).

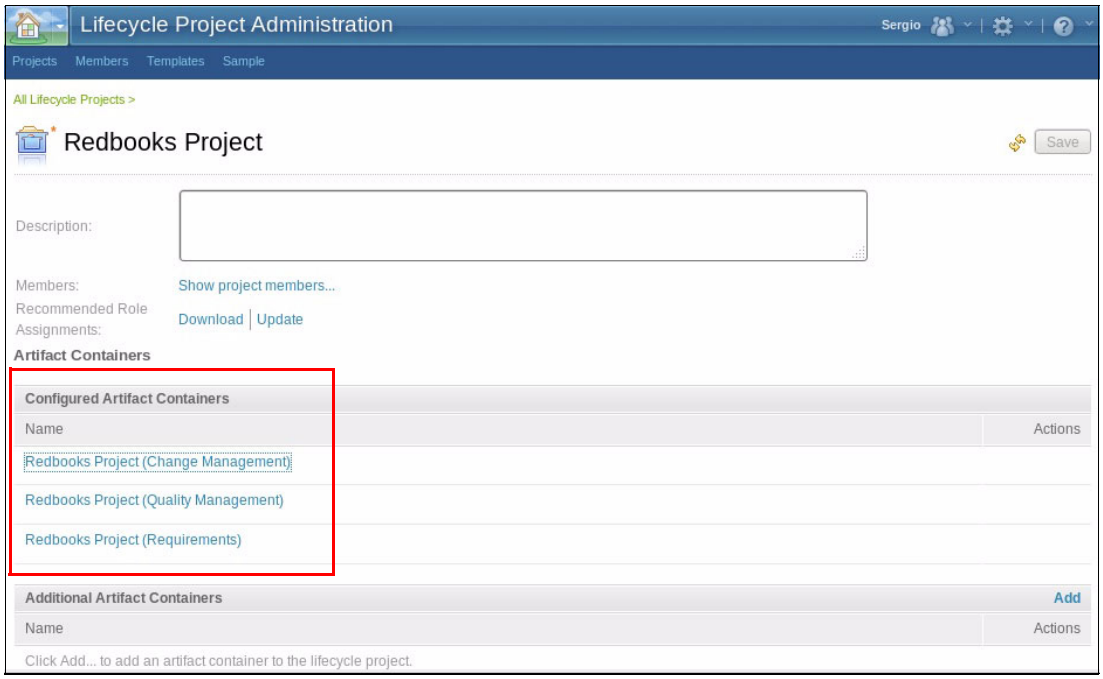


Figure 3-5 Select an application in the Configured Artifacts Container to configure each project area

For each application, one project area is created. A *project area* is an area in the repository where information about one or more software projects is stored. A project area defines deliverables, team structure, process, and schedule (see Figure 3-6 on page 34). You can also configure project area information using the Rational Team Concert client.

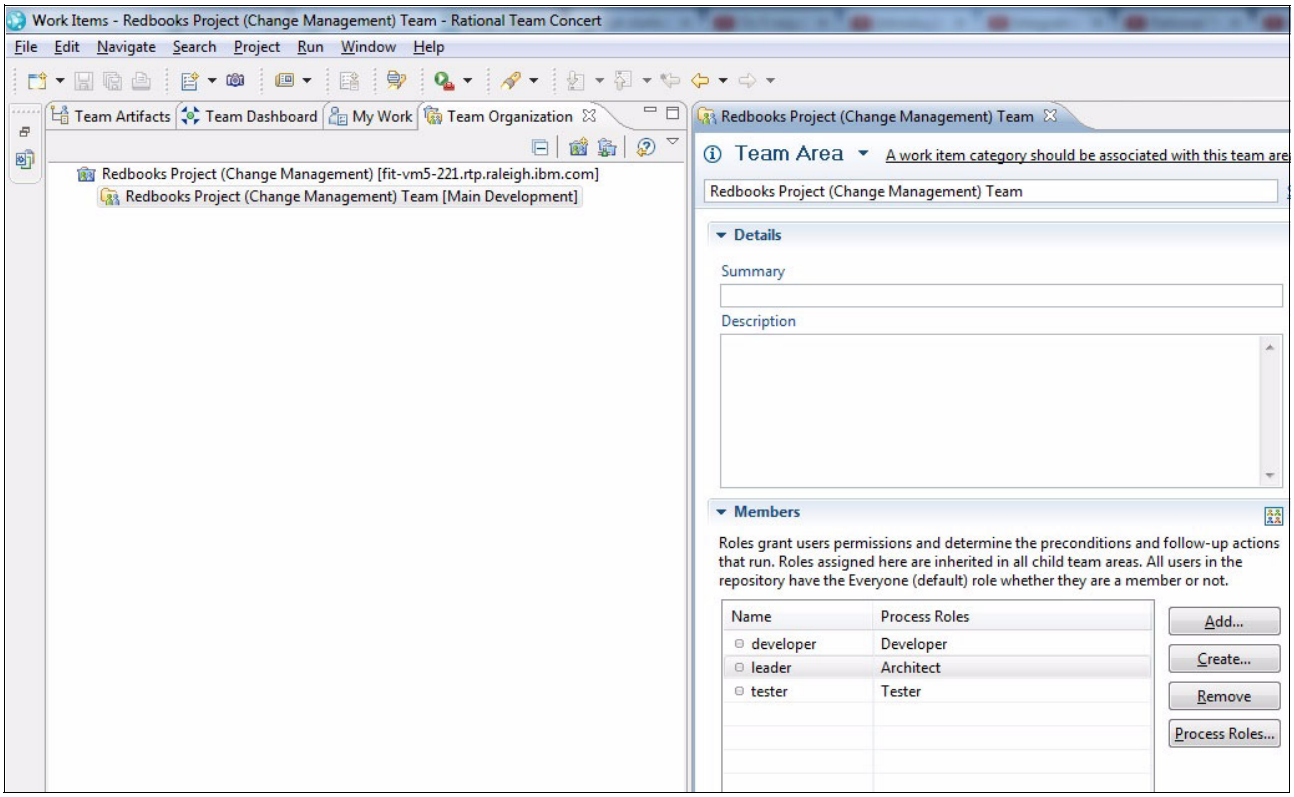


Figure 3-6 You can also configure Project Area information using Rational Team Concert Client

> **Tip:** After creating the lifecycle project, you can use the application to add users as members to each of the project areas that belong to the lifecycle project. You can customize the roles, permissions, iteration types, access control, team areas, categories, and process definitions as needed. For more information, see the following website:
>
> `http://jazz.net`

3.4 Contribute to a Collaborative Lifecycle Management project

Team members on a software development project play critical roles during the implementation of the solution. Information defined by the stakeholder is used during all lifecycle projects.

If a request is lacking the necessary level of detail, the business analyst is called upon to provide further detail to the stakeholder's request. Context is provided in the form of a business process or application sketches, storyboards, models, and other rich text content.

The next steps are assuming the following scenes:

- Stakeholder submits a Stakeholder's Requirement (we can name it "Business Need").
- The Project Team Leader requests Project Manager approval for that Stakeholder's Requirement.
- The Project Manager approves the Stakeholder's Requirements.
- The Project Manager estimates the Stakeholders Requirements and request Stakeholder approval for the estimate.

- The Stakeholder approves the estimate.
- The Project Team Leader defines and manages the requirements management.
- The Analyst specifies all Requirements.
- The Project Team Leader plans the sprints and assigns all developers activities. He also defines all Test Management effort and Releases.
- Developers coding.
- Testers testing.
- Release Engineer deploys and releases the application.

Defining and creating the requirements

Project activities begin with clearly defined requirements from the stakeholder. The Stakeholder can convey the business need by using the language and vocabulary of their business and its customers using Rational Requirements Composer. Requirements definition and management activities ensure that the business can capture and communicate their ideas to the solution team, and, at the same time, the solution team can learn the nuances and expectations of the customer.

The Stakeholder's request is reviewed to assess if it aligns with the business goals and strategy for the project team. If the request aligns with the strategy and can be contained within a given iteration, the request is added to the iteration plan. These kinds of activities allow an organization to capture the details of the business that might not be included in the request. Requirements content evolves through these efforts. It is critical that the business and solution teams create a common vision for the solution being developed. This process is typically similar to the following:

1. The Analyst organizes and manages the requirements, including capturing important requirement descriptors, such as priority, release, and origin, among others. Additionally, pertinent traceability information for related requirements can also be captured. In this case, we create all the requirements described in our Business Scenario 1.2, "The business scenario" on page 14.
2. The Stakeholder accesses the Rational Requirements Composer.
3. On the Artifacts page, click the down arrow next to the **Create** button. From the drop-down list, select **Stakeholder Requirement**, and another window displays.
4. Enter the initial content about the new artifact, then click **OK** (see Figure 3-7 on page 36).

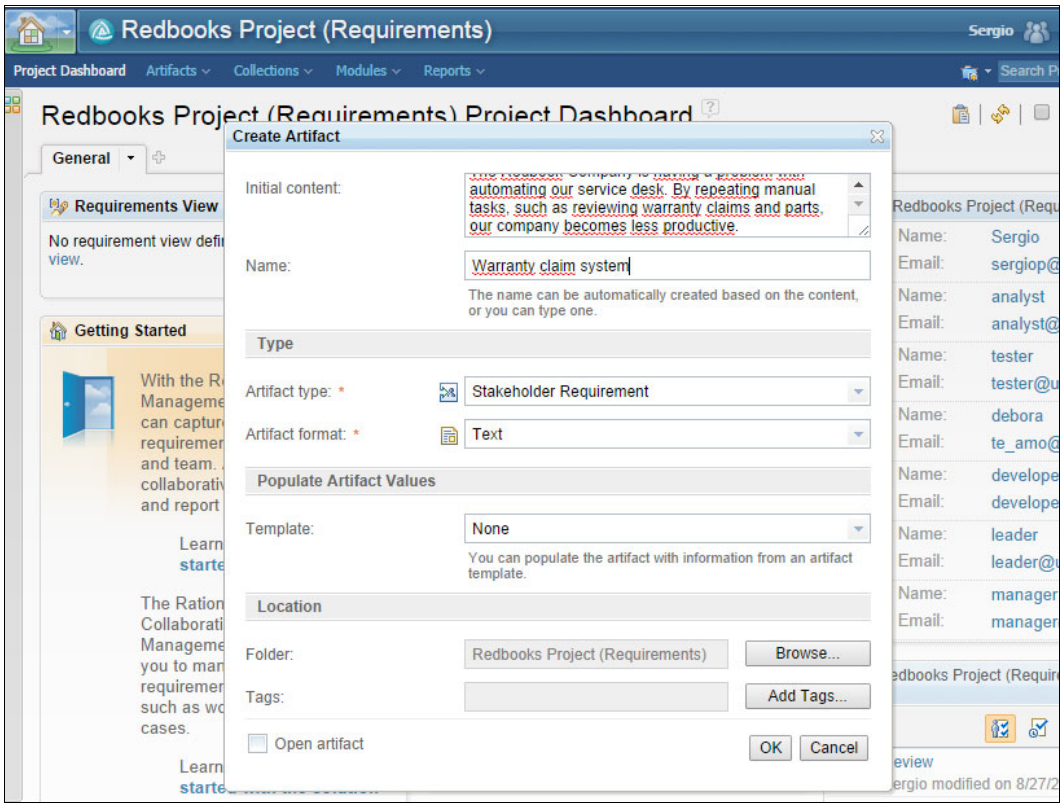


Figure 3-7 The stakeholder enters initial content

The Stakeholder can access the Requirements for latter edition by clicking the requirement on **Artifacts** > **Browse Artifacts**.

5. The Project Leader accesses the project dashboard to review the new requirements using the Dashboards Widgets (see Figure 3-8 on page 37).

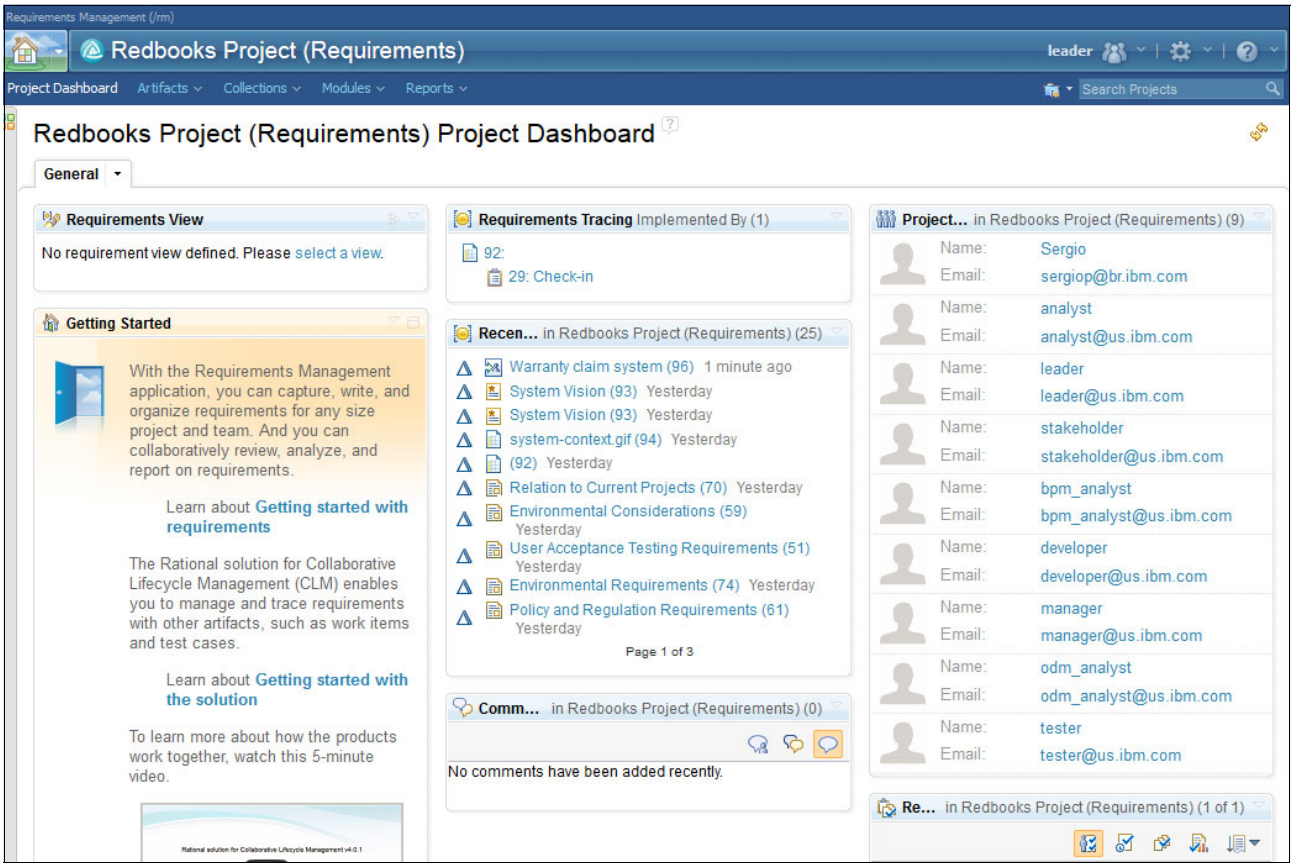


Figure 3-8 The project leader uses the Rational Requirements Composer dashboards to manage requirements

6. The Project leader reviews the stakeholder's requirements and requests an approval workflow.
7. All assigned reviewer participants review or approve the artifact (Figure 3-9 on page 38).

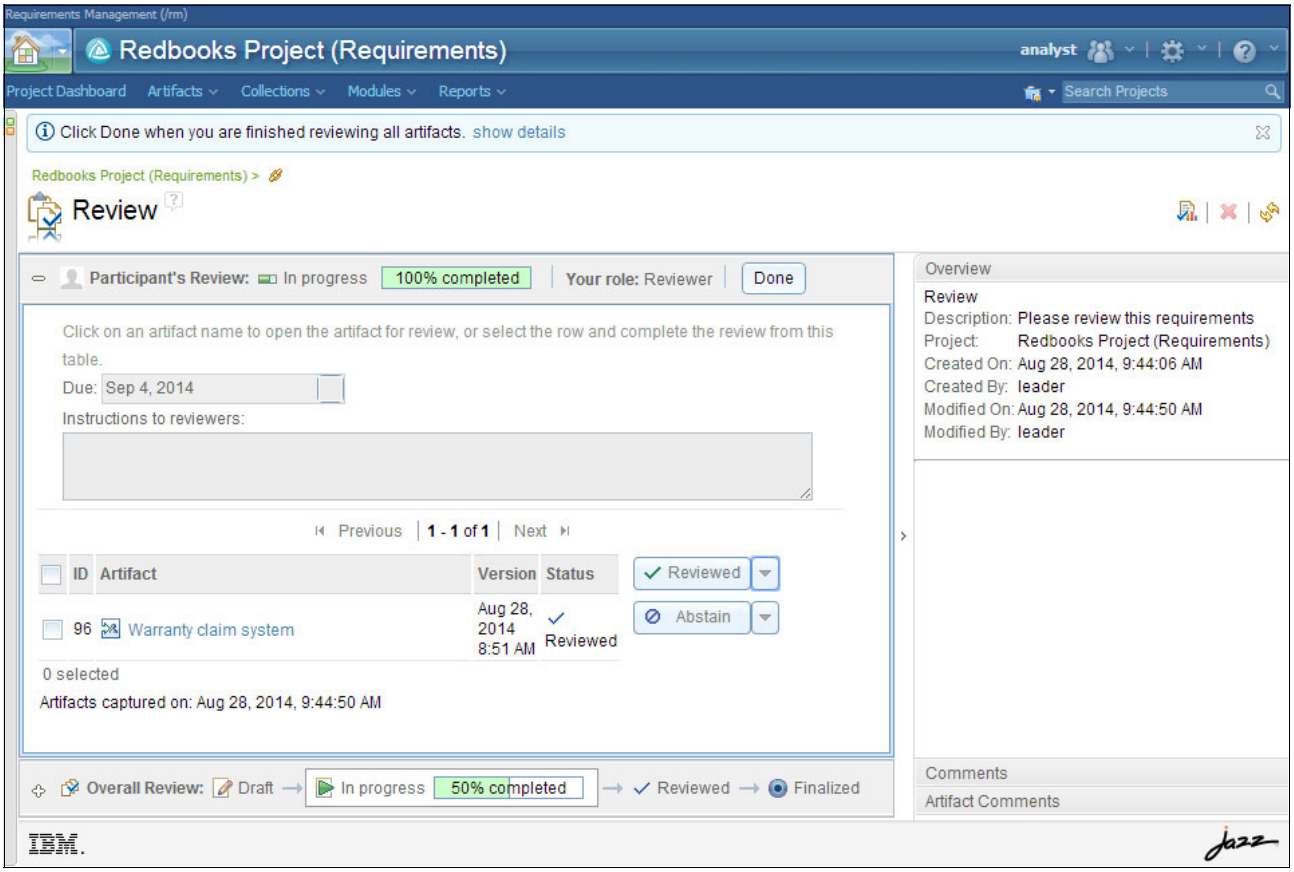


Figure 3-9 All participants use the review process for project review and approval

8. After the artifact is reviewed and approved, the Analyst collects requirement specifications and software requirements from the original stakeholder's requirements (see Figure 3-10 on page 39).

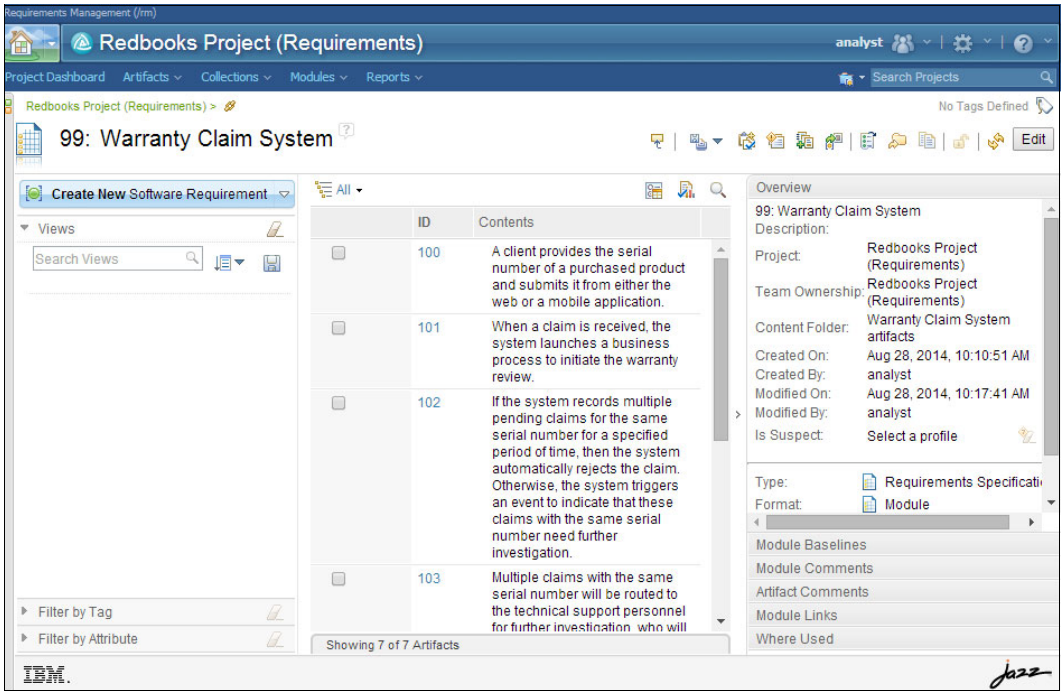


Figure 3-10 The business analyst creates the Requirements Specification

9. When the review and approval process is finished, the business analyst creates an Artifact Collection, grouping the requirements specification and the software requirements and starting the review and approval of the collection.

   Browsing a collection can help the team to confirm the requirements that are planned for a release. The artifacts can be grouped by different options to make them easier to find or to create tighter associations (see Figure 3-11).

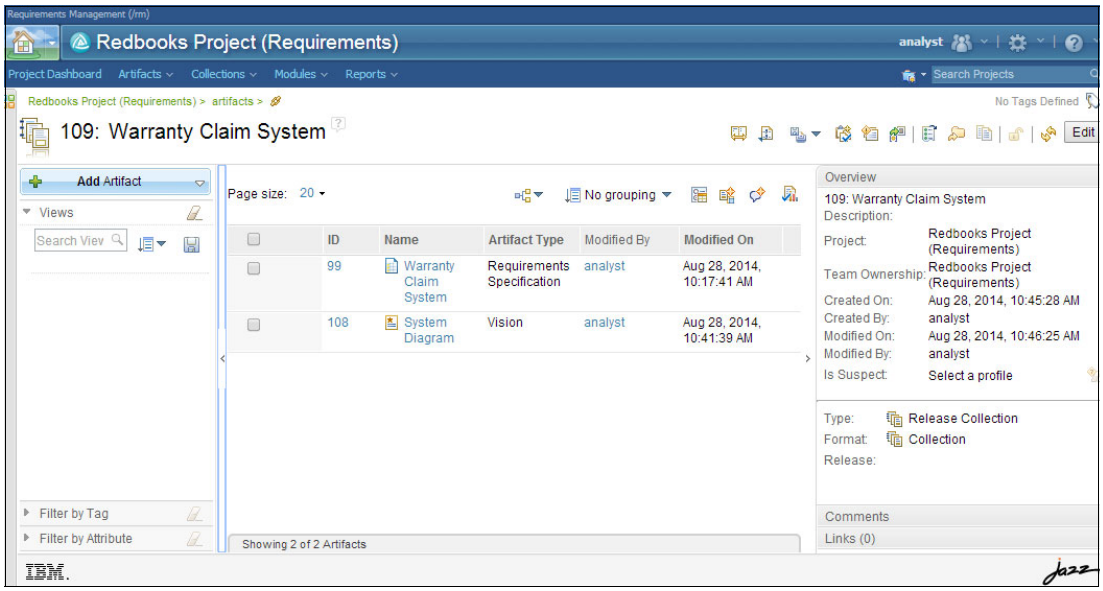


Figure 3-11 The business analyst creates a Requirements Collector to group the approved requirements

10. When the Artifact Collection is fully approved, it is used by the project leader to plan the project.

Tip: You can manage requirements by using traceability links, tags, attributes, filtering, and dashboards to create relationships between requirements and other artifacts, categorize requirements, assign properties to requirements, and monitor relationships and status among team applications.

3.4.1 Plan the project

The project leader can review the plan with the stakeholder, discuss the outcome with the team, and update the requirements accordingly.

Planning the project consists of the following phases:

1. Defining the iterations

   The iteration plan contains a list of items to be resolved. Members of the solution team collaborate about these items. As part of the planning process, activities are created and executed as part of an iteration.

Note: Collaborative Lifecycle Management tools allow you to use OpenUP on business process management projects. To learn more about OpenUp, see the following site:

http://epf.eclipse.org/wikis/openup

For development projects using the iterative development approach, organize the project into a sequence of phases or iterations. The open source OpenUP defines four phases:

- **Inception**:

  In this phase, the Team Leader, the Project Manager, and Stakeholder come to agreement on a set of requests to define the initial scope of the project. In addition, an initial architectural vision is created, and an assessment of the project team, tools, and environment is completed.

- **Elaboration**:

  This phase focuses on driving out technical risk by producing an end-to-end demonstration of the solution to prove and refine the architecture. Feature details are not addressed, but rather a “skeleton” of the system is completed to prove feasibility and identify areas of technical risk. For this book, this phase will not be executed.

- **Construction**:

  This phase involves one or more iterations that focus on completing the functionality. Each iteration involves planning, design, development, build, and testing. Agile teams advocate multiple two-week to four-week iterations, while more traditional teams might only have one iteration. This book highlights a single iteration in the Construction phase.

- **Transition**:

  After the acceptance criteria has been met, the solution moves into the *Transition phase* for stabilization testing. Unacceptable defects found in this phase cause the solution to go back to the Construction phase where the defect is fixed. The release candidate is then sent back to the Transition phase for stabilization testing until all expected tests have passed.

Each of these phases has one or more iterations.

The business scenario (described in 1.2, “The business scenario” on page 14) uses a predefined template (the OpenUp process template), a development project timeline, a release, and iterations that were created during the project creation (Figure 3-12).

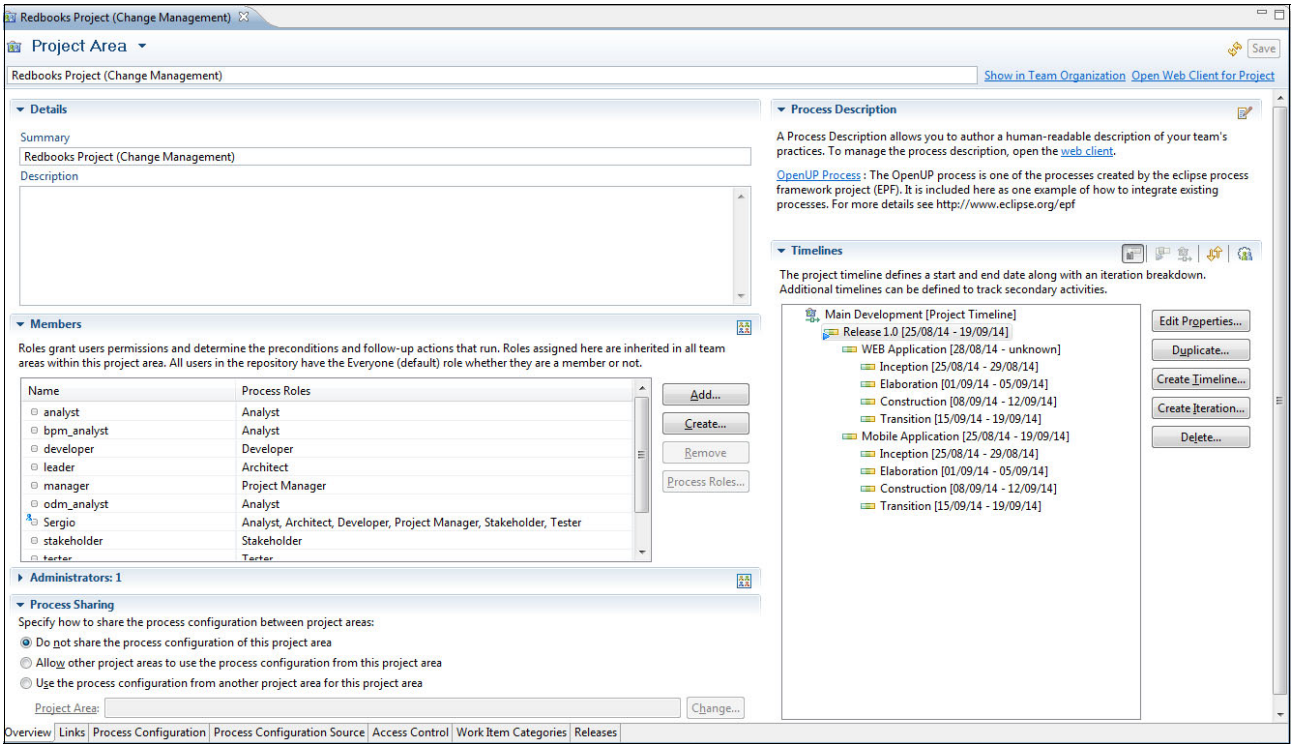


Figure 3-12 Showing the iterations and team members

You can create iterations by clicking “**Create Iteration**” on the Rational Team Concert Eclipse plug-in, Team Artifacts view.

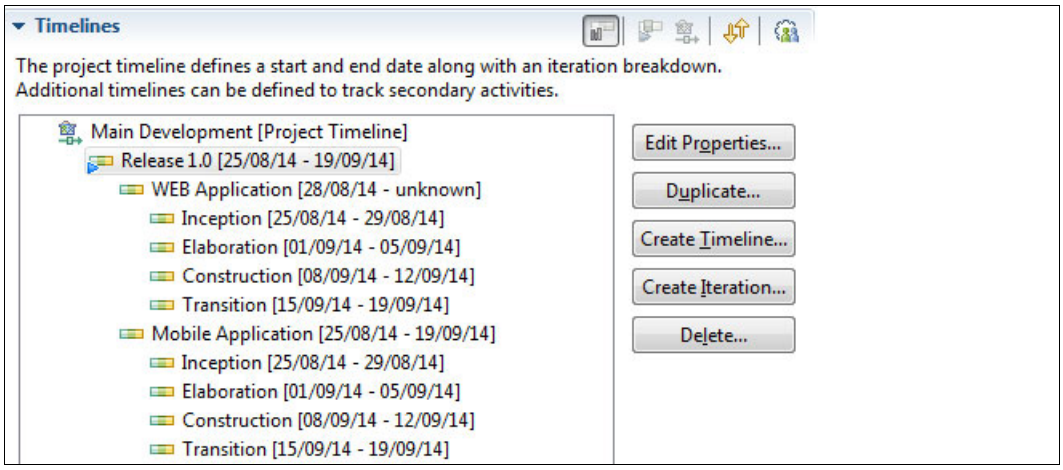


Figure 3-13 Iterations created to cover our business scenario

2. Define the release plan

   The release plan provides a high-level overview of the work associated with a product release. The plan might be for a major release and include significant development work, or for a maintenance release.

In the plan, you can view and manage work items that are targeted for the release iterations:

To create a release plan, the project leader needs to do the following:

a. In the Team Artifacts view, expand a project area.

b. Right-click the Plans folder and click **New** → **Plan**.

c. In the New Plan window, type a name for the plan, and select plan type **Release Plan**.

d. In the **Owner** field, select a project or team area.

e. If the plan is for a project area, select a timeline and an iteration. If the plan is for a team area, select a team area and an iteration.

f. Open the **Advanced Options** section.

g. Change the plan type, if needed.

h. Check the **Always load all execution items** check box.

i. Click **Next**.

j. Link the requirements collection to the release plan. Then, create work items from the requirements in the collection. The new work items are linked to the corresponding requirements:

   i. Determine the number and length of iterations.

   ii. Estimate the amount of work that the team can complete for the release.

   iii. Record the planning decisions on the release plan Overview tab.

k. Access the plan by clicking **Plan** → **All Plans**. All the plans for the business scenario in this book are presented (see Figure 3-14).

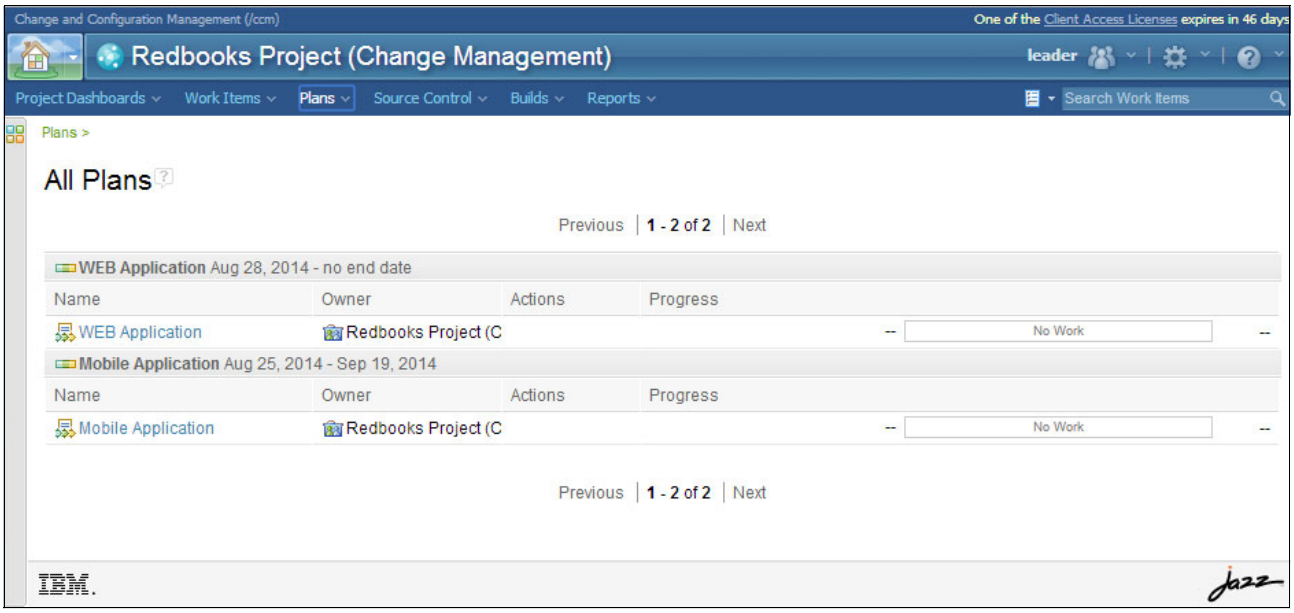


Figure 3-14 Listing all plans created on Rational Team Concert

3.4.2 Define development lifecycle activities

Now the Release Plan has been created, and it contains all of the work items associated with the release or iteration you selected. The project leader now needs to create activities:

1. Assign the activities to a Release Plan (the **Planned For** field)

2. Estimate the effort for each task (the **Estimating** field)
3. Assign them to an Owner (**Owned By** field)

   The created activities are shown in Figure 3-15.

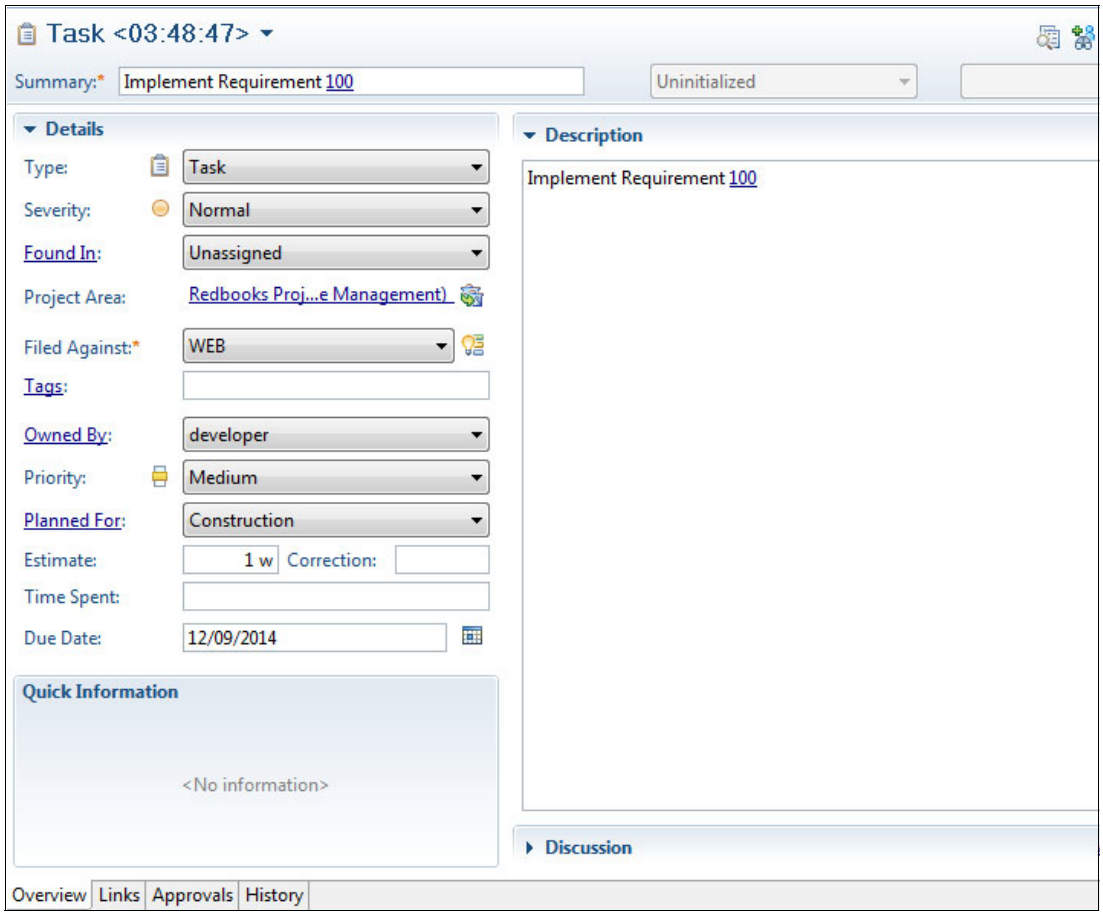


Figure 3-15 Creating tasks

4. The project leader links each task to the related Requirement or Artifact Collection. If one task contains other tasks, the project leader defines the wanted link type (for example, a child or parent link). See Figure 3-16 on page 44.

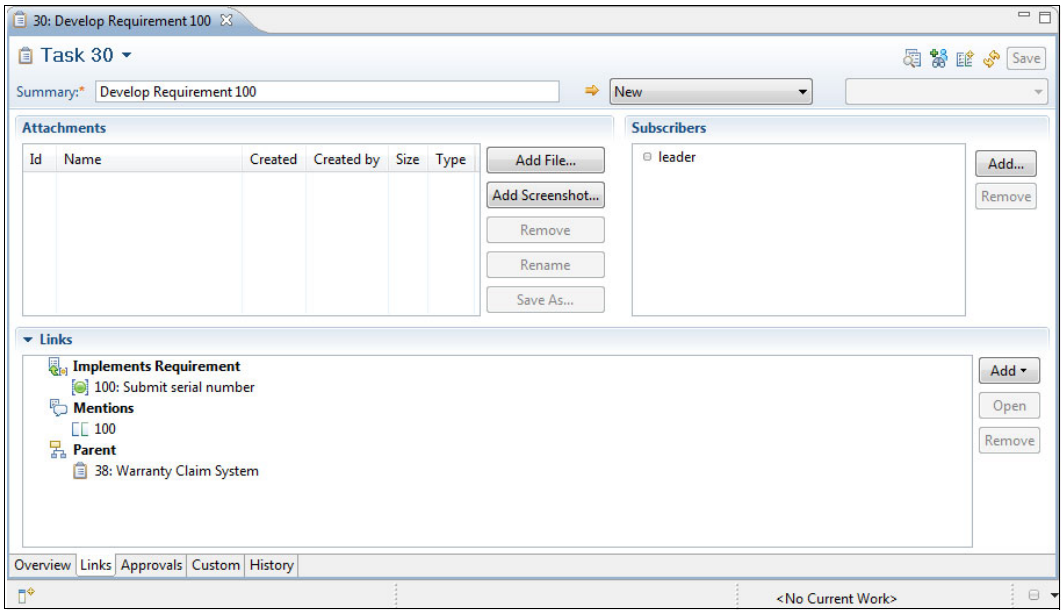


Figure 3-16 Defining links between work items

After all activities are created, the project leader places the created activities directly in the Release Plan, where Owner, Change Dates, Priorities, and so on, can be defined (see Figure 3-17).

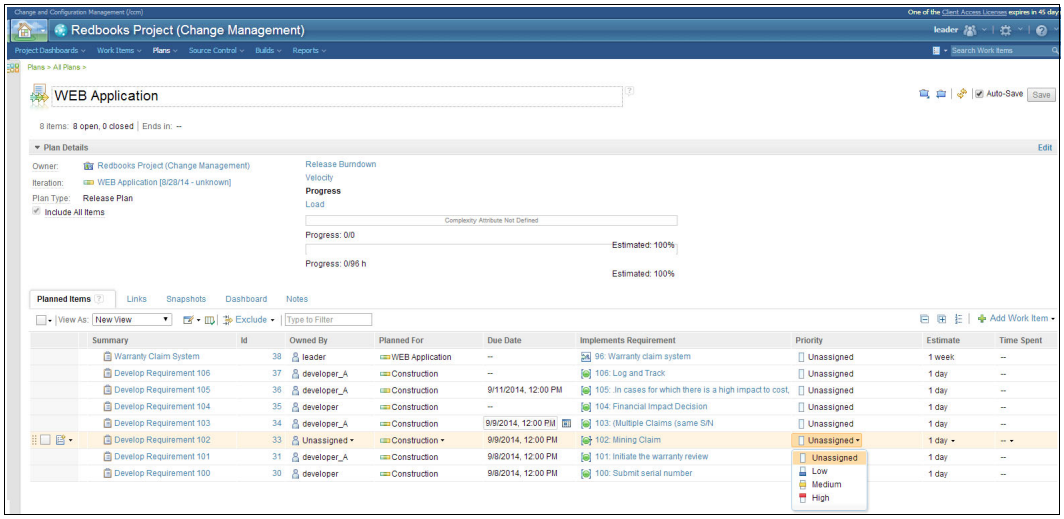


Figure 3-17 Use the Rational Team Concert Plan View to manage plan activities

3.4.3 Developers working on a Collaborative Lifecycle Management project

Developers use the approved requirements and design information to implement the solution. The developers now have an understanding of the following concerns:

- The business need and problem
- What is expected to address that need
- Potential designs or constraints that affect the implementation of the requirement

All of the planned activities have been created and assigned to developers (see Figure 3-17). The developers can now start to work on their assigned tasks. They use the Rational Team

Concert dashboard to track their activities and dependencies. The Rational Team Concert dashboard displays a real-time view of team activities (see Figure 3-18).

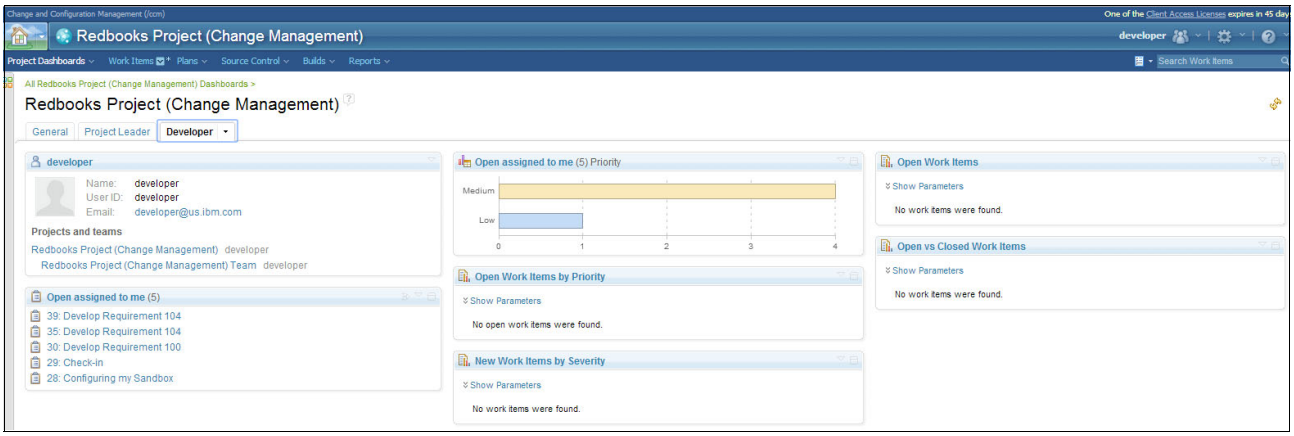


Figure 3-18 Developers can track their activities using the Rational Team Concert dashboards

Tip: You can customize Rational Team Concert Dashboards to address the need of a team. There are many Dashboard Widgets available for customization (Figure 3-19).

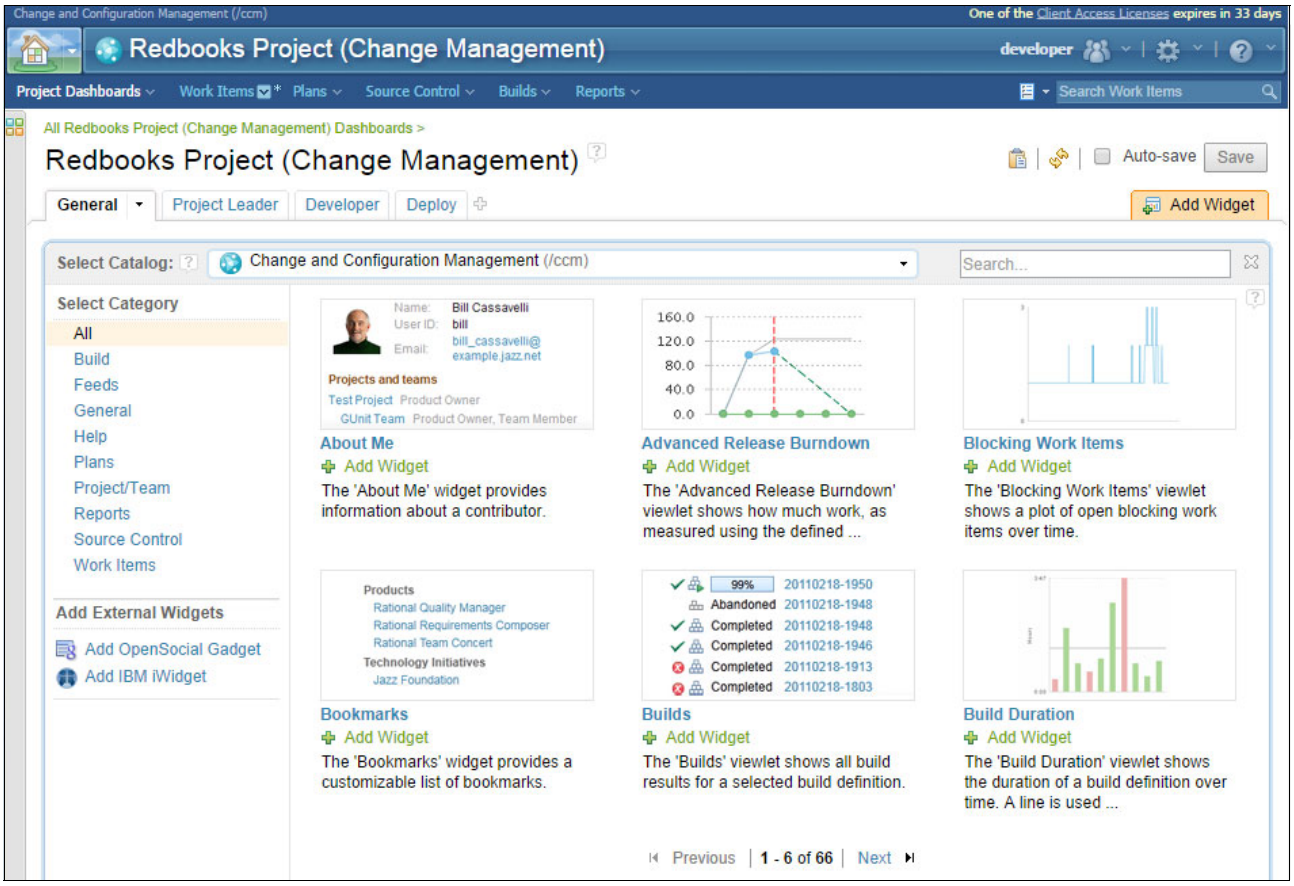


Figure 3-19 Available Rational Team Concert Dashboards Widgets

Click the wanted Task to execute it (see Figure 3-20).

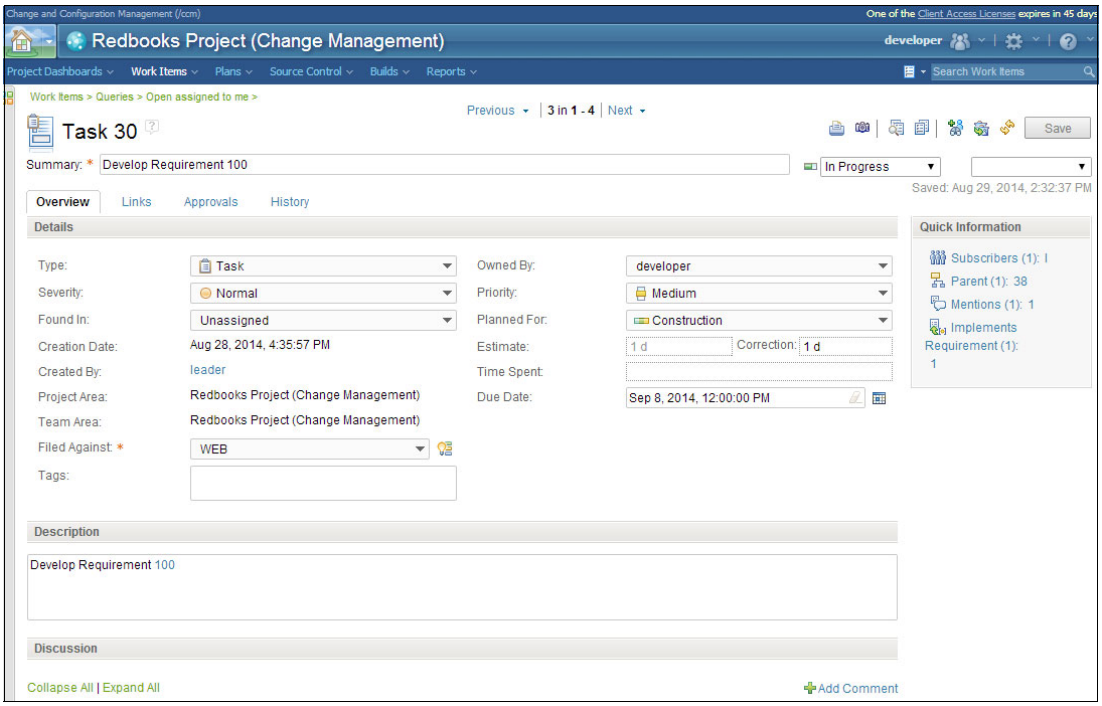


Figure 3-20 Developers can also use the Rational Team Concert web UI to manage activities

Note: If the developer uses any Eclipse integrated development environment (IDE) client for development, the client must be integrated with Rational Team Concert using the Rational Team Concert plug-in. This plug-in controls and tracks all of the activities that the developer executes, along with the assigned Rational Team Concert Workitem (task). To see how to integrate an Eclipse IDE with Rational Team Concert using this plug-in, see Appendix A, “Integrating the Eclipse environment with IBM Rational Collaborative Lifecycle Management” on page 223.

4

Configuring IBM Business Process Manager integrated with IBM WebSphere Operational Decision Manager

This solution provides detailed information about the IBM WebSphere Operational Decision Manager and the IBM Business Process Manager products. These products are part of the solution described in Chapter 2, “Solution overview” on page 19.

All of the products used in the business scenario (see 1.2, “The business scenario” on page 14) are installed as distributed systems. Also, PureApplication patterns can be used to speed up the installation process of the products as described in the IBM Redpaper publication, *Rapid deployment of integrated WebSphere solutions in your cloud*, REDP-5132, at:

http://www.redbooks.ibm.com/abstracts/redp5132.html?Open

The following topics are covered in this chapter:

- 4.1, “Decision management with WebSphere Operational Decision Manager” on page 48
- 4.2, “Managing business processes with Business Process Manager” on page 80

© Copyright IBM Corp. 2014. All rights reserved.

4.1 Decision management with WebSphere Operational Decision Manager

This section describes decision management and how it complements business process management (BPM). In addition, there is detailed guidance for implementing WebSphere Operational Decision Manager V8.0.1 within the context of a solution based on Business Process Manager V8.0.1.

4.1.1 Business drivers for decision management

Every business application, whether driven by a business process (as described in this book) or an alternative, must implement some form of business rules. Traditionally, business rules have been implemented in a programming language, such as Java or COBOL. Programming business rules into an application leads to a number of concerns and problems that arise in the development and maintenance of the application. For example:

- The rules written in a programming language are not accessible in a form that is understandable by those who define the rules, namely, the business user.
- Because some applications implement the same policies, having tightly embedded rules in one application makes it more difficult for the other applications to access these rules.
- At times, there might not be a standard for business rules, causing applications to implement them inconsistently.
- Because programming rules requires resources to develop, test, and analyze them, this process can become costly in dollars and the time to implement.
- The auditing of rule execution, to ensure that it supports business requirements, often produces results that are difficult to assess.
- Simulating new rules is often time consuming, which impacts the ability of the business to react quickly.

A decision management solution resolves these issues by providing a single, reliable source for an organization's business policies. This enables the decision management solution to streamline the development, test, simulation, and audit of business policies. We suggest that a decision management system be constructed so that non-technical business users can easily view, maintain, and test rules, often with little or no assistance from technical resources.

In this section, we use the WebSphere Operational Decision Manager as the decision management solution to define the rules for the service claim application. Figure 4-1 on page 49 describes the architecture of business rules in WebSphere Operational Decision Manager.

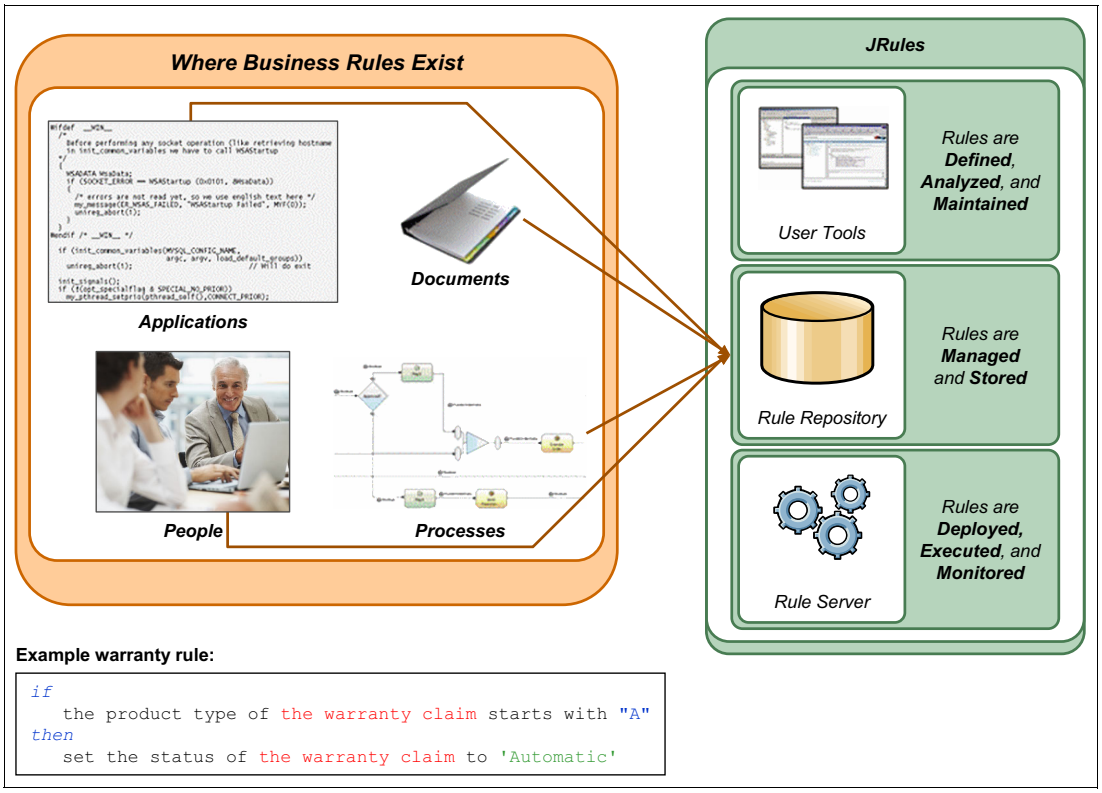


Figure 4-1 Business rules in the WebSphere Operational Decision Manager

The difference between business rules and event rules

To put the concepts of business rules and event rules in context, we describe how a company that processes service claims can manage its process using WebSphere Operational Decision Manager.

The rule shown in Example 4-1 might be implemented in a service claim system.

Example 4-1 Service claim

```
if
   the value of the service claim is greater than $1,00
then
   set manager approval to mandatory
```

As part of the same system, the service claim system might also contain a rule such as that in Example 4-2.

Example 4-2 Service claim rule

```
if
   the number of service claims of the customer within the last month is
greater than 2
then
   send a request to the call center to contact the customer
```

From a business perspective, Example 4-1 on page 49 and Example 4-2 on page 49 are both rules. More clearly:

- Example 4-1 on page 49 is a *business rule*. There is *no* reference to time in this example.
- Example 4-2 on page 49 is an *event rule*. There *is* a reference to time in this example.

From a technical, architectural perspective, this small difference impacts how each rule functions:

- A *business rule* application module:
  - Receives data synchronously.
  - Responds synchronously.
  - Is stateless.
- An *event rule* application module:
  - Is called asynchronously.
  - Might trigger an asynchronous response.
  - Is stateful if the rule correlates the current event with any previous events.

These technical differences are discussed further in 4.1.3, “WebSphere Operational Decision Manager” on page 53.

4.1.2 Decision management with BPM

Decision management does not and cannot function in isolation. The decision management runtime is a back-end service that interacts with other applications to add value. It is possible to add decision management to any application, however, it is best applied in business process applications.

In many enterprises, BPM and decision management are complementary capabilities. Simply put:

- A business process tells you what to do.
- Decision management tells you how to do it.

Sometimes organizations attempt to map business processes without using decision management. If the decisions that drive these processes are simple and infrequent, and if those decisions do not interact with other systems, it is possible to embed your decision logic within a process. However, implementing complex decisions within a business process without some form of decision management can lead your processes to become large, complex, and difficult to understand and maintain, as shown in Figure 4-2 on page 51.

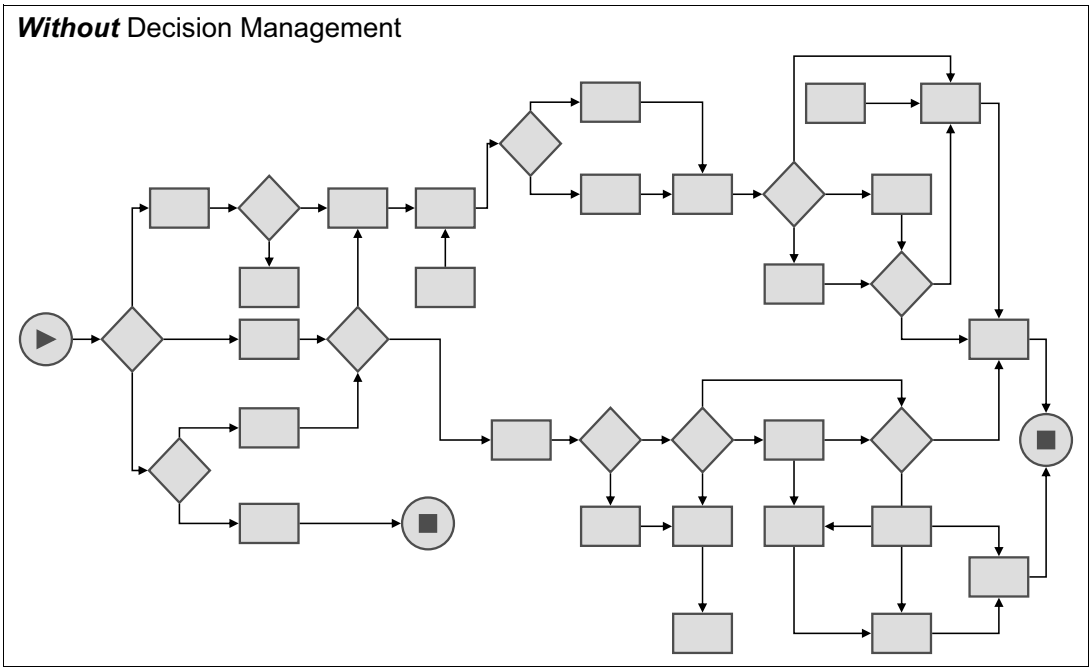


Figure 4-2 Complex business process with embedded decision logic

Adding decision management to the process can help remove the complexity of the process by simplifying parts of it that are decision services. At the same time, removing the decision logic from the process enables that logic to be reused elsewhere in the enterprise. Figure 4-3 on page 52 shows that it is possible to extract the decision services portions of a business process to create a decision management solution. As you can see, this new solution calls on three decision services:

- Eligibility
- Risk scoring
- Offers and promotions

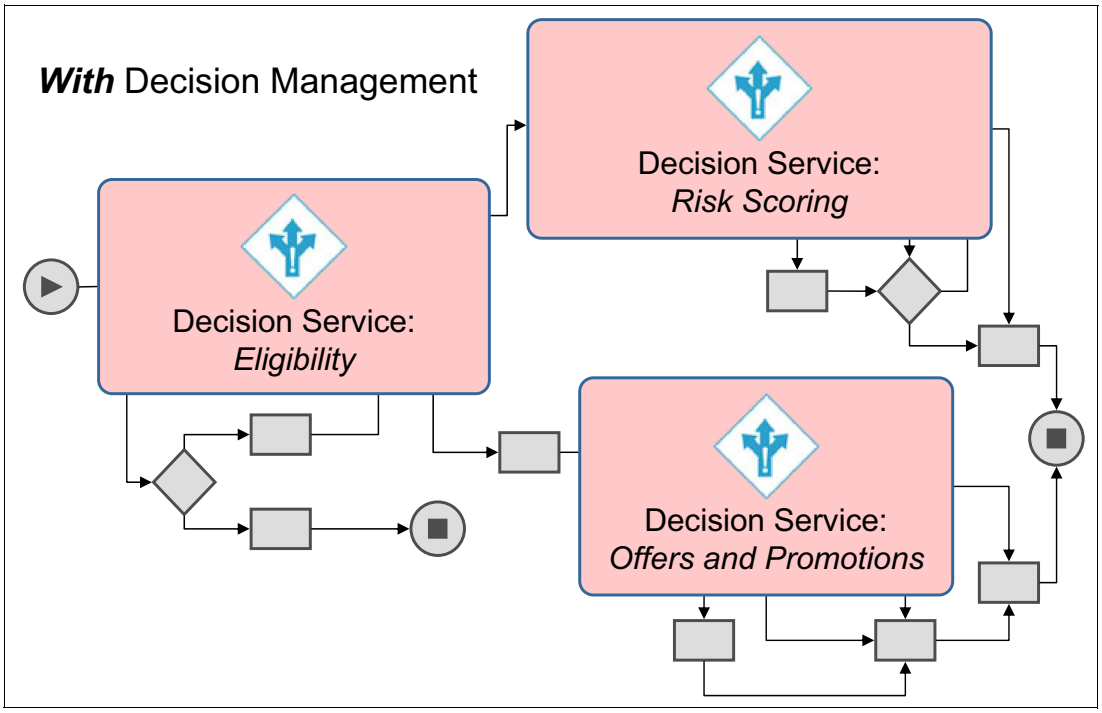


Figure 4-3 Business process simplified by extracting decision logic

Decision management usage patterns

Decision management can be used in the context of other applications or business processes in various ways. Examples throughout this book show a number of the most common usage patterns for how decision management can be used in the context of BPM.

Business process calling business rules

Business rules are typically called from a process, in the same way that any external, stateless service is called. That is, the process sends data to the decision service and receives a synchronous response. Because the decision service is stateless, all of the information required by the decision service must be passed from the process. A number of design decisions must be made when using this pattern, such as what degree of processing is to be done within the business process, and what will be done within the rules. As a general principle, minimizing the number of calls between the business process and the decision service is desirable because this minimizes performance hits. The design also focuses on keeping activities that are primarily process focused, such as human workflow and system interactions, within the business process and those that are decision focused within the decision service. Figure 4-4 on page 53 shows an example of how a business process can interact with the WebSphere Decision Server at runtime.

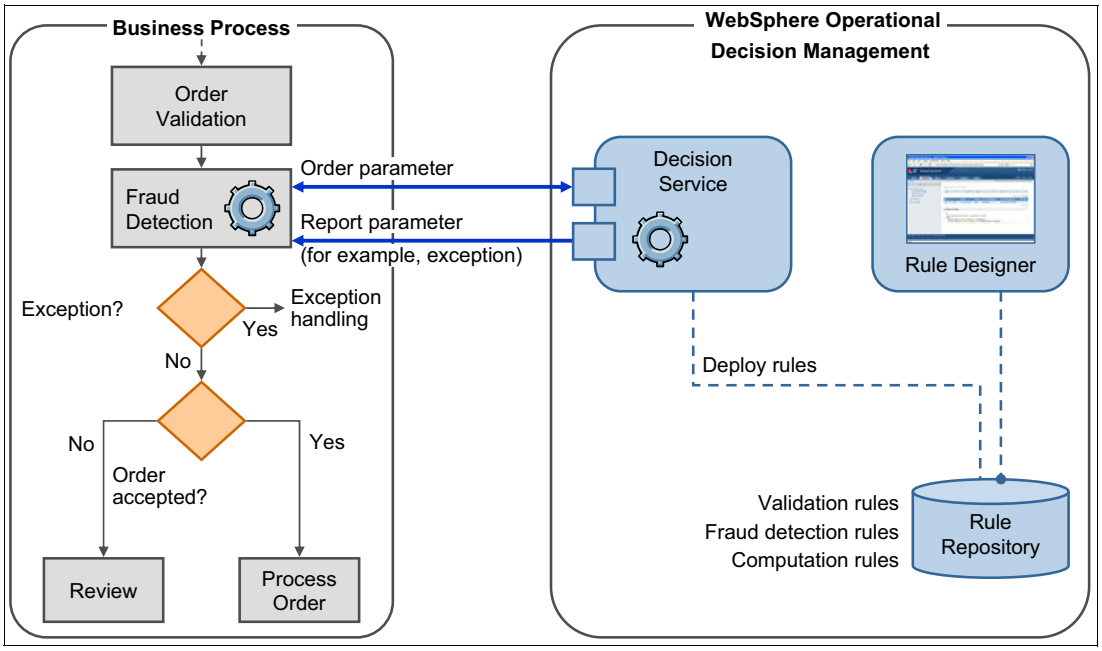


Figure 4-4 How rules and business processes interact

Triggering a process based on a business situation

Business events can be used to detect various situations (that is, conditions, such as possible fraud or a new opportunity) that require prompt action. It is common for any follow-up action resulting from such a business event to be a process. Although other patterns exist, this sense-and-respond pattern is quite common, which is why we selected it for our warranty application. This pattern is shown in Figure 4-5.

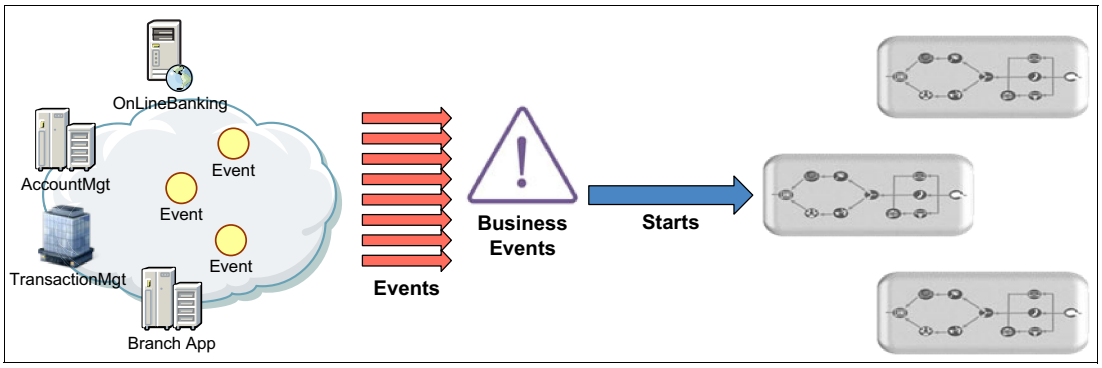


Figure 4-5 Common sense-and-respond pattern

4.1.3 WebSphere Operational Decision Manager

WebSphere Operational Decision Manager V8.0.1, the next generation of business rules technology, delivers comprehensive automation and governance of operational decisions that control the actions of critical business systems.

It enables organizations to:

- Automate, govern, and improve operational decision making across business processes and applications for better business outcomes.

- Make more profitable decisions with real-time detection of opportunities and risks.
- Implement changes easily, reliably, and securely to meet new market demands and policy requirements.

This section provides an overview of the components in the product and how each component works.

Figure 4-6 shows the components that make up WebSphere Operational Decision Manager V8.0.1.

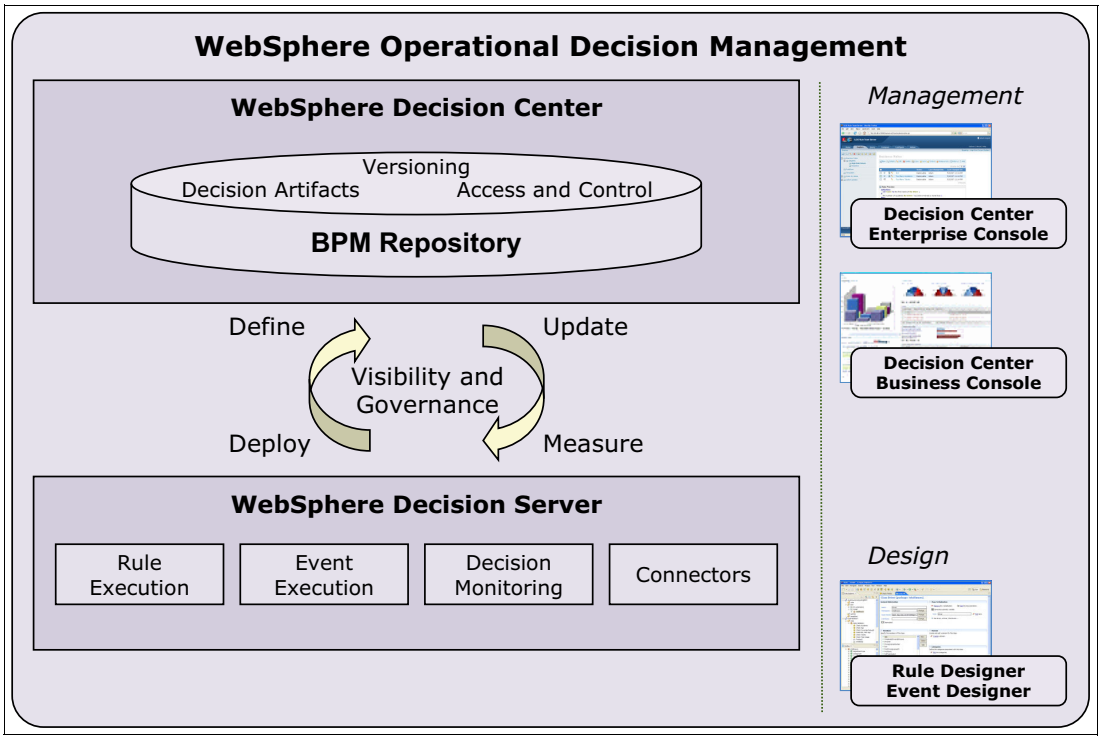


Figure 4-6 WebSphere Operational Decision Manager v8.0.1

These components are described in more detail in the following sections.

The WebSphere Decision Server

The WebSphere Decision Server is the runtime component where business rules and event applications are installed and executed. Typically, the WebSphere Decision Server is hosted on a Java based application server, and is packaged with WebSphere Application Server. You can also install and run it on other application servers or without an application server, albeit with limited capabilities.

The WebSphere Decision Center

The WebSphere Decision Center is a repository that contains the source of the rules. As such, it provides the governance capabilities and interfaces that enable business users to work directly with rules. WebSphere Operational Decision Manager might be used without using WebSphere Decision Center. However, doing so means that business users will not be able to manage and change rules directly in the tool.

Both the WebSphere Decision Center business console and the enterprise console enable business users to manage rules directly in WebSphere Operational Decision Manager. Each console is summarized in the next two sections.

Decision Center business console

This console provides a socially aware, web-based interface that enables business users to author, edit, organize, and search for rules in a collaborative environment. It is meant to coexist with the enterprise console, and offers the following features:

- Simplified navigation to quickly access the projects and elements you want to work on.
- Simplified editing of action rules and decision tables.
- You can see what changes have been made to projects of interest to you.
- You can see and post comments about recent changes made by other users, and you can attach files to these posts.
- Simplified search, available from a search box.
- Take snapshots of the current state of a project or its state at the time of a previous change.

> **Note:** The term *snapshot* is used in the business console and corresponds to the term *baselines* in the enterprise console.

The WebSphere Decision Center enterprise console

This console is more sophisticated than the business console because it provides support for users who are involved with the day-to-day management and testing of rules, such as:

- Managing branches
- Project settings
- Deployment, security, and repository administration
- Creating queries
- Project reports and rule analysis
- Testing and simulation
- Creating templates, variable sets, functions, technical rules, and resources

The WebSphere Decision Center Rule Designer and Event Designer

The WebSphere Decision Center Rule Designer and Event Designer are the development environments for rule and event applications. The following sections summarize each tool and describe the main activities that are required to make and deploy a project using each tool.

Rule Designer

The Rule Designer is an Eclipse-based integrated development environment (IDE) for developing business rule applications. It is used by a rules developer who has Java development experience and business rule development experience.

The highest level organizational unit in the Rule Designer is a Rule Project, which consists of:

- An eXecutable Object Model (XOM) that represents the basic data structures upon which rules will be created.
- A Business Object Model (BOM) that maps the business vocabulary to the XOM.
- Rules that exist in one of the following forms:
  - Business action language (BAL) rules, which are formatted as IF - THEN statements and are constructed using the vocabulary described in the BOM.
  - Decision tables, which consist of one or more input columns and one or more action columns. Typically, these are used instead of BAL rules when there are large numbers of rules that are similar in structure and only vary based on the parameters.

  - A decision tree of the rules that have built-in decision branches for dealing with more complex decisions than can be easily represented by a decision table.
- Ruleflows confirm the order in which rules must execute, and they might contain branching logic. There must be at least one ruleflow in a project (for example, the main ruleflow) because this is the entry point for rule execution.

After these items are developed, the Rule Project needs to be deployed. First, a RuleApp needs to be created that references the Rule Projects that need to be deployed. Then, the RuleApp is deployed to the WebSphere Decision Server run time.

Event Designer

The Event Designer is an Eclipse-based IDE for developing situational awareness applications. It is used by an event developer who has Java development experience and complex event processing experience.

The highest level organizational unit in the Event Designer is an Event Project that consists of:

- *Event:* A message that is to be processed and can come from many sources, such as a database, a file system, a JMS queue, or a web service.
- *Event objects:* Describe the data that is associated with the event. The event object is used to populate data inside a business object.
- *Business objects:* Temporary objects that are created at the time of an event to evaluate whether that event matches any event rules. Business objects can be made persistent, thus enabling rules to correlate events that happen over periods of time.
- *Event rule:* These are IF - THEN statements written in BAL and based on the vocabulary described in the business objects. Event rules enable patterns of events to be matched.
- *Action:* An outgoing message that can be triggered upon the success of an event rule. It can send a message to many different channels, such as a database, a file, a JMS queue, or a web service.
- *Action object:* Describes the data inside the business objects that are associated with an event.

4.1.4 Overview of business rules

Business rules can be represented in many forms, such as in a programming language or in company policy documents (in written form). When derived from written forms, business rules can be centralized and automated, which allows them to be read and understood by someone without having computer programming knowledge or experience. WebSphere Operational Decision Manager combines these approaches to allow business users to create rules in structured English (or for that matter, in any other human-readable language that can be represented in the Unicode Transformation Formats (UTF) character set), and then to deploy them so that they can be executed automatically. The following sections explain how to author and deploy business rules.

Building a Business Rule Project

Rule authoring is performed in the Rule Designer. It consists of a series of sequential steps described as follows:

Create a Java project with an XOM

The XOM is the base representation of the data structure that will govern the rules. For example, if a company is writing rules to process service claims, a data structure representing a service claim is needed. This data structure can be represented with one of the following:

- Java classes
- XML schema

For the service claim application, the XOM that is using Java classes is created using the steps below. After launching the Rule Designer, switch to the Java perspective by selecting it in the upper right corner of the Rule Designer, as shown in Figure 4-7.

Figure 4-7 Switching to the Java perspective in the Rule Designer

Next, a Java project for holding your Java classes needs to be created:

1. Select **File** → **New** → **Java Project** from the toolbar.
2. Name your project `ServiceClaimsXOM`, and then click **Finish**.
3. To create a Java class, select **File** → **New** → **Class**.
4. Complete the dialog box for the class description, as shown in Figure 4-8 on page 58, and click **Finish**.

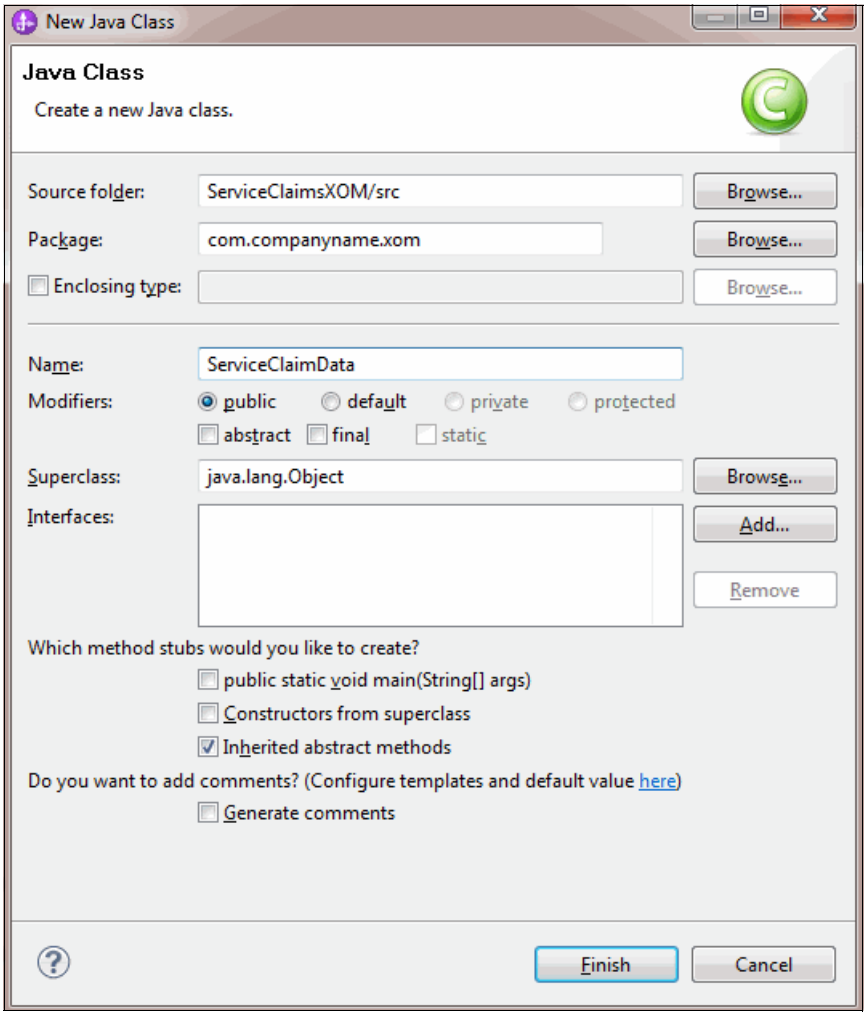


Figure 4-8 Creating a Java Class for the executable object model

5. Create the private fields for the class, as shown in the top part of Figure 4-9.

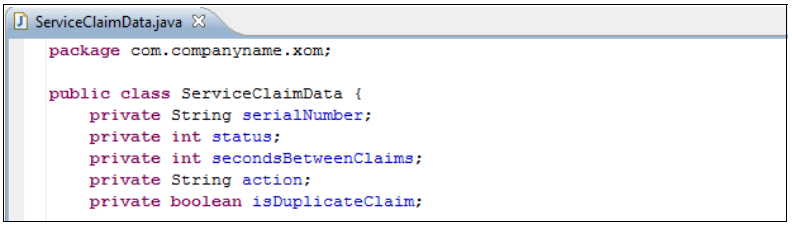


Figure 4-9 Editing the Java class for the executable object model

6. The get and set methods for each field can be typed in directly, or you can generate them using the **Source** → **Generate Getters and Setters** option.

Your Java project now contains the XOM.

Create a Rule Project with a BOM

Next, a BOM is created. The BOM is the human-readable form of the XOM that is used to write rules. BOM is automatically constructed from the XOM when you create the Rule Project. Switch to the Rule perspective, as shown in Figure 4-10 on page 59.

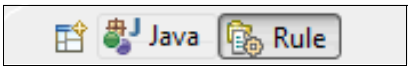


Figure 4-10 Switching to the Rule perspective in Rule Designer

To create a Rule Project with a BOM:

1. Select **File** → **New** → **Rule Project** from the toolbar**.**
2. Select the **Rule Project with a BOM** option and click **Next.**
3. Name the project `ServiceClaimsRules` and click **Next.**
4. In the New Rule Project dialog, select the **ServiceClaimsXOM** project that was created in "Create a Java project with an XOM" on page 57, and click **Finish** when done, as shown in Figure 4-11.

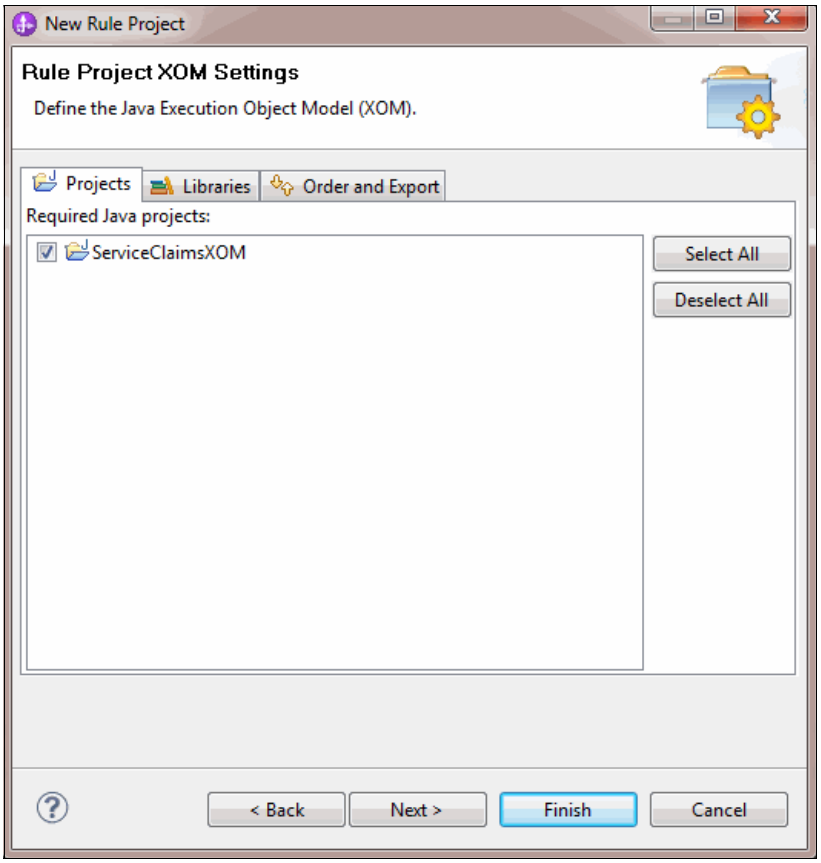


Figure 4-11 Attaching the executable object model to a Rule Project

The Rule Project containing the BOM is created.

To understand where you are in the process of creating a functioning Rule Project, you can switch to the tabbed pane at the bottom of Rule Designer, labeled *Rule Project Map*. This shows a roadmap that indicates which elements in the Rule Project have been created and which ones remain. The diagram will look similar to the one shown in Figure 4-12 on page 60.

Tip: The Rule Project must be selected in the Rule Explorer pane for the Rule Project Map to be visible.

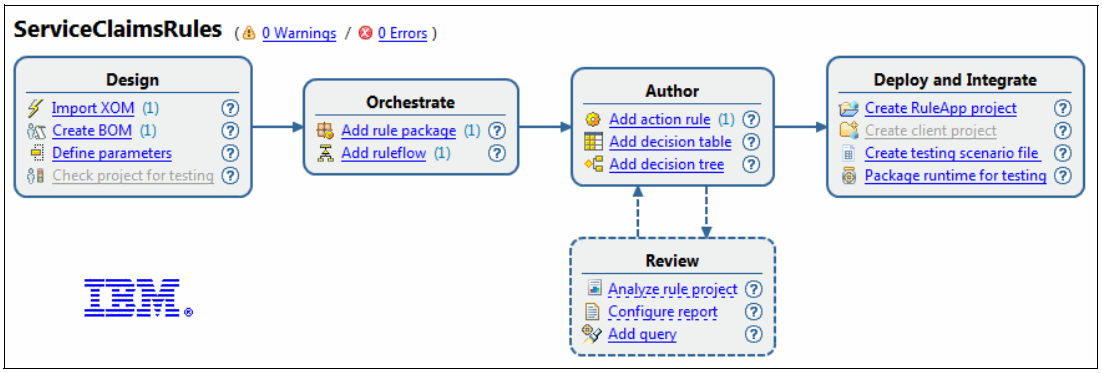


Figure 4-12 Overview of the steps involved in creating a Rule Project

Notice that there are numbers in parentheses next to some items, indicating how many of these elements have been completed. So far, we have imported XOM and created the BOM.

Define ruleset parameters

Before any rules can be written, the rule inputs and outputs need to be defined first:

1. Click **Define parameters** in the Rule Project Map window at the bottom of the Rule Designer.
2. Complete the parameters as shown in Figure 4-13.

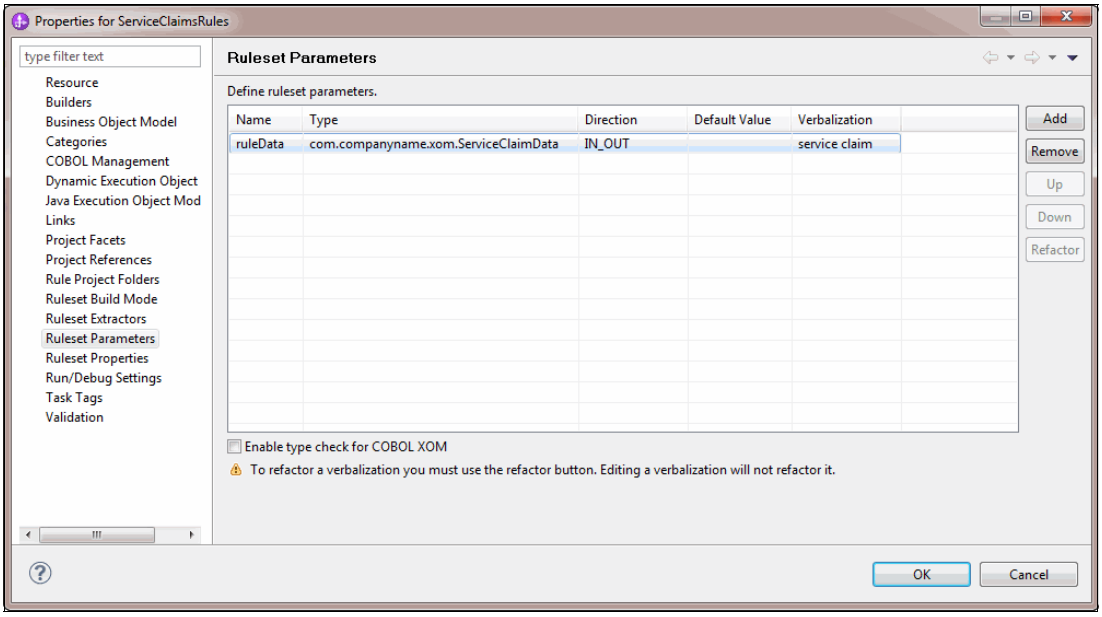


Figure 4-13 Defining input and output parameters for a Rule Project

With the parameters defined as shown in Figure 4-13, the Ruleset Parameters are complete. Notice that the parameter is IN/OUT (means input and output parameter) and that the data type of the ruleset is the XOM that was created in the previous section. The parameter is populated according to the rule that will be created in “Create a rule” on page 61.

Create a rule

Now it is time to create rules. To create a simple IF - THEN rule:

1. Select **File** → **New** → **Action Rule**.
2. Name your rule `partCostStatus` and click **Finish**.
3. Create the rule as shown in Figure 4-14.

```
if
    the serial number of 'service claim' starts with "S"
then
    set the action of 'service claim' to "Attention";
    print "Need Financial Manager attention";
else
    set the action of 'service claim' to "Proceed" ;
    print "No attention needed for Financial manager";
```

Figure 4-14 The service claim part cost rule

4. Press **Ctrl+S** to save the rule.

In Figure 4-14, the rule sets the action of the service claim to Attention status if the serial number starts with the letter *S*; otherwise, the action is set to Proceed. The print function was enabled for debugging purposes to ensure that the rule executes correctly.

Rule execution

Before a rule can be executed, the WebSphere Decision Server must be running, and the rules must be deployed. It is not possible to deploy a Rule Project directly; instead, a RuleApp project must be created that references one or more Rule Projects, and you deploy it. After the RuleApp is deployed, test it to ensure that it functions as intended.

Rule deployment

To deploy a Rule Project:

1. Start the WebSphere Decision Server by clicking **Start the server** of the **ODMSample8010** profile that is configured on WebSphere Application Server v8.0**.** (The first time this is done, it takes a few minutes for the server profile to be created.)
2. Select **File** → **New** → **Project** from the toolbar, and then select **RuleApp Project** to create a new RuleApp.
3. Name the project `ServiceClaimsRuleApp` and click **Next.**
4. Click **Add**, select **ServiceClaimsRules**, click **OK**, and click **Next** when done**.**
5. Click **Finish** to create your RuleApp project, which references the Rule project and is now ready to be deployed on the server.
6. Ensure that the WebSphere Decision Server has started by checking its status in the Command Prompt window that opened when the server was launched.
7. In the Rule Explorer view, right-click **ServiceClaimsRuleApp** and choose **RuleApp** → **Deploy,** as shown in Figure 4-15 on page 62.

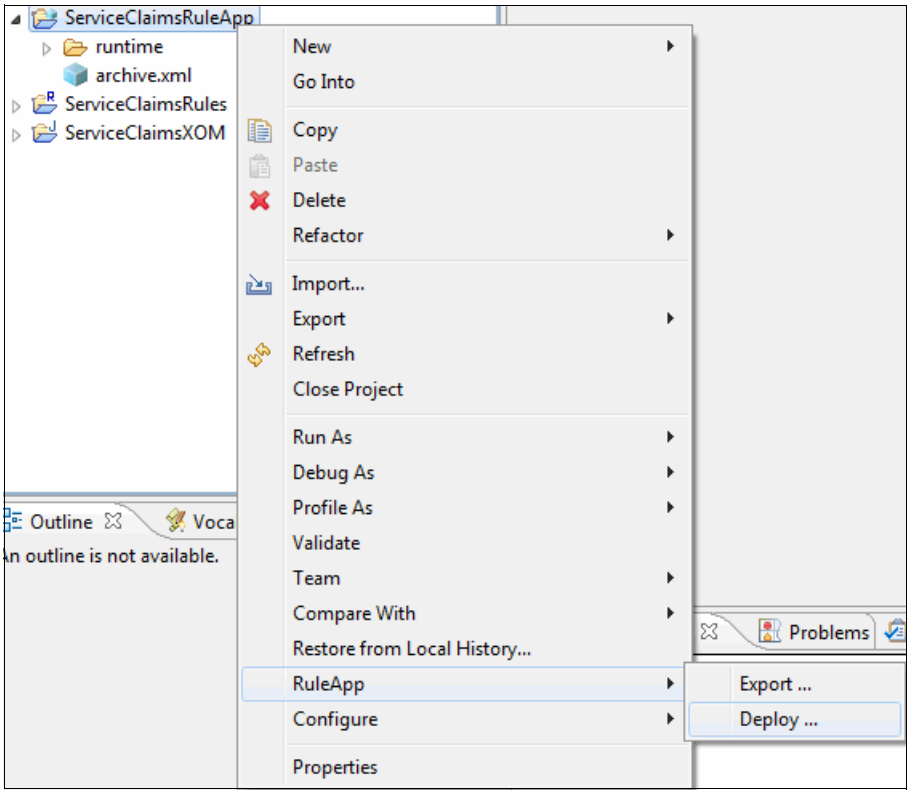


Figure 4-15 Creating a new RuleApp in Rule Designer

8. Click **Next** when asked about the RuleApp version, as this step is not important at this stage.
9. Enter the details for your WebSphere Decision Server. Typically, the default values are as follows:
   - URL: `http://localhost:9080/res`
   - Login: `resAdmin`
   - Password: `resAdmin`
10. Click **Finish** to initiate the deployment.

If the RuleApp deploys successfully, the Console tab in Rule Designer will look like Figure 4-16.

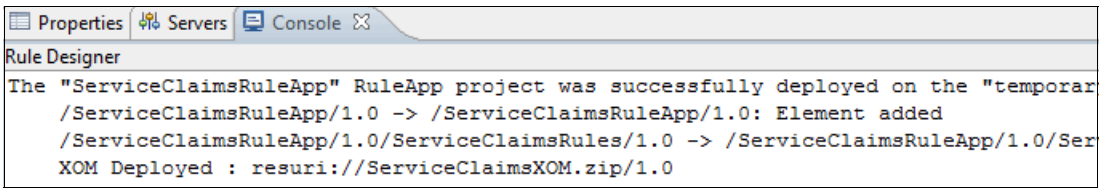


Figure 4-16 Rule Designer console showing successful deployment of a RuleApp

For security reasons, the deployment process may be different in a production environment. For details, see the link provided in the note below.

Note: For guidance about how to deploy business rules, see *Deploying business rules* at:

`http://pic.dhe.ibm.com/infocenter/dmanager/v8r0m1/topic/com.ibm.wodm.dserver.rules.deploying/topics/wodm_dserver_rules_deploying.html`

Rule testing

Now, test the rule that was just deployed:

1. Select **Run** → **Run Configurations** from the Rule Designer toolbar.
2. Click **ServiceClaimsRules** under Rule Project, and select the **Parameters & Arguments** tab.
3. Click **ruleData** and press **Edit Value**.
4. Edit the parameter value so that it appears as shown in Figure 4-17.

Figure 4-17 Creating a test object in the Rule Designer

5. Click **OK**, and then click **Run** to test your rule.

The rule will execute, and if it completes successfully, your Console tab in Rule Designer displays the *Need Financial Manager attention* message. This is the result of the debug statement that indicates the serial number that is given needs a Financial Manager's attention.

> **Note:** The Business Rule project is created and it is assumed that you share the created projects in Rational Team Concert. Visit Appendix A, "Integrating the Eclipse environment with IBM Rational Collaborative Lifecycle Management" on page 223.

4.1.5 Overview of event rules

Event rules are similar in many ways to the business rules that are described in 4.1.4, "Overview of business rules" on page 56. The key difference between them is that event rules always have a time component, which can be phrases such as *occurred within the last week* or *did not occur within the last 2 hours.*

Table 4-1 on page 64 shows the various artifacts that make up the service claim project.

Table 4-1 Artifacts of the Event Rule Project

Name	Type	Description
`EventReceived`	Event	The incoming event received from a web service, which indicates a single service claim has been made.
`EventReceivedObject`	Event object	A description of the data in the `EventReceived` event.
`ServiceClaimsBusinessObject`	Business object	An object that the event object data copies and enriches when the event rule is evaluated.
`ServiceClaimsHistoryBusinessObject`	Business object	An object that accumulates historical events for users to correlate over time.
`StartServiceClaimsEventInBPM`	Action	The action generated by importing the Web Services Description Language (WSDL) of the BPM process, which starts the service claim process.
`StartServiceClaimsEventInBPMRequest`	Action object	A description of the data associated with the `StartServiceClaimsEventInBPM` action that will be sent to the process.
`MultipleServiceClaimsEvents`	Event rule	An event rule that detects a number of claims for the same serial number, if the service claims are processes in quick succession.

Building an Event Project

The Event Designer is a component of the WebSphere Decision Server. The Event Designer supports the definition of the metadata layer required for business event processing (BEP). The Event Designer can be used to create all of the building blocks for the application, including events, business objects, actions, and event rules.

This section highlights the sequential steps for creating executable rules using the Event Designer.

Create an Event Project

The Event Project for the sample service claim application is tied to the business process described in 4.2, “Managing business processes with Business Process Manager” on page 80. This business process defines a web service interface that is used by our Event Project.

Note: It is important to save any item that is created whenever it is changed. The project will not synchronize until a save is done, and if you attempt one without first saving your work, many errors might result.

To create a new Event Project based on an existing web service:

1. Select **File** → **New** → **Event Project** from the toolbar.
2. Name your project `ServiceClaimsEvents`, and click **Finish**.

Note: You will receive an error stating that “At least one field must have a definition.” You will learn how to resolve this error in “Map the Event object to the Business object” on page 69.

Since Operational Decision Management and Business Process Management are tightly coupled, creating the business process needs to be done in parallel with creating the rules, so that the WSDL can be generated from the process to import into an event rule. Import the WSDL file for the web service that will invoke the business process when multiple service claims are detected:

1. Right-click the newly created **`ServiceClaimsEvents`** and select **Import**.
2. In the import dialog, expand **Event Designer** and select **WSDL into Event project**.
3. Click **Next** and enter the following address to specify where to import the WSDL from:

   `http://host:9080/teamworks/webservices/SCP/StartServiceClaimEventWS.tws?WSDL`

   Your dialog will be similar to that shown in Figure 4-18.

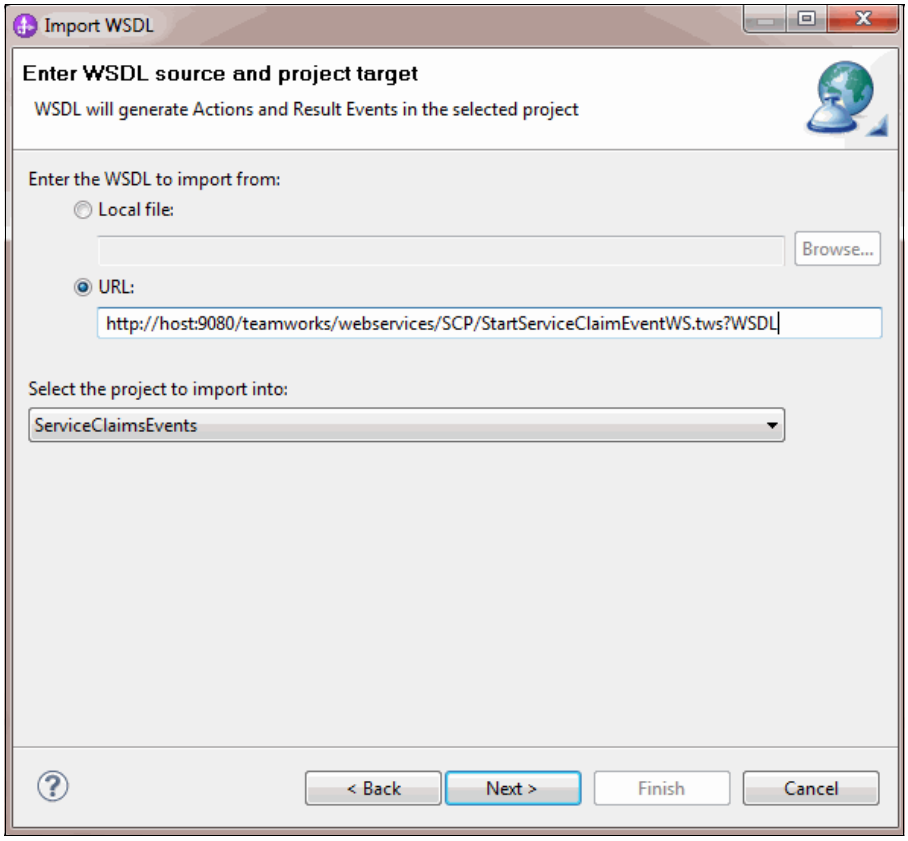


Figure 4-18 Importing a WSDL into the Event Designer

4. Click **Next**, followed by **Finish**.

The Event project has now been created with some minimal artifacts that trigger the BPM process when the events match the defined pattern.

Create an Event

The event and the event object that describe the event data need to be defined:

1. Right-click **ServiceClaimsEvents** and select **New** → **Event** from the pop-up dialog.

2. Leave the next panel as **Create a blank event** and click **Next**.
3. Name the event `EventReceived` and click **Next**.
4. Select Simple Object Access Protocol (**SOAP**) as the Event Connection type and click **Finish**.
5. Click **Save** to create your event.

The Event object does not have a data model associated with it, so you need to define the Event object that will hold the event data:

1. Select **EventReceived** (new event), and on the right side of the Event editor in the Event Objects section, click **Add**.
2. Specify **Add a new blank event object** and click **Next**.
3. Name the event object `EventReceivedObject` and click **Finish**.
4. In the EventReceivedObject editor, go to the Fields section and click **Add**.
5. Specify a Field name of `SerialNumber` with type `String`.
6. Click **Finish**.
7. In the Fields section, click **Add**.
8. Specify a Field name of `eventTime` with type `DateTime`.
9. In the Fields section, click **Add**.
10. Specify a Field name of `trackingNumber` with type String.
11. Click **Finish**.
12. Click **Save All**.

The resulting Event object fields look similar to Figure 4-19.

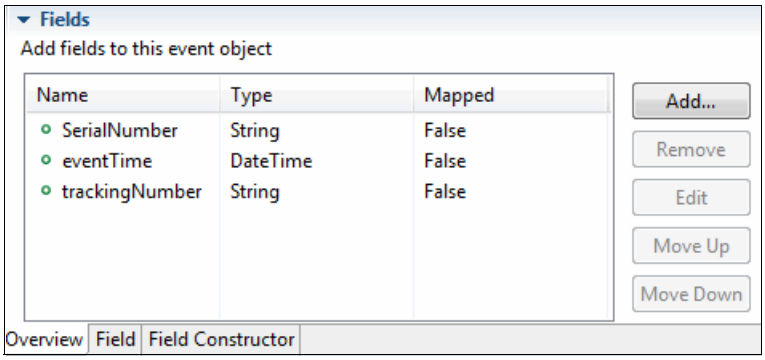


Figure 4-19 Event object fields after they have been defined in the Event Designer

Create a business object

Business objects are representations of objects such as Customer, Product, or Order. The primary purpose of a business object is to hold the data that is evaluated by event rules at run time. The data that populates a business object does exist in the enterprise applications, but it typically does not exist in a single record or structure. The fields in a business object can be populated from various sources.

A business object, as defined to Decision server Events, is the ideal representation of the business object because it serves as a template for sharing data, containing a number of fields that can be populated from many different business applications across the enterprise. Each field includes the name and type of information and an optional definition for how to get a value, if none is supplied with the event.

Two business objects are created in the service claim project: one to hold the event data for the current event, and another to accumulate historical data so that it is possible to compare events over time:

1. Right-click **ServiceClaimsEvents** and select **New** → **Business Object**.
2. Enter the name `ServiceClaimsHistoryBusinessObject` and click **Next**.
3. Select the option **Create an accumulating array that retains the values of fields from previous events** and click **Next**.
4. Enter a maximum number of instances of 2 and click **Next**.
5. Select **EventReceivedObject** and click **Finish**.
6. Click **Save**.

Now create another business object that holds the data that will be passed to the outgoing web service that launches the process in response to the situation being detected:

1. Right-click **ServiceClaimsEvents** and issue **New** → **Business Object**.
2. Enter the name `ServiceClaimsBusinessObject` and click **Next**.
3. Leave the option **Start with a blank business object** selected and click **Finish**.
4. In the Overview tab of the newly created business object, in the Fields section, click **Add**.
5. Specify a Field name of `SerialNumber` and a Data type of **String** and click **Finish**.
6. In the Fields section of the newly created business object, click **Add**.
7. Specify a Field name of `TimeDifference` and a Data type of **Integer** and click **Finish**.
8. In the Fields section of the newly created business object, click **Add**.
9. Specify a Field name of `LastEventTime` and a Data type of **DateTime**.
10. In the Fields section of the newly created business object, click **Add**.
11. Specify a Field name of `TrackingNumber` and a Data type of **String** and click **Finish**.
12. Double-click the **TimeDifference** field to edit it, choose a definition type of `Javascript`, and complete the expression box as shown in Example 4-3.

Example 4-3 TimeDifference function

```
function timediff(f) {
   var diff = f[1] - f[0];
  return Math.floor(diff/1000);
}

timediff(ServiceClaimsHistoryBusinessObject.eventTime);
```

The Field definition will look similar to Figure 4-20 on page 68.

13. Click **Save**.

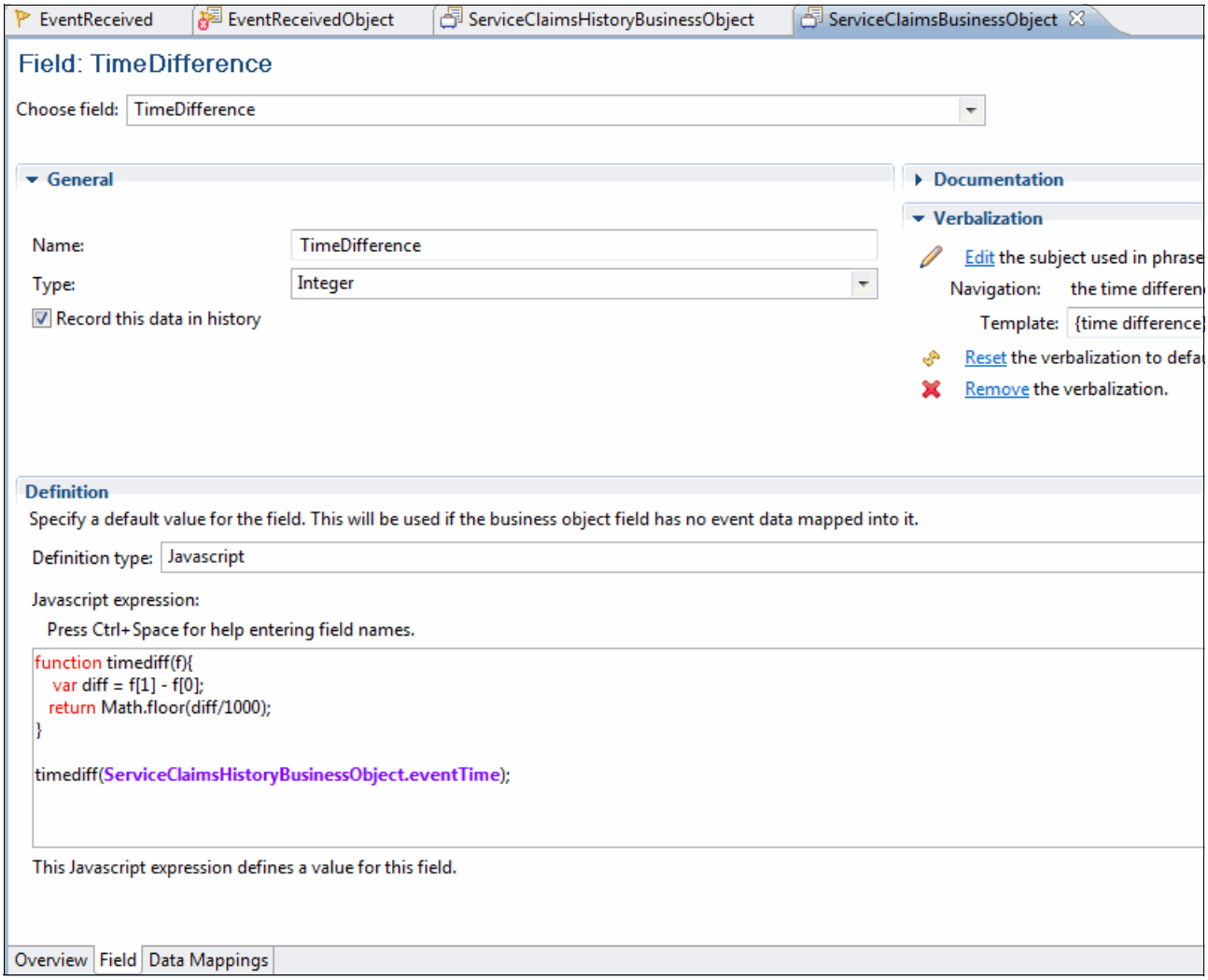


Figure 4-20 JavaScript for the TimeDifference field

14. Click the **Overview** tab to return to the list of fields.
15. Double-click the **LastEventTime** field to edit it, choose a definition type of Java script, and complete the expression box as shown in Example 4-4.

Example 4-4 LastEventTime function

```
function lastTime(f) {
  return f[1];
}

lastTime(ServiceClaimsHistoryBusinessObject.eventTime);
```

The field definition will look similar to Figure 4-21 on page 69.

16. Click **Save**.

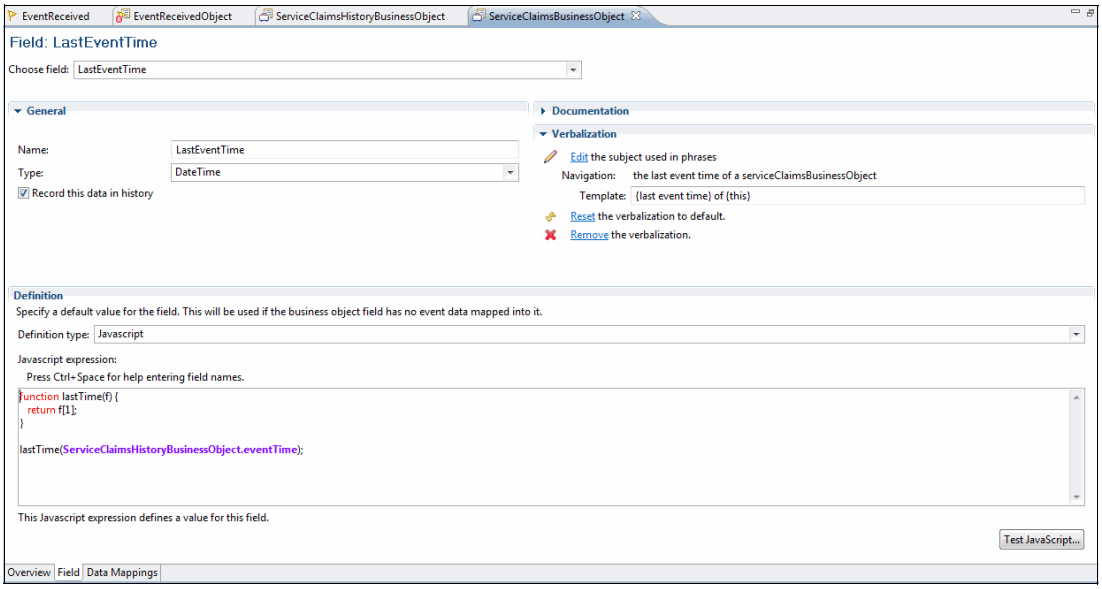


Figure 4-21 JavaScript for the LastEventTime field

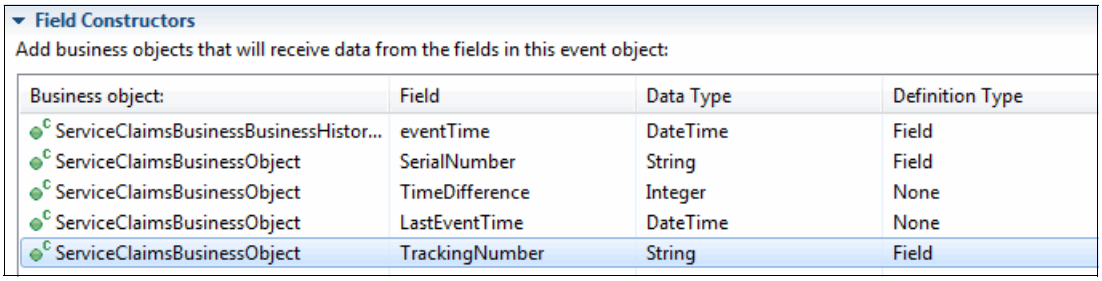
The creation of the Business objects is now complete.

Map the Event object to the Business object

When your Event object was created, it contained errors because there was no Business object to populate. In the previous section, you created the Business objects, so now the fields in the Event object to the Business objects can be mapped:

1. Double-click **EventReceivedObject**.
2. In the Fields Constructors section in the Overview tab, select **SerialNumber** and click **Remove**.
3. In the Field Constructors section in the Overview pane, click **Add**.
4. Select the **ServiceClaimsBusinessObject** and click **Finish**.
5. In the Field section, double-click the **SerialNumber**.
6. Select a Definition Type of **Field** and select the Event object field **SerialNumber**.
7. Click **Save**.

The mapping of the Event object to the Business objects and your mapping will look similar to Figure 4-22.



Field Constructors

Add business objects that will receive data from the fields in this event object:

Business object:	Field	Data Type	Definition Type
ServiceClaimsBusinessBusinessHistor...	eventTime	DateTime	Field
ServiceClaimsBusinessObject	SerialNumber	String	Field
ServiceClaimsBusinessObject	TimeDifference	Integer	None
ServiceClaimsBusinessObject	LastEventTime	DateTime	None
ServiceClaimsBusinessObject	TrackingNumber	String	Field

Figure 4-22 Field constructors for the completed Business object

Map the Business object to the Action object

Just as it was necessary to map the input from an Event object to the Business object, it is necessary to map the output from a Business object to an Action object:

1. Double-click the Action Object **StartServiceClaimsEventInBPMRequest**.
2. Change Definition Type to **Field** for the fields:

- **impl:SerialNumber**
- **impl:TimeBetweenEvents**
- **impl:TimeOfLastEvent**
- **impl:TrackingNumber**

3. Select the Business Object **ServiceClaimsBusinessObject** and the Business Object field as in Figure 4-23.
4. Double-click the field **impl:isDuplicate**.
5. In the Definition section, select **Parameter** as the Definition Type.
6. Type the **Parameter Name** as `isDuplicated`.
7. Click **Save**.

The mapping of the Business object to the Event object is complete, and your mapping will look similar to Figure 4-23.

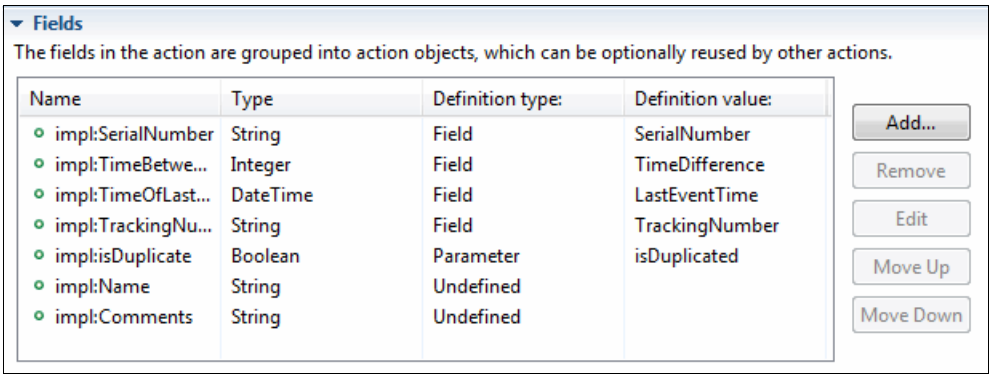


Figure 4-23 Completed Action object mappings

Create an Event rule

To make this a functioning event project, indicating a response to the situation being detected, is an event rule. Event rules are the conditions under which, based on a pattern of events, an action fires. To create an event rule:

1. Right-click **ServiceClaimsEvents** and select **New** → **Event Rule**.
2. Name the rule `MultipleServiceClaimEvents` and click **Next**.
3. Select the event **eventReceived** and click **Next**.
4. Select **System Context** and click **Next**.

   Specify the context relationship as **the serial number of the serviceClaimsBusinessObject** and click **Finish**.
5. Before typing the rule create a new action template. Double-click the action **StartServiceClaimEventInBPM**.
6. Once the action is displayed, check the verbalization section on the right side of the window and update the action template to **startServiceClaimEventInBPM with**

parameter {0,isDuplicated}. This provides the parameter to be sent when starting the BPM process.

7. Create the rule by typing in the content from Figure 4-24.

> **Tip:** As you type the rule, you are presented with suggestions based on the object model. You can activate this list of options any time in the rule by pressing **Ctrl+Space**.

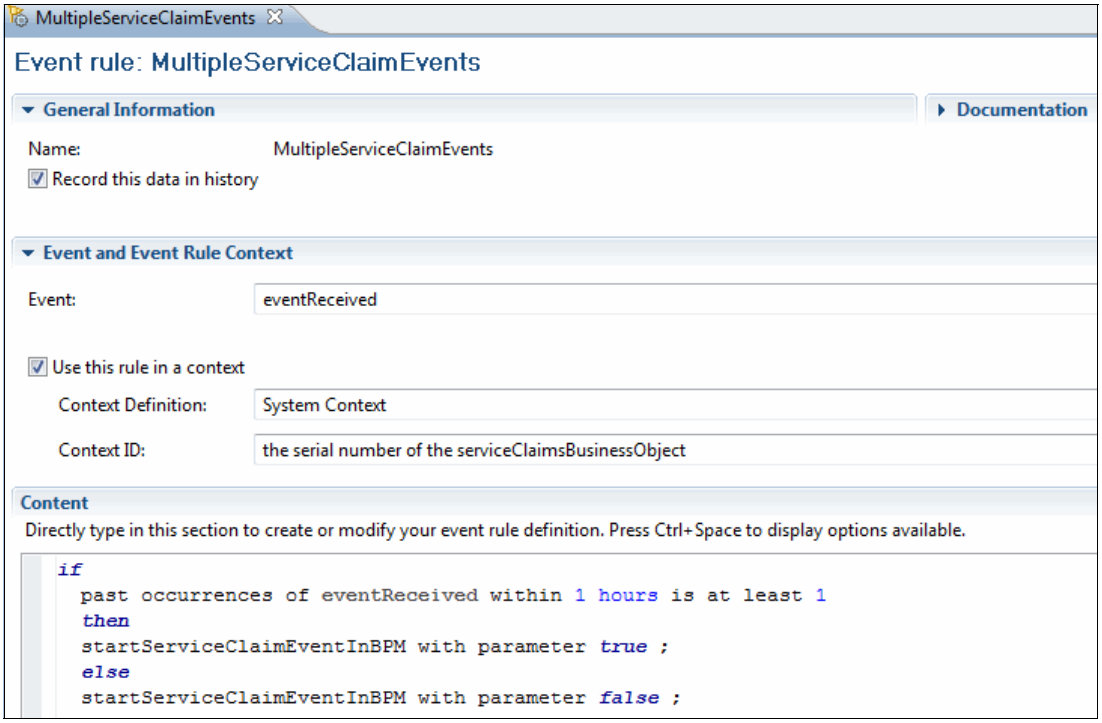


Figure 4-24 Event rule to detect multiple claims within a short time

8. Click **Save**.

The Event project is complete and is ready for testing. The next step is to deploy the Event project.

Event deployment

Deploy the event rule as follows:

1. Start the WebSphere Decision Server by clicking **Start the server** of the **ODMSample8010** profile that is configured on WebSphere Application Server v8.0.

> **Note:** The first time this is done, it takes a few minutes for the server profile to be created.

2. Switch to the Event perspective on the upper right corner.
3. Right-click **ServiceClaimsEvents** and click **Deploy**.
4. Select **Deploy all assets** and click **Next**.
5. Enter the credentials of your WebSphere Decision Server configuration. The credentials will display similar to those in Figure 4-25 on page 72.

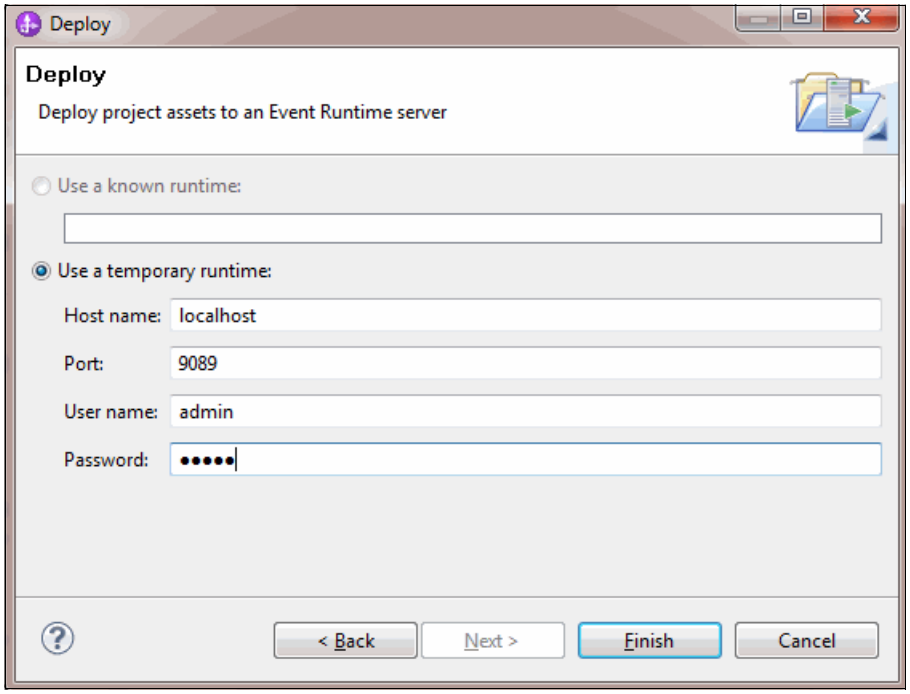


Figure 4-25 Configuration dialog for the Event deployment server

6. Click **Finish** and wait for the deployment to terminate.
7. After receiving a message that the deployment is successful, click **OK**.
8. Check WSDL to ensure that the event project deployed correctly.
9. Open a web browser and go to the following address:

   `http://localhost:9080/wbecasoap/SOAPEventWSDL`

   The browser displays a WSDL similar to the one in Figure 4-26 on page 73.

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <wsdl:definitions targetNamespace="http://www.ibm.com/wbe/casoap" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl
    xmlns:tns="http://www.ibm.com/wbe/casoap" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="htt
  - <wsdl:types>
    - <xsd:schema targetNamespace="http://www.ibm.com/wbe/casoap/types">
      - <xsd:element name="EventReceived1">
        - <xsd:complexType>
          - <xsd:sequence>
            - <xsd:element name="EventReceivedObject1" minOccurs="0" maxOccurs="unbounded">
              - <xsd:complexType>
                - <xsd:sequence>
                    <xsd:element name="SerialNumber" type="xsd:string" />
                    <xsd:element name="eventTime" type="xsd:dateTime" />
                    <xsd:element name="TrackingNumber" type="xsd:string" />
                  </xsd:sequence>
                </xsd:complexType>
              </xsd:element>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:schema>
    </wsdl:types>
  - <wsdl:message name="EventReceived1Request">
      <wsdl:part name="parameters" element="types:EventReceived1" />
    </wsdl:message>
  - <wsdl:portType name="SOAPEventPortType">
    - <wsdl:operation name="EventReceived1">
        <wsdl:input message="tns:EventReceived1Request" />
      </wsdl:operation>
    </wsdl:portType>
  - <wsdl:binding name="SOAPEventBinding" type="tns:SOAPEventPortType">
      <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http" />
    - <wsdl:operation name="EventReceived1">
        <soap:operation soapAction="http://www.ibm.com/wbe/casoap/EventReceived1" />
      - <wsdl:input>
          <soap:body use="literal" />
        </wsdl:input>
      </wsdl:operation>
    </wsdl:binding>
  - <wsdl:service name="SOAPEventService">
    - <wsdl:port name="SOAPEventPort" binding="tns:SOAPEventBinding">
        <soap:address location="http://fit-vm5-112.rtp.raleigh.ibm.com:9080/wbecasoap/SOAPEventConnector" />
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>
```

Figure 4-26 Event WSDL showing that the event run time can receive external events

There might be other environments, such as test or production, to which you want to deploy the same rule. In that case, the event run time for each environment needs to be defined. During the deployment of the rule, you can specify which event run time the rule will be deployed to.

Note: For guidance about defining event runtimes and deploying to event runtimes, see *Defining an event runtime server connection* at:

http://pic.dhe.ibm.com/infocenter/dmanager/v8r0m1/topic/com.ibm.wodm.dserver.events.dev/topics/tsk_dse_definingruntime_serverconn.html

Event testing

WebSphere Operational Decision Manager has a simple, web-based testing interface to check that the event project is working. The testing interface is available using IBM Business Space. Other tools, such as SoapUI, can be used to test the event as well.

Set up Business Space

To set up Business Space, follow these steps:

1. Access Business Space by opening a browser and going to:

   http://localhost:9080/BusinessSpace

2. Enter the administrator user name and password and click **Login**.
3. Select **Manage Spaces** → **Create Space**.
4. Name the new space **Event Testing** and click **Save**.
5. Create a page for the space. Provide a name for the space, and leave the **Create an empty page** radio button selected. Click **OK**.
6. Click the **Event Testing** space you just created.
7. Click the **Edit page** link in the right corner.
8. Select **Events** in the drop-down list.
9. Drag the **Event Tester** widget onto the new blank page, as shown in Figure 4-27.

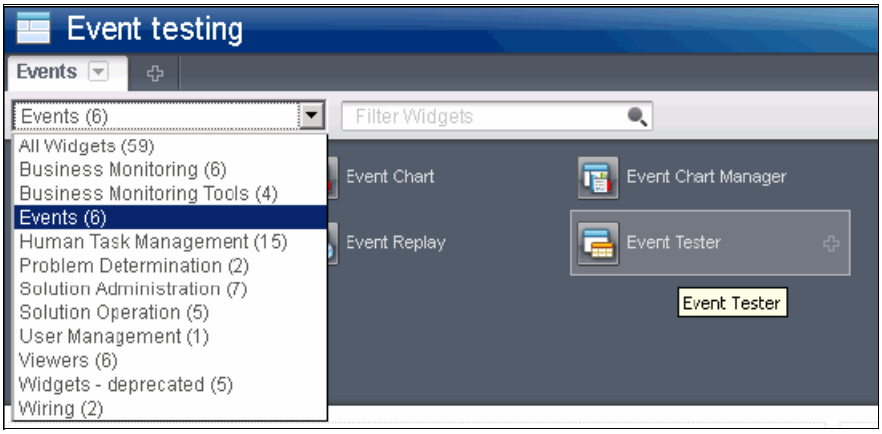


Figure 4-27 Event testing page

10. Click **Save** and click **Finish Editing**.

The Event testing widget enables you to send event data to the event run time. Select the event template for EventReceived and try entering different events. Each event triggers the action with a parameter. When you enter two events that are within an hour of each other, and for a service claim that contains the same serial number, action begins by assigning the parameter with a different value according to the rule.

This action is visible in the Business Events testing interface. Additionally, as the event action has been wired to the business process, you can open the Process Designer by selecting **Start** → **All Programs** → **IBM** → **Process Designer** → **Process Designer v8.0**, logging in with a valid user name and password, and going to the inspector view to confirm the new processes that have been launched.

Note: The Event Rule project is created, and it is assumed that you share the created project in Rational Team Concert. For details, see Appendix A, “Integrating the Eclipse environment with IBM Rational Collaborative Lifecycle Management” on page 223.

4.1.6 Business management of rules

The core business benefit of business rules and event rules is that they can be managed by business users. This requires a large cultural change for many organizations, and many in IT might resist giving the ability to make system changes directly to business users. To address these concerns, it is important to discuss how WebSphere Operational Decision Manager provides the ability for business users to manage rules in a controlled manner.

In this section, we describe the principles of the rule governance model. This model can be customized to provide the level of control wanted by an organization. This section includes the procedure for deploying a rule application so that it can be managed effectively.

Rule governance and promotion model

We define rule governance as the procedures and mechanisms that are put in place around rule maintenance. These procedures and mechanisms ensure that the rules are performed in a controllable manner. The core component of WebSphere Operational Decision Manager, which enforces rule governance, is the Decision Center by way of the rule repository. Every rule in this repository has a status (such as *New*, *Defined*, *Tested*, or *Rejected*), and each is defined by the organization building the rules. In addition, the organization defines roles that are authorized to interact with the rule management system, given the level of access control granted to each user. Using a combination of roles and statuses, it is possible to construct a state diagram that describes the rule governance model at a high level. An example model is shown in Figure 4-28 on page 76.

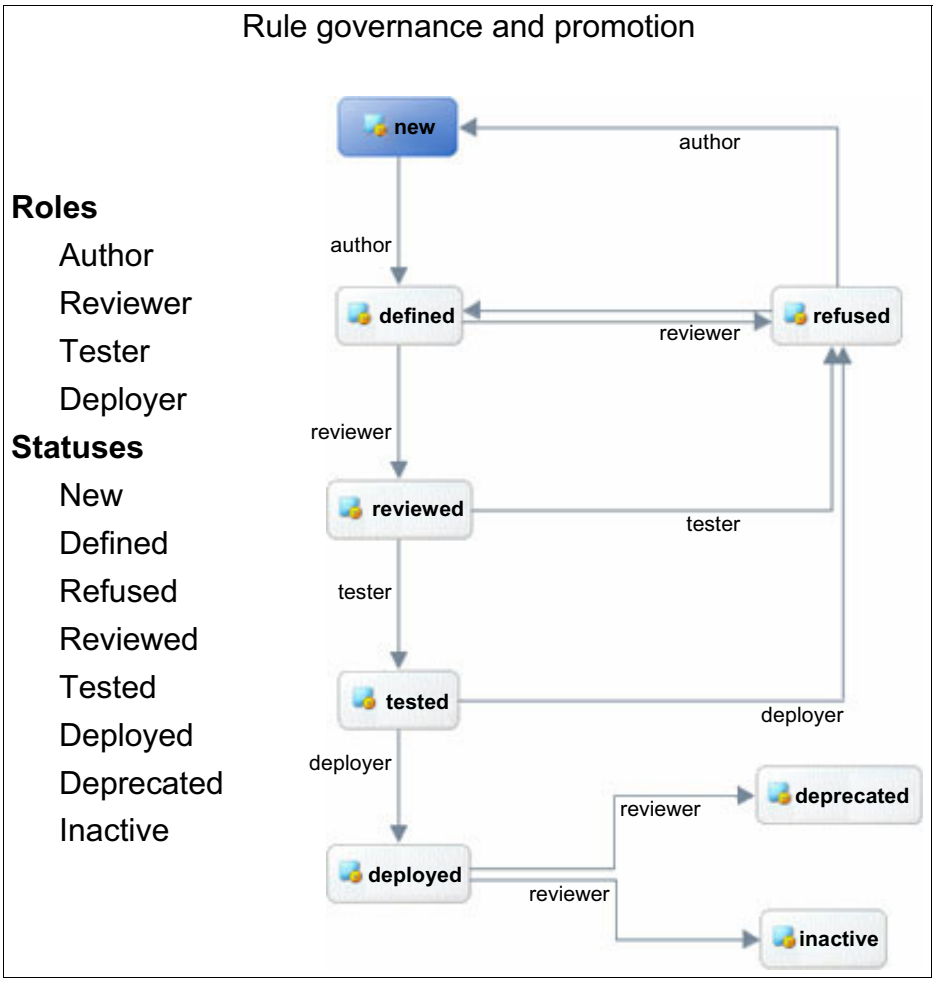


Figure 4-28 Example of a rule governance and promotion model

In Figure 4-28, you can see that each role has a defined set of transitions that it can perform from one status to another. In addition, it is possible to restrict each role in terms of capabilities. For example, certain authors might only have the ability to change certain groups of rules. It is also possible to define templates that restrict which parts of a rule can be modified.

Using Decision Center

Decision Center has a repository where rules are stored for business user interaction. It is the role of the business rule developer and event rule developer to synchronize, in the Decision Center, the rules that each developer designed to make them available to business users. After the rule or event projects have been deployed, a business user can log in to the Decision Center using a web interface to manage their rules.

There are two web interfaces:

- The Decision Center Enterprise console

  As noted in “The WebSphere Decision Center” on page 54, the Decision Center Enterprise console is a more sophisticated environment than the WebSphere Decision console. The Decision Center Enterprise console provides business users with a significant level of control over rules. Although the Decision Center Enterprise console is not as powerful as the Rule Designer (in that the console hides the low-level

implementation details), the Decision Center Enterprise console does support the full range of rule management activities.

- The Decision Center Business console

  The Decision Center Business console is a simpler environment for business users who are primarily focused on maintaining rules and need to interact with others to track and discuss changes with other users.

Deployment to Decision Center

To examine the tools that are available to business users for rules management, export the Rule project and Event project to Decision Center.

> **Note:** The procedure that follows is identical for both the Rule Designer and the Event Designer. If you want to access each of these in the Decision Center, carry out this procedure in each IDE separately. If you are already connected to Decision Center, we suggest that you disconnect. These instructions do not cover how to synchronize or delete an existing deployment.

To carry out this export to the Decision Center, follow these steps:

1. Open or switch to the IDE from which you will perform the deployment (either the Event Designer or the Rule Designer).
2. Right-click the project (either **ServiceClaimsRules** or **ServiceClaimsEvents**) to be deployed and select **Decision Center** → **Connect**.

   > **Note:** If the deployment is being done from within Rule Designer, the project that is deployable to Decision Center is the Rule Project, not the RuleApp or the Java Project.

3. Enter the connection details for the Decision Center. The address is the same as that when opening the Decision Center from the Start Menu. The user name is typically set to the default value of either `admin` or `rtsAdmin` and has a password that is the same as the user name. Your parameters will be similar to those shown in Figure 4-29.

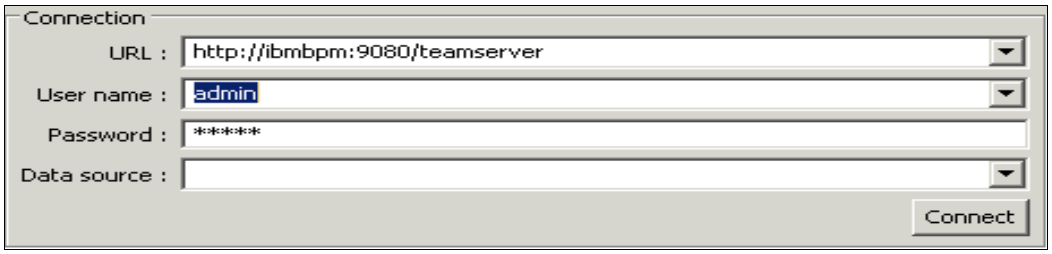


Figure 4-29 Connection parameters for the Decision Center

> **Note:** The machine name and port number for your installation might be different. If you cannot connect, confirm that your server is running, and then try changing the machine name to `localhost` (assuming the server is running locally) and check the port that it is running on. If port 9080 is already allocated when the server was installed, Port 9081 can be used.

4. Click **Connect**.
5. After the connection is established, click **Finish**.

> **Note:** The next two steps might need to be done in reverse order, depending on how long you wait for the deployment to run. Read these steps before performing them.

6. Click **No** to avoid switching to the Synchronizing Perspective.
7. Click **OK** to accept the Synchronize Complete dialog.

Your project is now deployed in Decision Center.

For security reasons, the deployment described in “Deployment to Decision Center” on page 77 might not be feasible for a production environment. Refer to the link in the following note for further information.

> **Note:** For guidance about how to deploy to offline server, see *Deploying business rules* at:
>
> http://pic.dhe.ibm.com/infocenter/dmanager/v8r0m1/topic/com.ibm.wodm.dserver.rules.deploying/topics/wodm_dserver_rules_deploying.html

Business user interface

It is now possible to log in to either the Decision Center Enterprise console or the Decision Center Business console to view or modify the existing rules. The sections that follow show how to access each interface and perform simple navigation.

Decision Center Enterprise console

To open the Decision Center Enterprise console:

1. From the browser, enter the following:

   http://host:9080/teamserver/faces/login.jsp

2. Enter the user name and password and click **Login**.

   > **Note:** The user name is typically set to a default value of `admin` or `rtsAdmin` and has a password that is the same as the user name.

   The home page of the Decision Center Enterprise console displays.

3. Select a project to access, as shown in Figure 4-30 on page 79.

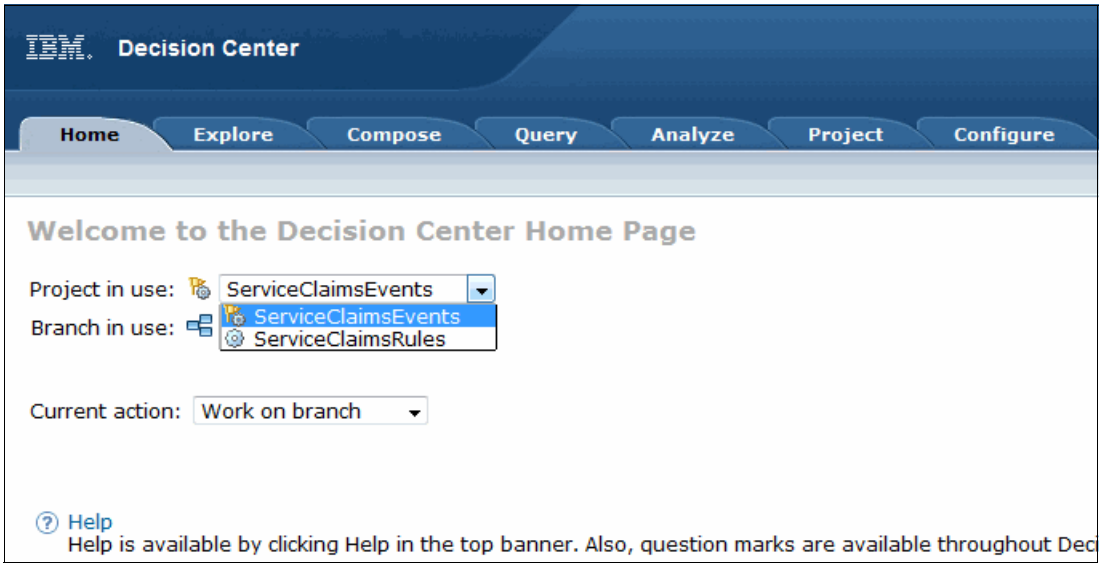


Figure 4-30 Decision Center Enterprise console home page

You can now explore the project from a business user perspective. The Explore tab contains the project structure, and from this tab you can select rules and modify them. For example, to view the event rule that you created, follow these steps:

1. From the home page, set the **Project in use** to **ServiceClaimsEvents**.
2. Click the **Explore** tab.
3. Click the **Rules** folder.
4. Click **MultipleServiceClaimEvents**.

The rule displays, as shown in Figure 4-31.

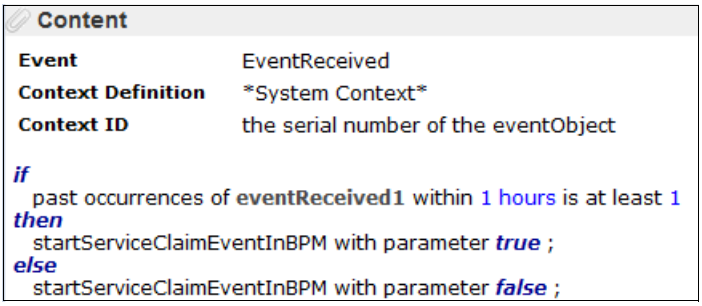


Figure 4-31 A rule inside Decision Center Enterprise Console

You can explore the interface and edit the rule here. However, see the following Note.

Note: You should not edit the rule from Decision Center Enterprise Console. All rule editing should go through the requirements, change, and quality management processes for proper governance. For the rules to be synchronized, this task must be performed from Event Designer, which is beyond the scope of this section.

Decision Center Business console

To open the Decision Center Business console:

1. From the browser enter the following:

   `http://host:9080/decisioncenter/login`

2. Enter the user name and password and click **Login**.

The What's New tab on the home page of the Decision Center Business console is now visible.

> **Note:** The user name is typically set to a default value of `admin` or `rtsAdmin` and has a password that is the same as the user name.

To view a rule, perform the following steps:

1. Click the **Library** tab.
2. Click the **ServiceClaimsRules** project.
3. Click the **partCostStatus** rule.

The rule you created displays as shown in Figure 4-32.

Figure 4-32 Rule in Decision Center Business console

This panel displays the rule in the left pane, with rule properties displayed in the right pane. Click the **Timeline** tab in the left pane to view the history and past versions of this rule. There have been no updates to the rule being viewed in Figure 4-32, however.

Click the **Stream** tab in the right pane to view all comments and other activities associated with this rule. You can add comments here as well. Other business users working on the same rule can use this feature to discuss and collaborate about this rule.

4.2 Managing business processes with Business Process Manager

BPM solutions enable an enterprise to choreograph processes and the process steps across disparate applications, people, and systems. Our service claim sample uses Business Process Manager V8.0.1 to realize the advanced, integration-related components of this comprehensive BPM platform. Service Claim Business Process is modeled with BPM.

This section of the chapter describes the following major components of the service claim sample:

- Service claim business process: Manages the core business process flow of the solution, and acts as the choreographer of process steps and activities.
- Integration with events and decision management: Manages the integration between process components and events, database, and decision management components.

Important: A basic knowledge of Business Process Manager is needed to fully understand this chapter.

4.2.1 Overview of BPM

The concept of process optimization was introduced during the Industrial Revolution through specialization. Thought leaders worked on streamlining their processes for producing goods, whether physical or natural, to get more products to market at a lower cost. This concept was adopted by many business leaders over the past decade because they needed help in managing the interactions between systems and humans. This discipline is BPM.

The key distinction between BPM and process optimization is the added focus of BPM on flexible and dynamic process design, process orchestration, and automation by the use of IT enablement. In addition to reduced cost through continued process improvement and automation, BPM provides the foundation for converged and agile businesses and IT responsiveness. Business leaders today look to BPM for guidance in getting products and services to market, better, faster, and cheaper than their competitors.

Figure 4-33 illustrates the core concepts of this discipline.

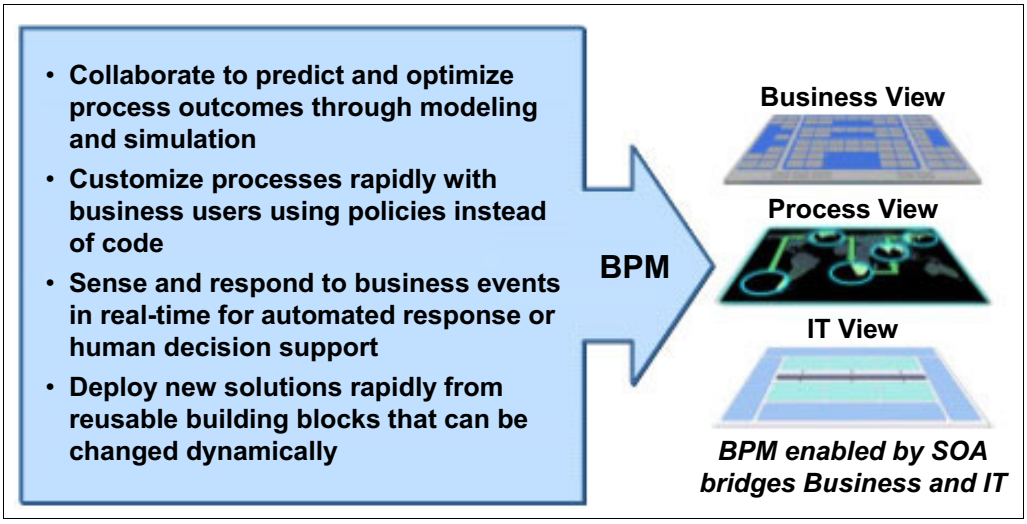


Figure 4-33 BPM drives business and IT alignment and responsiveness

BPM focuses on driving overall bottom-line success by integrating business verticals and optimizing core work (for example order-to-cash, integrated product development, and integrated supply chain). This focus helps to direct the deployment of resources throughout the organization into efficient processes that create customer value. This differentiates BPM from traditional (that is, compartmentalized) functional management disciplines.

Managed business process

A *managed business process* is a process in which stakeholders and process owners have both visibility into the process and the ability to modify it to produce better business outcomes. With process control, you can make informed decisions about how best to change the behavior of your process. These decisions might affect the process itself, or they might impact domains outside of the business.

As a manager, you regularly participate in decisions for change (see Figure 4-34). All management decisions, at any level in the organization, can be associated with one or more of the following domains: corporate strategy, business resources, and business processes.

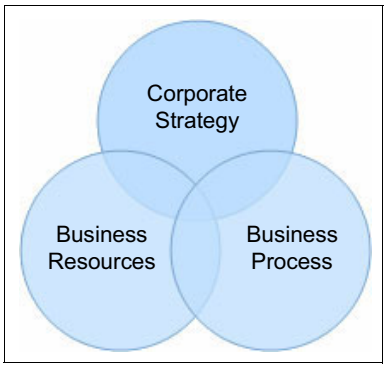


Figure 4-34 Domains of change for a managed business process

Examples of corporate strategy decisions include entering new markets, discontinuing a product, and selling assets. You can imagine how these decisions might impact or lead to decisions in the *business resource domain*, such as hiring or training new human resources, outsourcing jobs, investing in technology, or making capital improvements to facilities. Decisions in the *business process domain* might include changing a decision threshold in a process flow to reduce the human workload (that is, lower the threshold) or reducing risk (raising the threshold) by changing the behavior of the process. Business process decisions might also include feedback to change the process by adding or removing activities.

4.2.2 Process automation, visibility, and control

Process automation, visibility, and control are compounding elements of the business impact realized by BPM (see Figure 4-35).

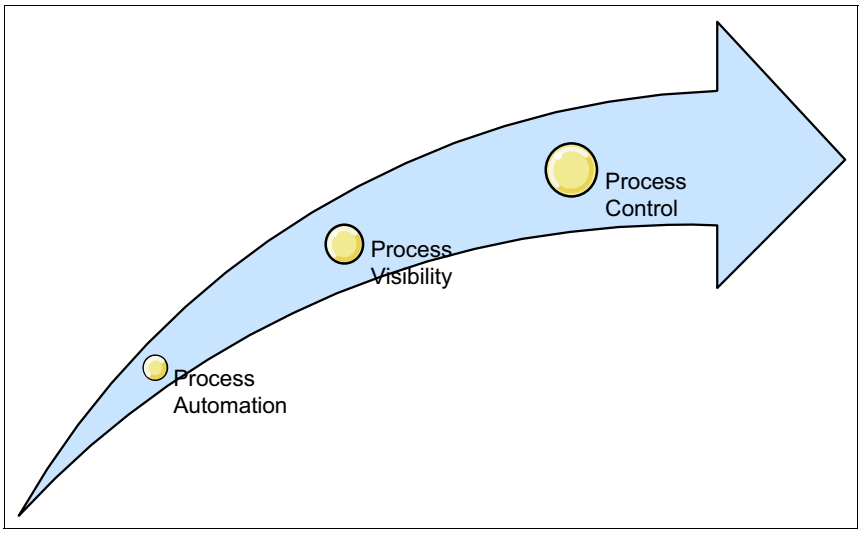


Figure 4-35 Elements of compounding values of a managed process

Process automation immediately accrues business value by increasing efficiency, reducing errors, eliminating process variation, and removing rework for human tasks. It is important to recognize, however, that automation is not the end goal of process improvement using BPM.

Process visibility allows you to see new aspects of your processes in tangible ways, and in some cases, in real time (when invoked). These capabilities provide insight into the performance of your processes, which can help identify areas for improvement through timely action. In addition, you will see a full, end-to-end view through process visibility.

Process control is what ultimately differentiates a managed process from an unmanaged one. Having control over your design-time and runtime business processes means you can engage the correct skills to address problems or effect change in a timely manner. By default, control demands full governance of your processes to ensure that all are operating consistently and in compliance with both internal and external policies and regulations.

4.2.3 Overview of Business Process Manager

Business Process Manager is a comprehensive and consumable BPM platform that provides visibility and management of your business processes. It allows IT to enable business users and managers to track their entire business operation on a single dashboard, receive alerts, and subsequently drill down to the lowest level of instance detail.

Figure 4-36 on page 84 depicts the major components of Business Process Manager.

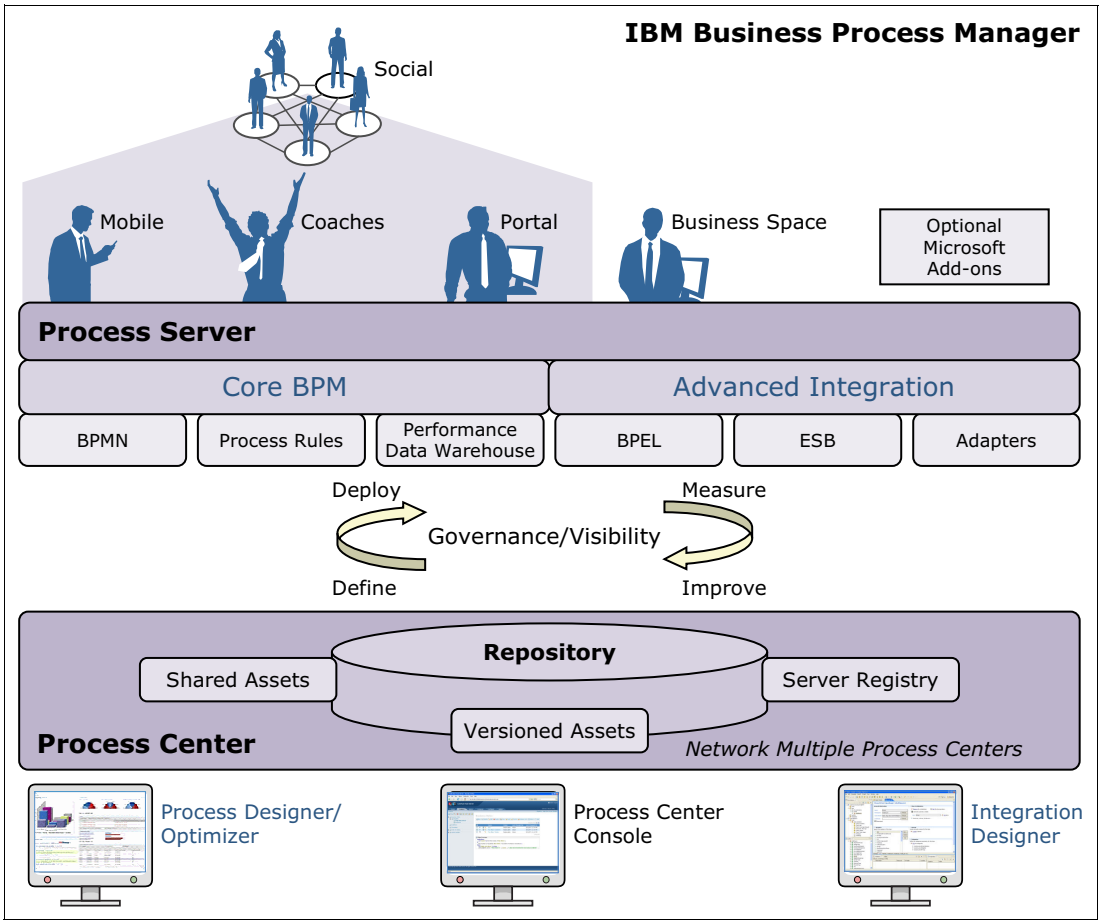


Figure 4-36 Major components of Business Process Manager

Next, we describe the available configurations in the Business Process Manager.

Business Process Manager configurations

Business Process Manager V8.0.1 is available in the configurations listed in Table 4-2.

Table 4-2 Business Process Manager configurations

Configuration	Phase
Advanced	**Transformation** Complete set of BPM capabilities: ► Extended support for high-volume process automation ► Built-in service-oriented architecture (SOA) components for extensive enterprise-wide service integration and orchestration
Standard	**Program** Configured for typical BPM projects: ► For multiproject improvement programs, with high business involvement ► Basic system integration support ► Rapid time-to-value and improved user productivity
Express	**Project** Configured for a first BPM project: ► Rapid time-to-value and improved user productivity ► Low entry price ► Easy installation and configuration

Business Process Manager is a single BPM platform that combines human-centric and integration-centric capabilities into a unified product, which can be consumed per the capability requirements shown in Table 4-3.

Table 4-3 Business Process Manager configuration capabilities

Capability	Advanced	Standard	Express
WebSphere Lombardi Edition compatible execution	X	X	X
Process Designer (Business process model and notation, or BPMN)	X	X	X
Collaborative editing and immediate playback	X	X	X
Interactive *process coach* user interfaces	X	X	X
ILOG based process rules	X	X	X
IBM WebSphere Process Portal	X	X	X
Real-time monitoring and reporting	X	X	X
Performance analytics and optimizer	X	X	X
Business Performance Data Warehouse (BPM feature)	X	X	X
Process Center (BPM feature) and shared asset repository	X	X	X
Unlimited process authors and users	X	X	200 users and 3 authors
High availability: clustering and unlimited cores	X	X	► 4 cores production ► 2 cores development ► No cluster
WebSphere Process Server compatible execution	X		
IBM Integration Designer (business process execution language and SOA	X		
Built-in enterprise service bus	X		
Transaction support	X		
Integration adapters	X		
Flexible Business Space user interface	X		
Advanced platform support (Linux on IBM System z®, IBM AIX®, Solaris)	X	X	

Process Center

The Process Center is a feature of BPM. It is a runtime environment that includes a repository for all process models, services, and other assets that were created using the Process Designer or IBM Integration Designer, the IBM Business Process Manager authoring environments. It has a console with the tools that you need to maintain this asset repository.

From the Process Center console, administrators can:

- Install process applications that are ready for testing or production on the process servers in those environments.
- Manage running instances of process applications in configured environments.
- Grant appropriate authorization for users and groups to access the repository.

Administrators who do not actively work in the Designer view can use the Process Center console to provide a framework in which BPM analysts and developers can build their processes and underlying implementations.

Process Server

The Process Server is a BPM feature. It provides a single BPM runtime environment that can support a range of business processes, service orchestration, and integration capabilities. Because the Process Server is integrated with the Process Center, you can run your processes as you build them directly in the authoring environments.

When you are ready, you can install and run those same processes on the process servers in your runtime environments to test and prepare each for full deployment. The Business Performance Data Warehouse component collects and aggregates the process data from your processes that are running on the process servers. This enables you to test and refine your processes before formal deployment. You can use this data to improve your business processes.

Authoring environments

Business Process Manager Advanced offers two authoring environments:

- *IBM Process Designer:* Used to model business processes that involve human tasks.
- *IBM Integration Designer:* Used to build services that are self-contained or that invoke other existing services, such as web services, enterprise resource applications, or applications running in CICS and IBM IMS™.

Normally, a business analyst creates a new process in the Process Designer and defines an advanced integration service that the process uses. An integration developer creates that service in the Integration Designer and publishes it so the business analyst can use it to complete the process. The advantage of using these authoring environments is that both the business analyst and the integration developer can collaborate effectively using the Process Center repository, which reduces the time to build, test, and deploy new or revised business processes.

Process Designer

The Process Designer is available in all editions of Business Process Manager, and it has easy-to-use, graphics tools for creating process models, reports, and simple services. In addition, you can call a service that was created in the Integration Designer using an interface to access back-end systems or obtain customer data. In short, the Process Designer focuses on the business process, and the Integration Designer focuses on automated services to complement the business process.

Process applications developed in the Process Designer can, at any time, be run on the Process Center server or saved to a snapshot and deployed on the Process server. The same is true of services that are developed in the Integration Designer and associated with process applications.

Integration Designer

The Integration Designer is available in Business Process Manager Advanced Edition or as a stand-alone toolset. It is designed as a complete integration development environment for those building integrated applications. Integration developers use it to call applications on enterprise information systems that involve business processes across departments or enterprises. These developers can also invoke applications that are locally or remotely written, in various languages, and running various operating systems.

The Integration Designer tools are based on a service-oriented architecture. All services created with this tooling comply to leading, industry-wide standards. By using visual editors that abstract service components from their implementations, integration developers can assemble integrated applications without having a detailed knowledge of the underlying implementation of the components.

In the Integration Designer, components are assembled in modules. Imports and exports are used to share data between modules. Artifacts placed in a library can be shared among modules. Modules and libraries can be associated with a process application for use with the Process Center, and can be used as services by processes created in the Process Designer.

In such cases, processes can also be deployed with the process application. Alternatively, modules and libraries can be deployed directly to the test environment or to the Process Server. You can use mediation modules to create mediation flows, which you can deploy to the IBM WebSphere Enterprise Service Bus or to the Process Server.

Administration tools

Business Process Manager includes a set of administration tools to help you accomplish tasks, ranging from installing and managing snapshots to administering processes and working with the resources in your IT environment.

Command-line tools

The command-line tools, scripting interfaces, and programming interfaces enable you to administer your runtime environment:

- Command-line tools are simple programs that you run from an operating system command-line prompt to perform specific tasks. Using these tools, you can start and stop application servers, check server status, add or remove nodes, and so on.
- The WebSphere administrative (`wsadmin`) scripting program is a nongraphical command interpreter environment that enables you to run administrative options in a scripting language, and to submit scripting language programs for execution. It supports the same tasks as the administrative console, in addition to many of the Process Center console tasks. The `wsadmin` tool is intended for production environments and unattended operations.
- Administrative programming interfaces are a set of Java classes and methods under the Java Management Extensions (JMX) specification that provide support for administering Service Component Architecture (SCA) and business objects. Each programming interface includes a description of its purpose, an example that demonstrates how to use the interface or class, and references to the individual method descriptions.

The Process Center console

The Process Center console provides a convenient location for users to create and maintain high-level library items, such as process applications and toolkits. It helps to provide a framework in which BPM analysts and developers can build their processes and underlying implementations. In addition, the Process Center console provides tools for maintaining the repository, including setting up the appropriate authorization for users and groups.

You can access the Process Center console using a web browser:

http://host:9080/ProcessCenter

The Process Admin console

The Process Admin console is used to administer the process servers in your environment, including the users and installed snapshots for each server. In addition, it provides tools to help you manage queues and caches.

The Process Admin console includes the Process Inspector, a tool to view and manage process instances for process applications that are running on a specific process server. You can access the Process Admin console using a web browser:

http://host:9080/ProcessAdmin

The Business Performance Admin console

The Business Performance Admin console includes tools for managing the Performance Data Warehouses in your environment. You can use this tool to manage server queues and monitor server performance. You can access the Business Performance Admin console through a web browser:

http://host:9080/PerformanceAdmin

The WebSphere Application Server administrative console

The administrative console is used to administer applications, services, and other resources at a cell, node, server, or cluster scope. You can use the console with stand-alone servers and with deployment managers that manage all servers in a cell in a networked environment.

If you installed a stand-alone profile, you have a single node in its own administrative domain, known as a *cell*. Use the administrative console to manage applications, buses, servers, and resources within that administrative domain. Similarly, if you installed and configured a network deployment cell, you have a deployment manager node and one or more managed nodes in the same cell. Use the administrative console to manage applications, set up managed nodes in the cell, and monitor and control those nodes and their resources. You can access this console using a web browser:

http://host:9043/ibm/console

Business Process Choreographer Explorer and Business Process Archive Explorer

Depending on your user role, you can do the following with the Business Process Choreographer Explorer and Business Process Archive Explorer:

- Use these client interfaces to manage *business process execution language* processes and human tasks created in IBM Integration Designer.
- Work with your assigned tasks.
- View completed business process execution language processes and human tasks that are in an archive database.
- Delete processes and tasks from the archive.

Business Space

Business Space is an integrated user experience for business users across the IBM portfolio of BPM products. Business Space provides a customizable and collaborative environment for you to monitor, review, and administer common business processes, such as human task flows, modeling, and performance indicators.

Business Space not only provides a single web-based point of access for the content. You can also use Business Space to combine the content in useful and interesting ways. These combinations can provide insight into your business and the capability to react to changes.

Business process rules manager

The business process rules manager is a web-based tool that assists the business analyst in browsing and modifying business rule values. The tool is an option on IBM Process Server that you can select to install at profile creation time or after installing the server.

4.2.4 Create a service claim sample

Now that the components and merits of Business Process Manager are discussed, we use it to create a business process for the service claim sample application, which is designed to process service claim exceptions. These exceptions are defined as either multiple claims coming in for the same item (that is, identified by its serial number) over a short period of time, or single claim.

In this example, a business analyst is assigned with creating the `Service Claim Exception Process` using the Process Designer. At the same time, an integration developer is tasked with creating the integration of the events and rules with the business process through the mediation layer.

To create the process:

1. Log in to the Process Designer.
2. Create a new process application by clicking the link to **Create New Process App**, indicated by the red arrow in Figure 4-37.

   This process application acts as a container or package for business processes and related artifacts.

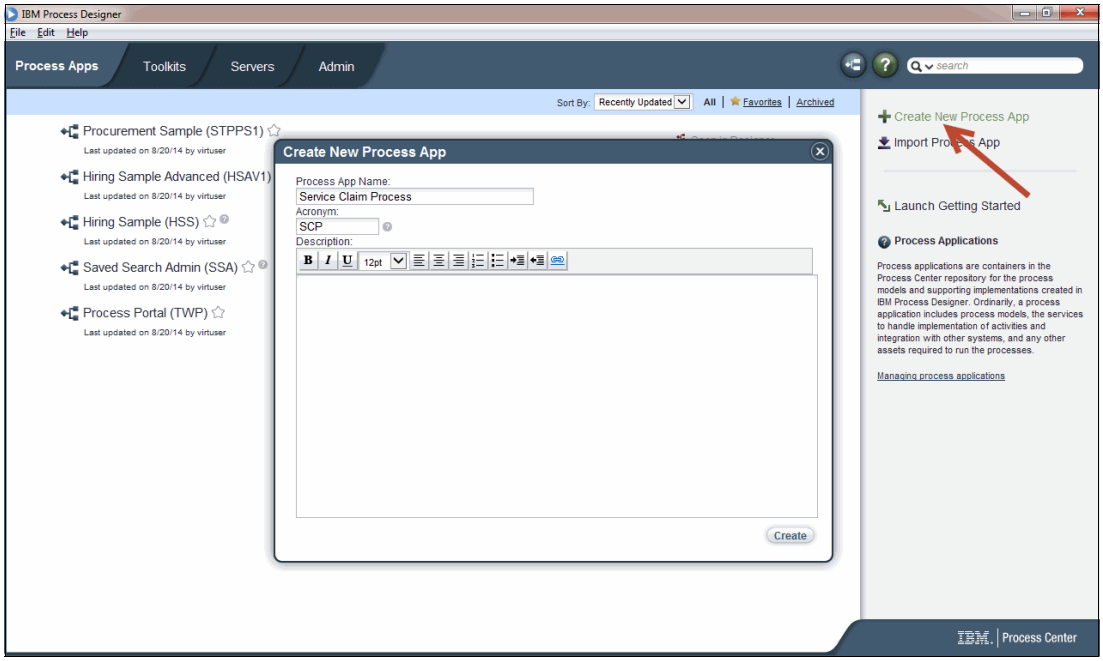


Figure 4-37 Create a new process application

3. Enter the name of the new application as `Redbooks Service Claim Processes`, provide an acronym, for example `SCP`, and enter a short description for your new process application.

4. Click the **Open in Designer** link to open the *Service Claim Process* process application in the Designer. Figure 4-38 shows this link with the red arrow.

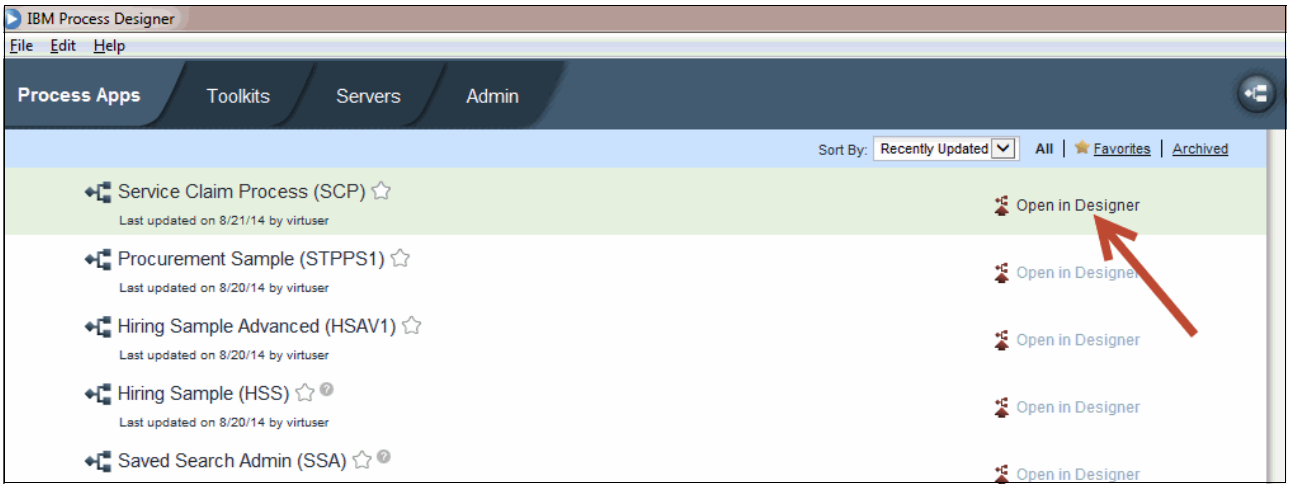


Figure 4-38 Open the newly created process application

5. Create a new business process definition by selecting **Processes** → **Business Process Definition** (click the **plus (+)** sign to the right of Processes) as shown Figure 4-39.

   To model a process in the Process Designer, create a business process definition, which is a reusable model of a process that defines what is common to all runtime instances of that process model.

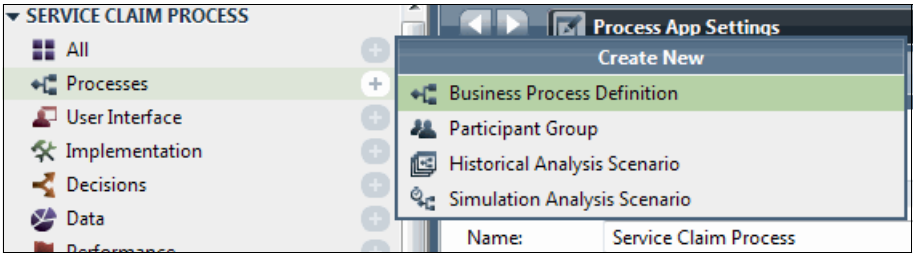


Figure 4-39 Create a new business process definition

6. Name the new business process definition as `Service Claim Exception Process`, and click **Finish** to proceed (see Figure 4-40).

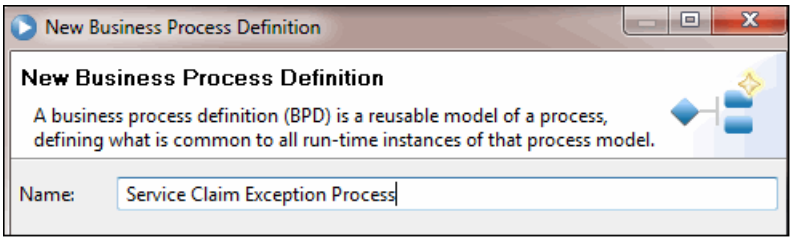


Figure 4-40 Name the new business process definition

7. The Process Designer generates your process diagram with the initial modeling constructs shown in Figure 4-41 on page 91.

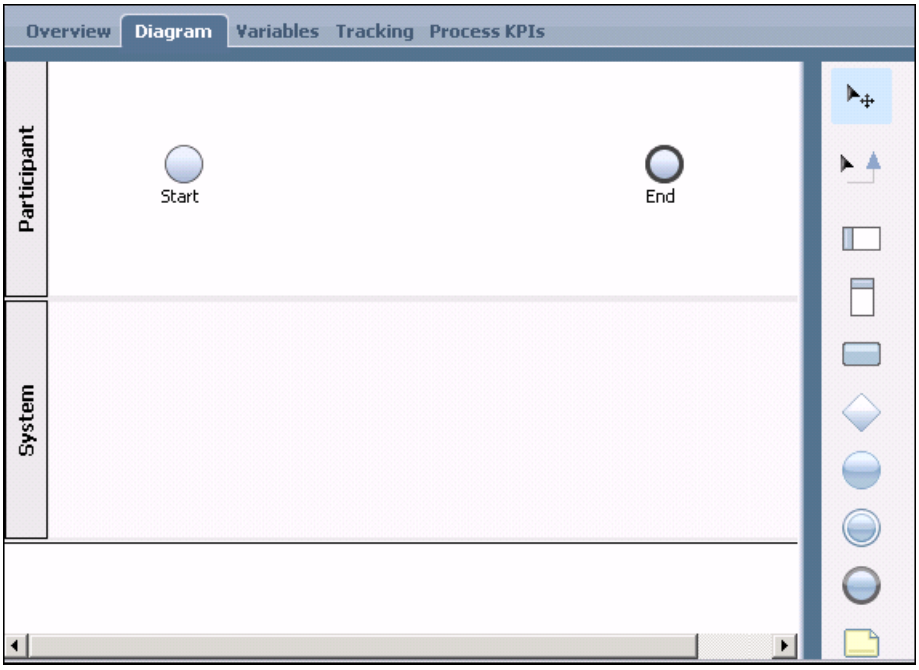


Figure 4-41 Default process diagram for the new business process

A business process definition must contain a start event, an end event, at least one swimlane, and one or more activities, which is why the default process diagram contains these initial modeling constructs.

A swimlane is a visually separated row within a process flow diagram that groups all of the activities in the process that are performed by a particular combination of roles, resources, organization units, or locations. A business process definition needs to include a swimlane for each system (a *system lane*) or group of users who participate in a process (a *participant lane*).

You can designate any specific person or group to be responsible for the activities in a participant lane. Each lane that you create is assigned to the All Users participant group by default. You can use this default participant group for running and testing your business process definition in the Inspector. The All Users participant group includes all users who are members of the `tw_allusers` security group, which is a special security group that automatically includes all users in the system.

A system lane contains activities handled by a specific Process Center system. Each activity needs an implementation, which defines the activity and sets the properties for the task. During implementation, a developer creates a service or writes the JavaScript necessary to complete the activities in a system lane.

For the `Service Claim Exception Process`, create the following lanes:

- User (user who triggers the process)
- Technical support
- Financial manager

Note: For guidance about how to add and adjust swimlanes, see *Deploying business rules* at the following site:

http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.wle.editor.doc/modeling/topic/adding_swim_lanes.html

8. When complete, your diagram looks like what is shown in Figure 4-42.

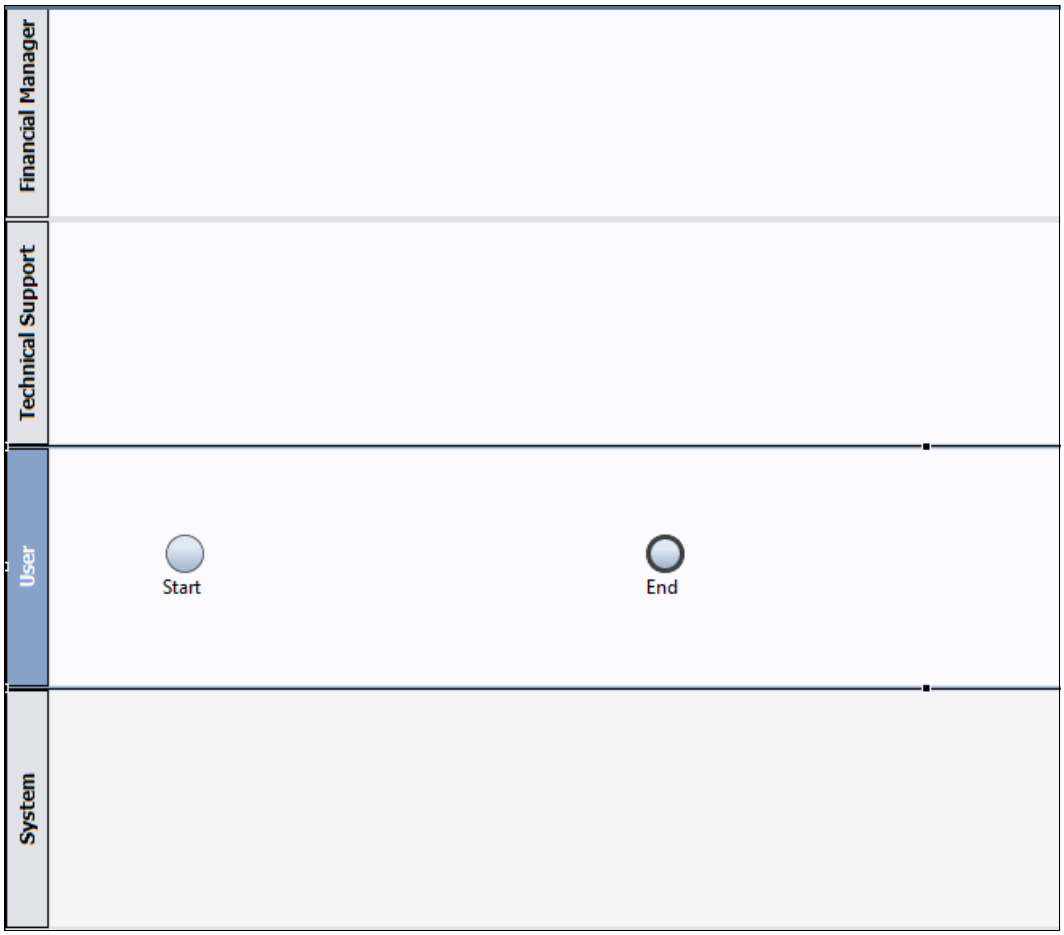


Figure 4-42 Service Claim Exception Process diagram

9. Next, model your business process, which entails the following high-level steps:
   - Dragging modeling constructs from your palette onto the diagram area
   - Specifying the details for a modeling construct by selecting it in your diagram and editing the properties in the Properties view.

   Figure 4-43 on page 93 illustrates the activities to be added for `Service Claim Exception Process`. The activities in the diagram use the following color convention. It is not mandatory to do so, but any such convention used can be useful for readers to understand the workflow:

   - Blue for Human task activities
   - Red for Rule usage activities
   - Green for Database activities

   Note: For guidance about how to model processes, see *Deploying business rules* at:

   `http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.wle.editor.doc/modeling/topic/establishing_process_flow.html`

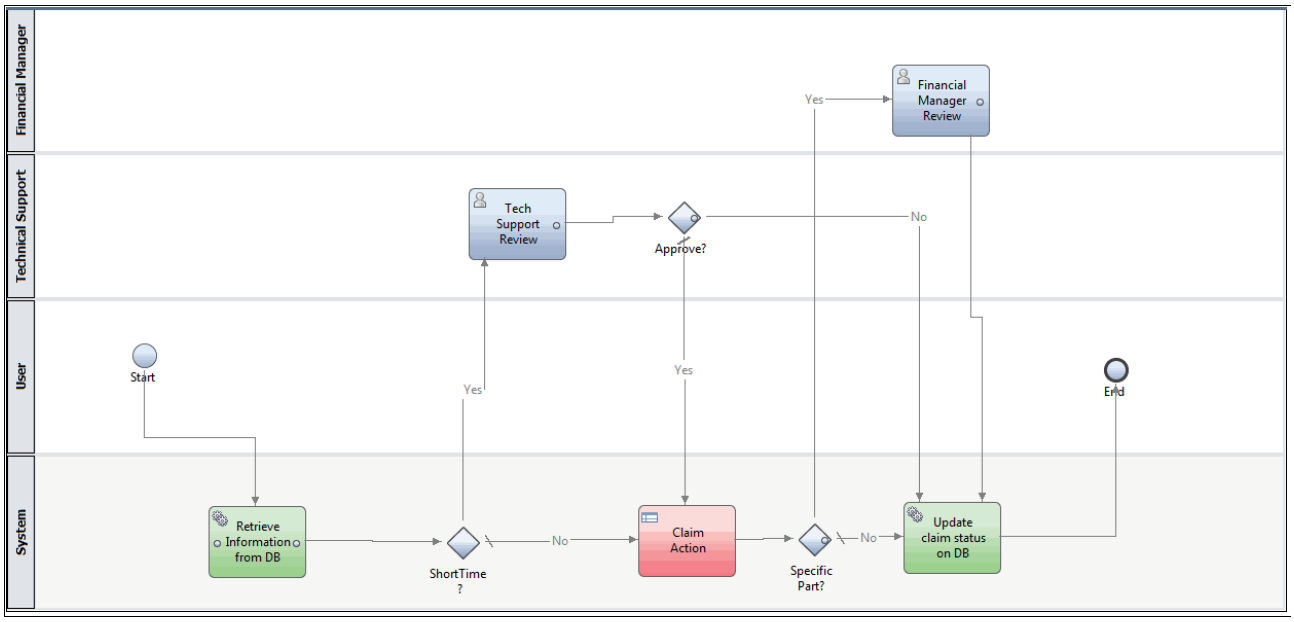


Figure 4-43 Service Claim Exception Process diagram

Table 4-4 describes each of the activity steps of the `Service Claim Exception Process`. Use the information shown in this table to create the process diagram shown in Figure 4-43.

Each activity step in the business process needs to be assigned a *task type* and the implementing *service*. Table 4-4 lists all the task types and service types that are available in the Business Process Manager.

Table 4-4 Service Claim Exception Process tasks

Steps or activities	Description	Task type	Service type
Start event	Trigger for the process to start.	N/A	N/A
Retrieve information from database	Automated database lookup. Using the product serial number and tracking number, retrieve the information about the claimed product from the database.	System task	Integration service
Decision gateway	Split the execution path based on the parameter of the process input.	N/A	N/A
Technical support review	Technical support reviews the claim, along with the comments provided by the user.	User task	Human service
Decision gateway	Split the execution path based on the result of the Technical support review, that is, whether or not it is approved.	N/A	N/A
Claim action	Execute rules. Based on the serial number parameter, execute the rule to determine if the claim requires additional management review.	System Task	Decision Service
Decision gateway	Split the execution path based on the result of the previous step, in which the rule is executed.	N/A	N/A

Steps or activities	Description	Task type	Service type
Financial Manager Review	A financial claim manager must review any claim where the value (as determined by Claim Action) is high, for example if the action is to replace a high value part with a new one.	User task	Human service
Write claim status to the database	All claims (both approved and rejected) are written to an external database with the status of the claim.	System task	Integration service

4.2.5 Add a rule invocation for rules-related activities

Note: In this section, we create a Decision Service that uses the Business Rules application (`ServiceClaimsRulesApp`) described in 4.1.4, “Overview of business rules” on page 56. It is assumed that you have completed 4.1.4, “Overview of business rules” on page 56.

Our `Service Claim Exception Process` contains activities with a Decision service. This type of service is used when you want a decision or condition in a business rule to determine which process implementation is invoked.

Create a new Decision Service

To create a new Decision Service, follow these steps:

1. Open the Process Designer and navigate to the Designer view.
2. On the upper left corner of the window, click the **plus (+)** sign to the right of Decisions, and then select **Decision Service**, as shown in Figure 4-44.

Figure 4-44 Create a new Decision Service

3. Name the service `ClaimActionOnSerialNumber` as shown in Figure 4-45, and click **Finish**.

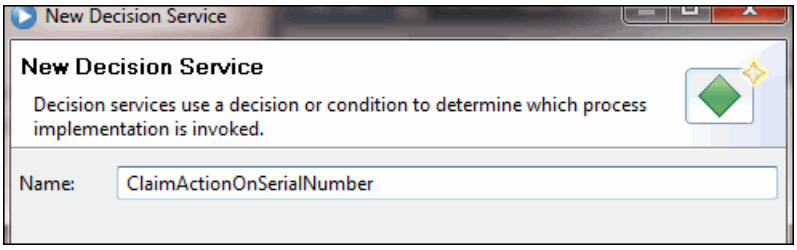


Figure 4-45 Naming the new Decision Service

A Decision Service contains one or more of the following components:

- Business Action Language (BAL) Rule

  You can use the rule editor in this component to author business rules using BAL. BAL is a declarative language that relates business concepts to business data and actions.

- JRules Decision Service

  Business Process Manager integrates with IBM WebSphere ILOG JRules and the WebSphere Operational Decision Manager server using the JRules Decision Service component. You can use this rule component to connect to and implement rule applications that are available on a JRules Rule Execution Server.

- Decision table

  The Decision table component is a rule table. Each row represents a Boolean condition that evaluates to true or false at run time. When a rule evaluates to true, the JavaScript expression that you provide as the rule action is started.

For the `Service Claim Exception Process`, we have chosen to use JRules Decision Service to create our Decision Service. This way, we can connect to the WebSphere Operational Decision Manager server and use the Business Rules application that we created in Section 4.1.4, “Overview of business rules” on page 56.

When building a Decision Service, follow these guidelines:

- Build your rule hierarchy so that rule conditions are ordered from most complex to least complex.
- Create a final condition that is a catch-all rule. This rule is necessary if you cannot guarantee that the variable you want to modify in a rule will be set before running the process that triggers your Decision Service.
- Consider encapsulating your rules in a single-function Decision Service, one that allows the service to be available to any other part of the process application that might need the same rule logic.

Note: For guidance about building a Decision Service, see *Deploying business rules* at:

http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.wle.editor.doc/topics/crule_ruleserv_ovw.html

Connect to the WebSphere Operational Decision Manager

To connect to the WebSphere Operational Decision Manager server:

1. First, the server must be added to the Process Designer. In order to do that follow the instructions in the link:

   http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.admin.doc/managinglib/topic/adding_servers.html?lang=en

2. Double-click the Decision Service **ClaimActionOnSerialNumber** that you created in “Create a new Decision Service” on page 94.
3. After the service diagram opens, drag a JRules Decision Service from the palette onto your diagram and name it `JRules Serial Number decision`.
4. Select the arrow tool from the palette, link the start node to the newly created Decision Service, and then link that service to the end node.
5. With the JRules Decision Service component selected, click the Implementation option on the Properties tab.
6. In the Discovery section, select the Server from the drop-down box, SOAP port (if the Rule Execution Server is running on WebSphere Application Server), and the user name and password (in a secure connection). This connects you to a Rule Execution Server that contains the deployed rule applications (Rule Apps) that you want to use.

Tip: The SOAP port, user name, and password fields accept embedded JavaScript expressions, so you can use variables to provide those values.

7. Select the **Implementation** option in the Properties tab (see Figure 4-46) followed by the **RulesServer** from the server drop-down menu.

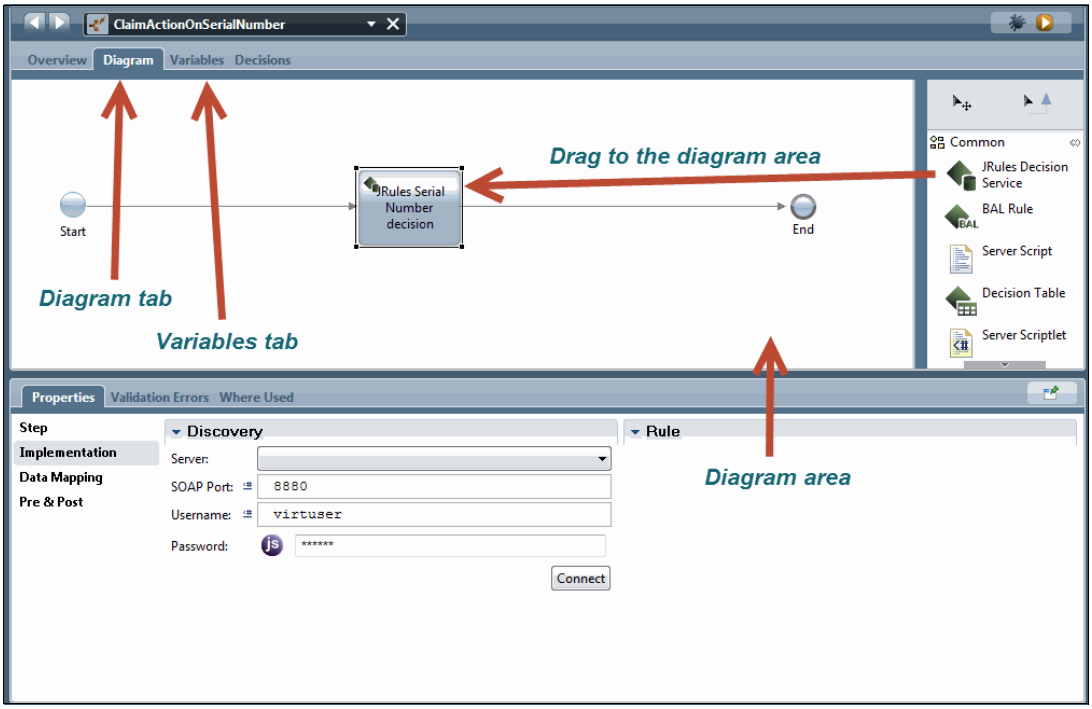


Figure 4-46 Connect to the WebSphere Operational Decision Manager

Note: If your Decision Server is not started, start the same server that you used to build the service. The link is included in Step 1.

8. Next, enter the user name and password that authenticates to your Decision Server instance, and click **Connect** (see Figure 4-47).

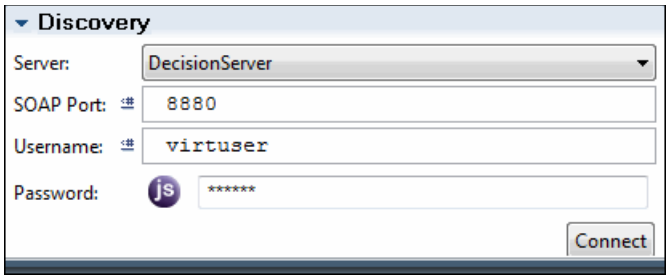


Figure 4-47 Discover the Rules Application to link with Decision Service

9. In the Rule section, select the Rule App that you want, followed by the version that you want to use. For the `Service Claim Exception Process`, use the `ServiceClaimsRuleApp` Rule App and the `ServiceClaimsRules` Ruleset, as shown in Figure 4-48.

> **Important:** If a secure connection to the Rule Execution Server has not been established, the menu is not populated. In this case, manually enter the name and version of the Rule App and Ruleset that you want to use. The names must be accurate for the next step to work.

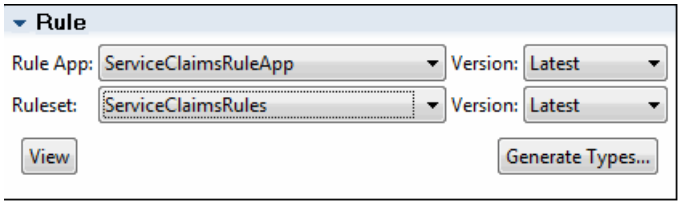


Figure 4-48 Discover the Rules Application to link with Decision Service

10. Click **Generate Types**.
11. In the Generating Types Wizard, make sure the *Generate request/response wrapper types* option is *not* selected (see Figure 4-49) before clicking **Next**.

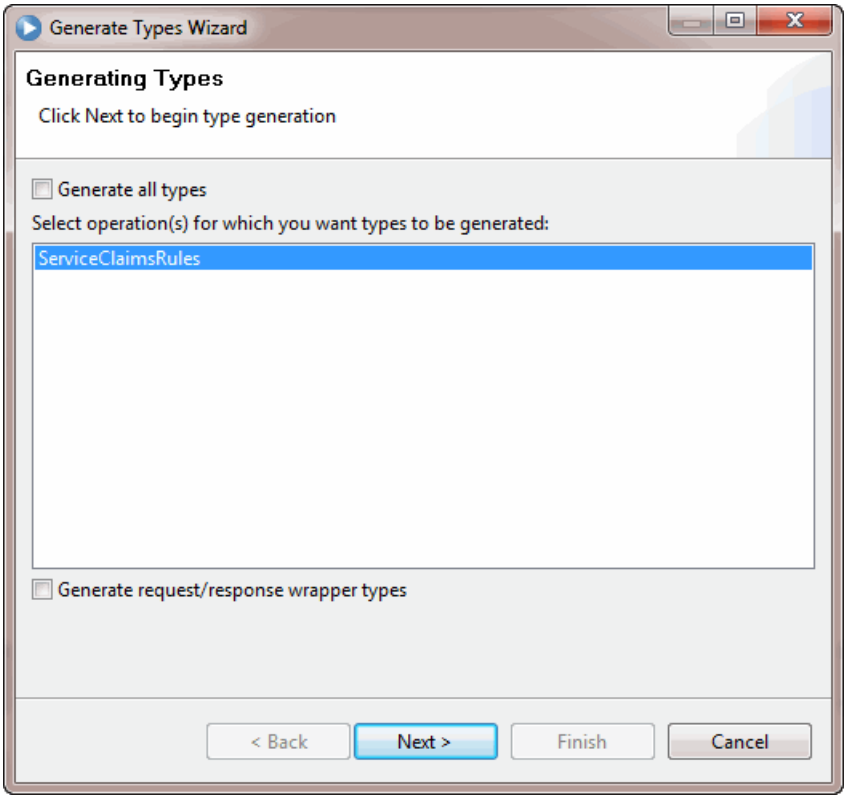


Figure 4-49 Generate data types

12. Click **Next** through the remaining wizard windows until the type generation is complete, and then click **Finish** to accept default type generation (see Figure 4-50).

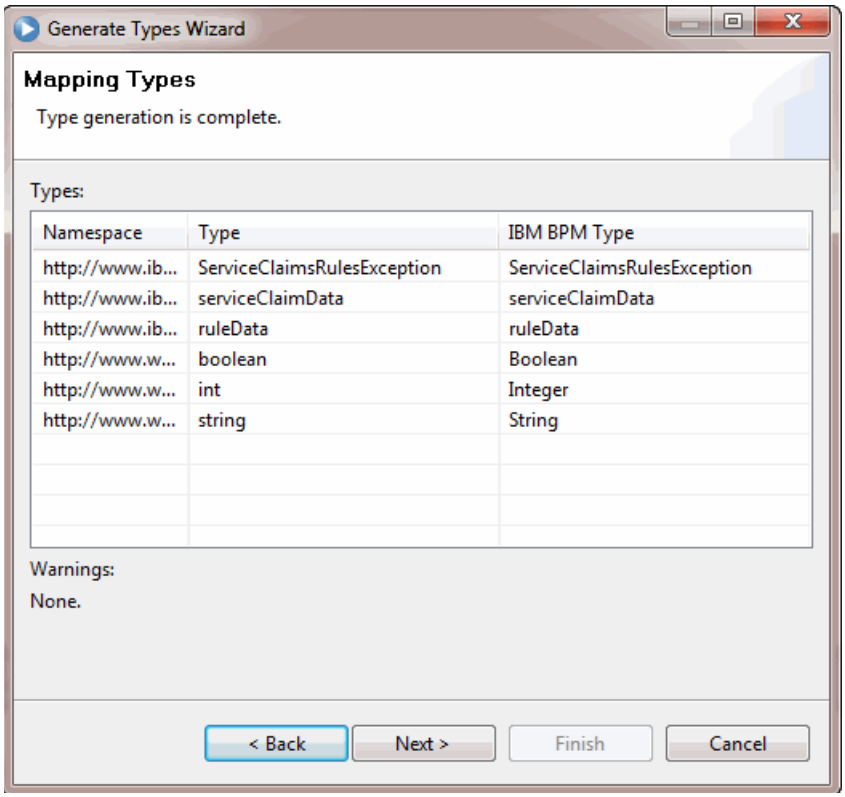


Figure 4-50 Generate data types

After verifying that your business objects have been created, examine them to check that they match with the parameters you defined when creating your rule project:

1. Click the Variables tab (see Figure 4-46 on page 96) and click **Add Input.**
2. Specify the name of the input data type as `input` and the type as `ruleData`, and select the **Has Default** option.
3. Click **Add Output**, and specify the name of the output data type as `output` and the type as `ruleData`.
4. Click the Diagram tab (see Figure 4-51 on page 99).
5. Click **back** on the JRules Rule Service that you created.
6. Select the Data Mapping section (see Figure 4-51 on page 99) and enter a value of **0** for the Decision ID.

> **Note:** The value for the Decision ID can be used for correlating business processes and decisions by passing the process instances ID as a parameter. This scenario does not use the Decision ID for any purpose, but it is a mandatory field. A value must be provided.

7. Map the `tw.local.input` input type to `ruleData`.
8. Map the `tw.local.output` output type to `ruleData`.

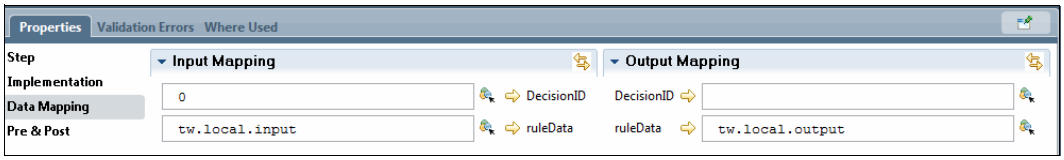


Figure 4-51 Input and output data mapping

> **Note:** For guidance about adding a JRules Decision Service component to a service, see *Adding a JRules Decision Service component to a service* at:
>
> http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.wle.editor.doc/topics/building_rule_service_C.html?lang=en

Test the Decision Service

To test a rule component and the rules it contains, you can load the Decision Service with default data and then step through the activities in your business process diagram to see the generated process data as it interacts with your defined business rules. For example, if you set a breakpoint on an activity that has an associated Decision Service, you can make sure that the Decision Service is producing the data that you expect, and that it is not producing any error messages or exceptions.

To create a test process to execute the rules:

1. In the Process Designer, click the **plus (+)** sign next to Processes and choose to create a new BPD.
2. Name the business process diagram `TestProductRule` and then click **Finish**.
3. Click **Decisions** and drag the `ClaimActionOnSerialNumber` service into the system swimlane of the process diagram.
4. Connect the start node to the Decision Service using a connection arrow from the palette.
5. Connect the Decision Service to the end node in similar fashion to Step 4.
6. Click the Variables tab, click **Add Input**, and specify the name of the input data type as `input` and the type as `ruleData`.
7. Select the **Has Default** option and enter the value "`S724`" in the serialNumber field.
8. Click **Add Output** and specify the name of the output data type as `output` and the type as `ruleData`.
9. Click the Diagram tab and select the **ClaimActionOnSerialNumber** service.
10. Click the Data Mapping tab, set the input mapping to `tw.local.input` and the output mapping to `tw.local.output`.
11. Now your process and rule are ready to test. Click **Play**, and if asked to switch the view to the Inspector view, click **Yes**.

> **Note:** You will see your process execute in the Inspector view. Wait until the process and Decision Service execute. The first time your process executes, all the service artifacts are compiled, so it might take 5 - 30 seconds to complete. Refresh the view every few seconds, and do not proceed until the status of your activity is complete and the process instance status is closed.

12. On the Execution State tab, select **TestProductRule** and then click the **Variables** tab to view the input and output parameters.

13. Select the input parameter to verify that the XML document contains the "`S724`" input parameter that you set.
14. Click the output parameter to verify that the action is set to `Attention` in the XML document.

Figure 4-14 on page 61 shows the logic of the rule that was written earlier in "Create a rule" on page 61. Try changing the input parameters for the test and observe the output change according to the logic.

> **Note:** For more testing guidance, see *Testing a Decision service* at:
> http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.wle.editor.doc/topics/trule_service_testing.html?lang=en

4.2.6 Add the user interface for human task activities

Users interact with, and participate in, a business process through user interfaces. Human services (*human tasks*) represent the actions that these users perform within the process, and *coaches* are the user interface for these services. In a Business Process Manager process, users interact with services and the service data using coaches. Coaches feature reusable *coach views*, which are reusable user interfaces that you can create and customize. Coach views consist of one or more other coach views, data bindings, layout information, and behavior.

To create a new human service and add the appropriate coaches to the user interface for this service:

1. Create a new human service by clicking **Human Service**, as shown in Figure 4-52.

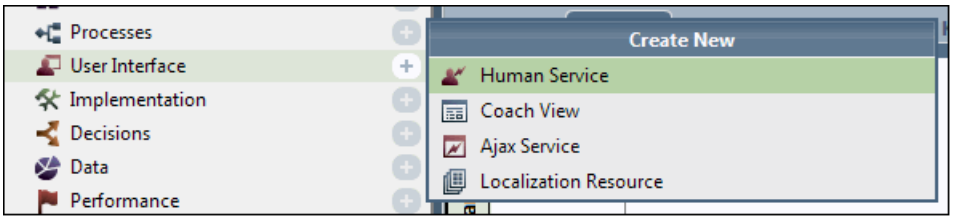


Figure 4-52 Create a new human service

2. Enter a name for the service (see Figure 4-53) and click **Finish**.

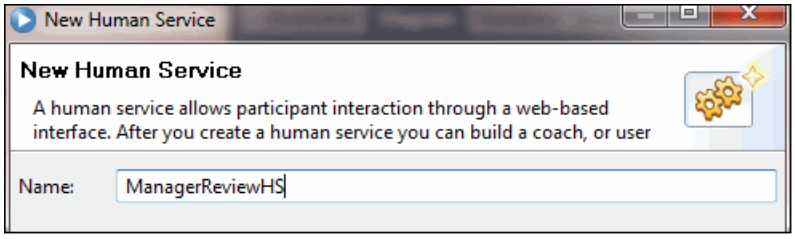


Figure 4-53 Name the human service

3. Add a coach, named *ReviewClaims*, to the Human Service diagram, as shown in Figure 4-54 on page 101.

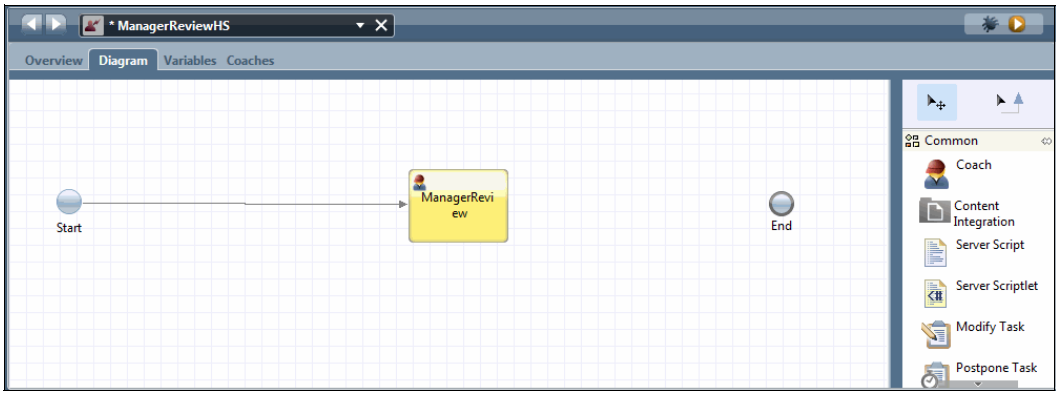


Figure 4-54 Add the coach to the human service diagram

4. Add the UI controls from the right side palette to the coach panel (see Figure 4-55), and link the controls to input and output variables as needed.

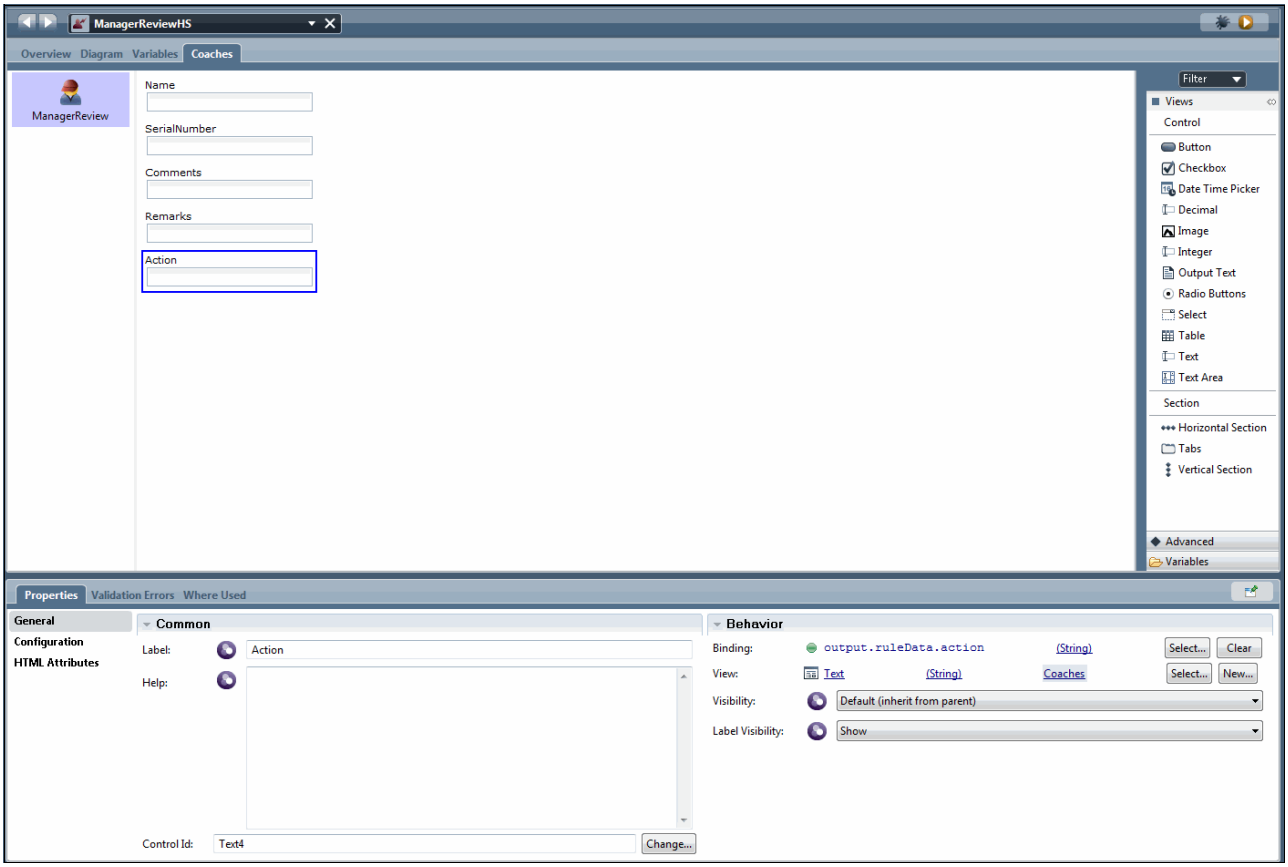


Figure 4-55 Add UI controls to the coach panel

After adding the UI controls, the ReviewClaims coach looks as shown in Figure 4-56 on page 102.

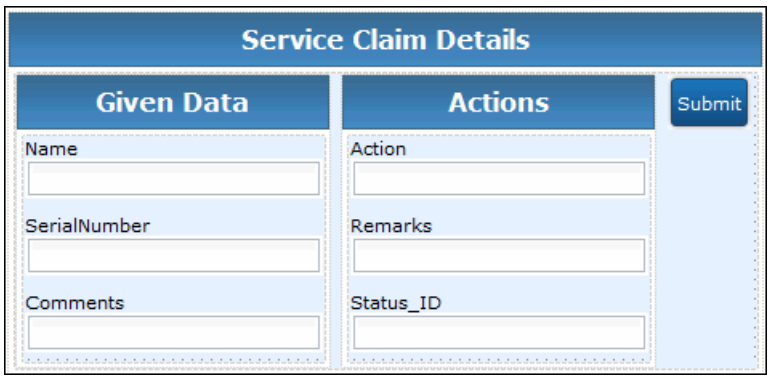


Figure 4-56 The ReviewClaims coach view

5. Navigate to the Warranty Exception Process, and link the Review Claims and Manager Review activities to the human service, as indicated in Figure 4-57 and in Figure 4-58.

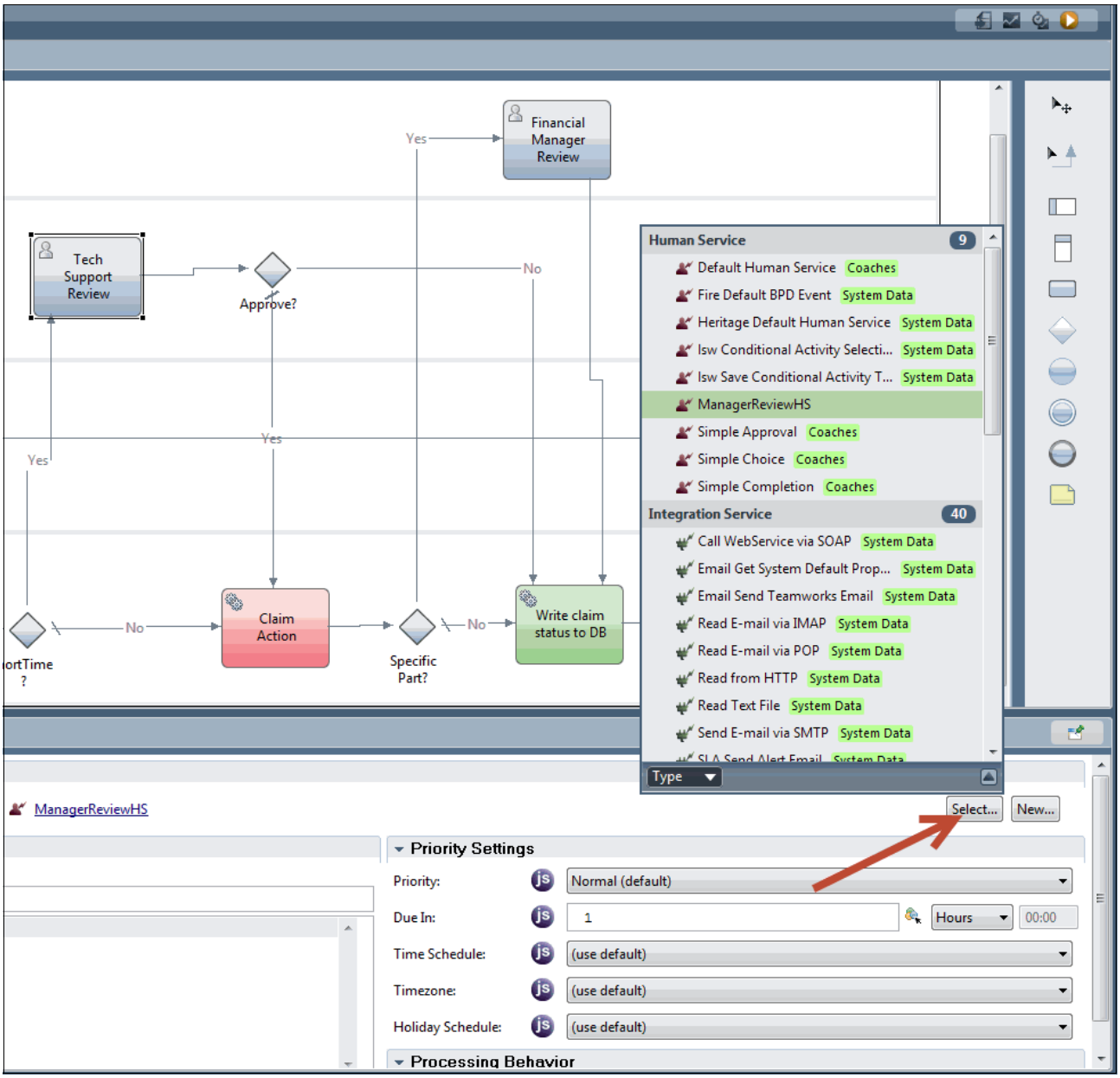


Figure 4-57 Link the human service to the business process diagram activities

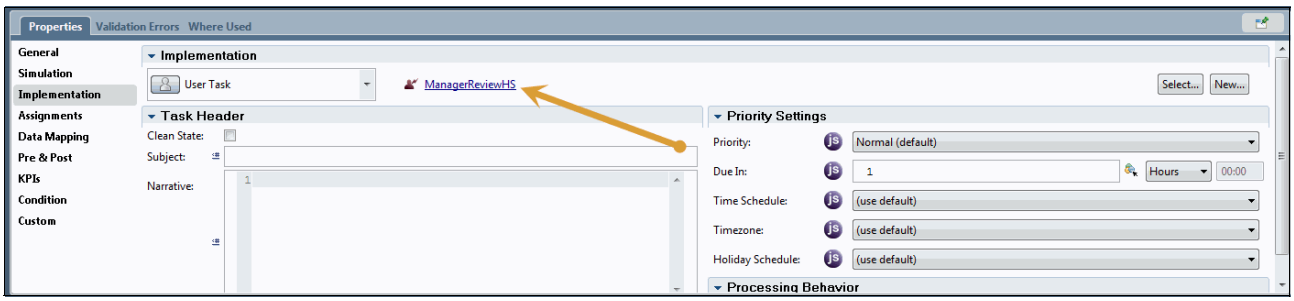


Figure 4-58 Link the human service to the business process diagram activities

6. Modify the decision gateways in the flow to use the output from Step 5. This determines the processing paths.

Now we have the event-related activities of the business process diagram linked to the business rules, and the decision gateways are using the information to split the process execution path. We have integrated the business process with the business rules in a loosely coupled way, according to the pattern “Business process calling business rules” on page 52.

Note: For guidance about setting up a user interface for your human task, see *Setting up a user interface for your human task* at:

http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.wid.bpel.doc/topics/tnclient.html?lang=en

4.2.7 Create an Event Rule to trigger the business process

The business process that is defined is triggered as follows:

1. A web user interface sends a serial number through the WebSphere MQ queue.
2. The WebSphere MQ message is received by IBM Integration Broker flow, which converts the WebSphere MQ message into a web service invocation.
3. The `Service Claim Exception Process` is triggered when each WebSphere MQ message is received to raise an event by setting a parameter.

In this section, we define:

- The event that produces an action when the Service Claim Exception situation was detected by the WebSphere Operational Decision Manager
- How to add a message-based trigger to our `Service Claim Exception Process` through the use of Undercover Agents and other services is described.

Creating a new business object

Business objects define the business data that is used to describe the service and component contracts, in addition to the business data that the components manipulate. Basically, they represent the data that flows between services in an application. We need to define the data that will trigger the exception process.

To start, define a new business object called `ServiceClaimEventType`:

1. Open the Process Designer to the Designer perspective, then open the process application you have been working on.
2. Next to the Data field, click the **plus (+)** icon and select **Business Object**.
3. Name the Business Object `ServiceClaimEventType`.
4. Add the following fields to the ServiceClaimEventType:
   - `SerialNumber` with type String
   - `TimeBetweenEvents` with type Integer
   - `TimeOfLastEvent` with type Date
   - `TrackingNumber` with type String
   - `isDuplicate` with type Boolean
   - `Name` with type String
   - `Comments` with type String
5. Save the business object.

Add a message-based trigger to the business process

To create a message-based trigger, we create the following general system services:

- Start Event
- Start Receiver

To begin:

1. Click the **plus (+)** sign next to the Implementation and select **General System Service**.
2. Name the new service `Start Event`, and click **Finish** to create the new service (see Figure 4-59).

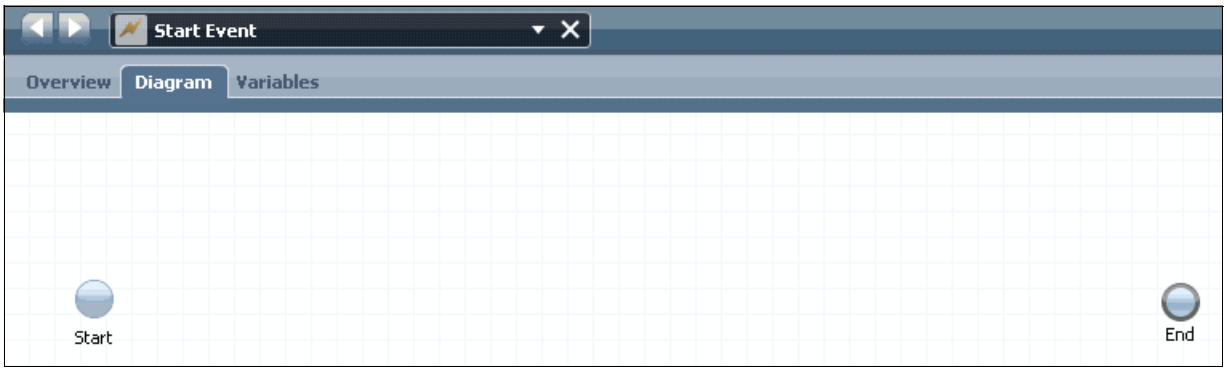


Figure 4-59 Start Event service diagram

3. Navigate to the Variables tab, click **Add Input,** name the variable `ServiceClaimEvent`**,** and change its type to ServiceClaimEventType.
4. Click **Add Output**, name the variable `ServiceClaimEvent`**,** and change its type to ServiceClaimEventType.
5. Navigate to the Diagram tab, and connect the start node directly to the end node using the arrow tool from the palette.
6. Save the activity.
7. Next to Implementation, click the **plus (+)** sign and select **General System Service**.
8. Name the new service `Start Receiver` (see Figure 4-60).

> **Important:** This is the second general system service that you need to create. It is important not to confuse the two services, so pay close attention to the name of the service that you are working in as you proceed.

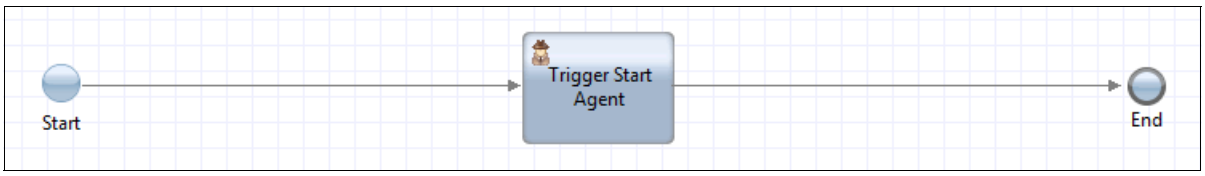


Figure 4-60 Start Receiver service diagram

9. Navigate to the Variables tab, click **Add Input**, name the variable `ServiceClaimEvent`, and change its type to `ServiceClaimEventType`.
10. For the input parameter, select the **Has Default** option.

11. Click **Add Output**, name the variable `ServiceClaimEvent`, and change its type to `ServiceClaimEventType`.
12. Navigate to the Diagram tab, drag an undercover agent (UCA) from the palette to the diagram, and name the UCA activityTrigger `Start Agent`.
13. Navigate to the Implementation tab, click **New** to create a new UCA, as indicated in Figure 4-61, and name it `Start Agent`, as shown.

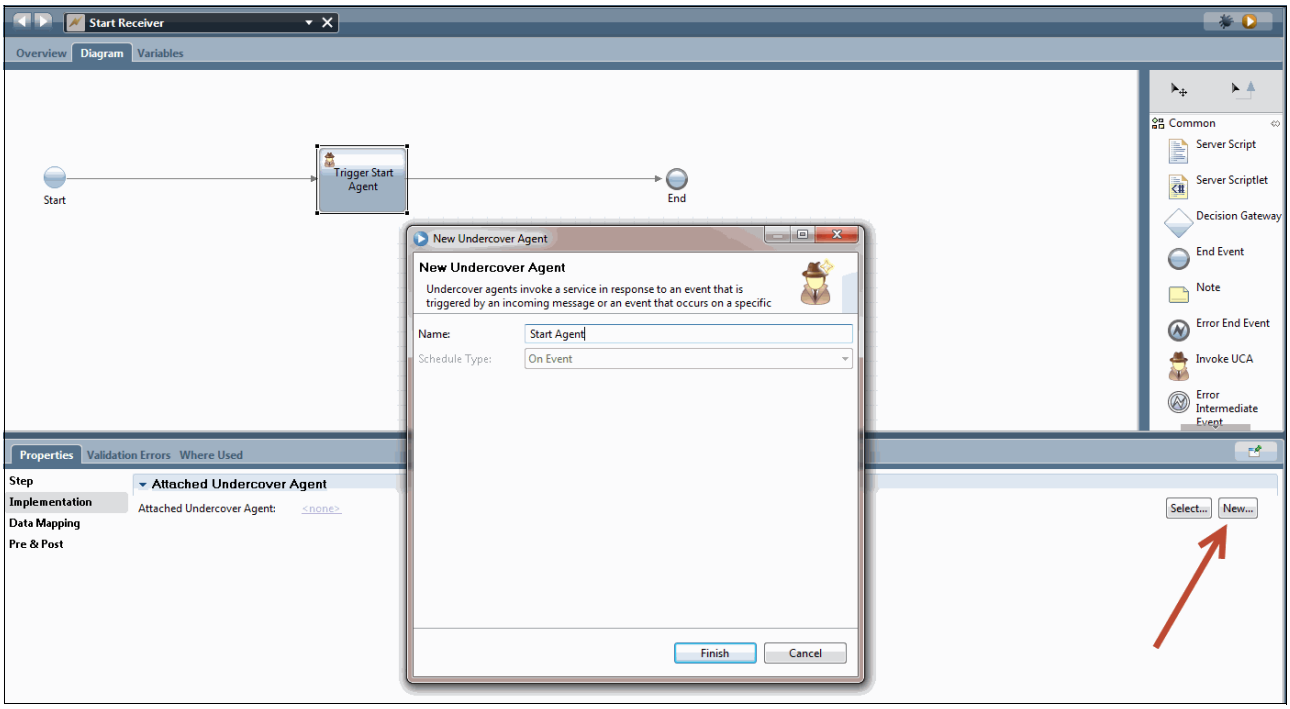


Figure 4-61 Creating a new undercover agent

> **Note:** To learn about undercover agents and their use, see *Understanding and using undercover agents* at:
>
> http://www-01.ibm.com/support/knowledgecenter/SSFPJS_8.0.1/com.ibm.wbpm.wle.editor.doc/modeling/topic/using_undercover_agents.html

14. Click **Select** and choose the Start Event service.
15. Click the Data Mapping tab and enter the value `tw.local.ServiceClaimEvent`.
16. On the diagram, connect the start node to the UCA Activity named Trigger Start Agent using the arrow tool on the palette.
17. Connect the Trigger Start Agent to the end node in the same way as it was done in Step 16.
18. Click the **plus (+)** sign next to Implementation, choose **Web Service**, and name the new Web Service `StartServiceClaimEventWS`.
19. In the Operations pane, click **Add**, and name the operation `StartServiceClaimEventInBPM`.
20. Click **Select** and choose the **Start Receiver** service.

Now that the implementation is complete, you can create a new web service to expose the Start Event of the `Service Claim Exception Process` as a callable web service using SOAP:

1. Attach the Start Receiver service to the newly created web service (as shown in Figure 4-62).

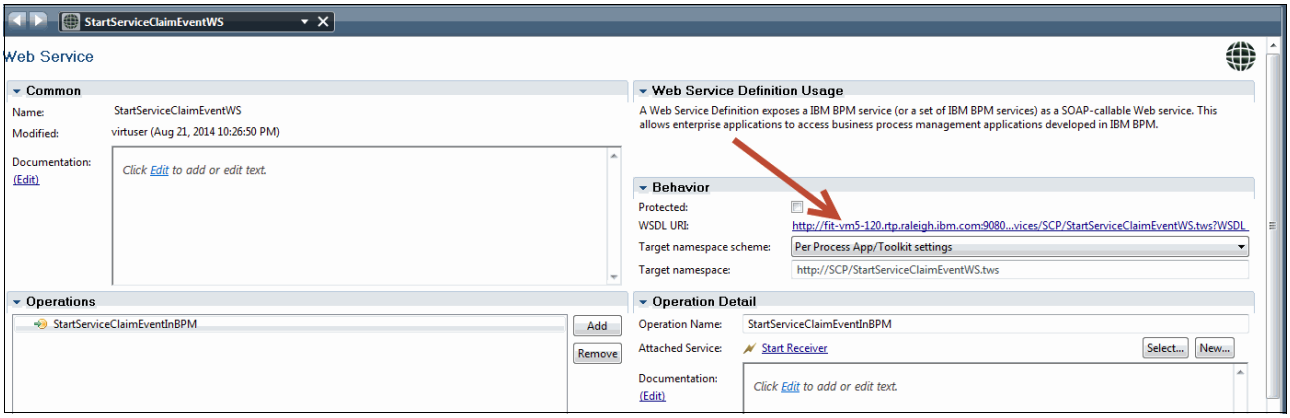


Figure 4-62 Attach the Start Receiver service to the new web service

2. In the Behavior pane, open the following address in a new browser window to display the WSDL:

   `http://ibmbpm:9080/teamworks/webservices/SCP/StartServiceClaimEventsWS.tws?WSDL`

3. Copy the address and save it in a text file.
4. Navigate to the `Service Claim Exception Process` and drag a Start Event into the diagram.
5. Name the event, `Start Event`, and place it at the start of the process, that is, before the Retrieve Information from database named activity.
6. Connect the Start Event to the next activity, that is, the Retrieve Information from database named activity (see Figure 4-63 on page 108).

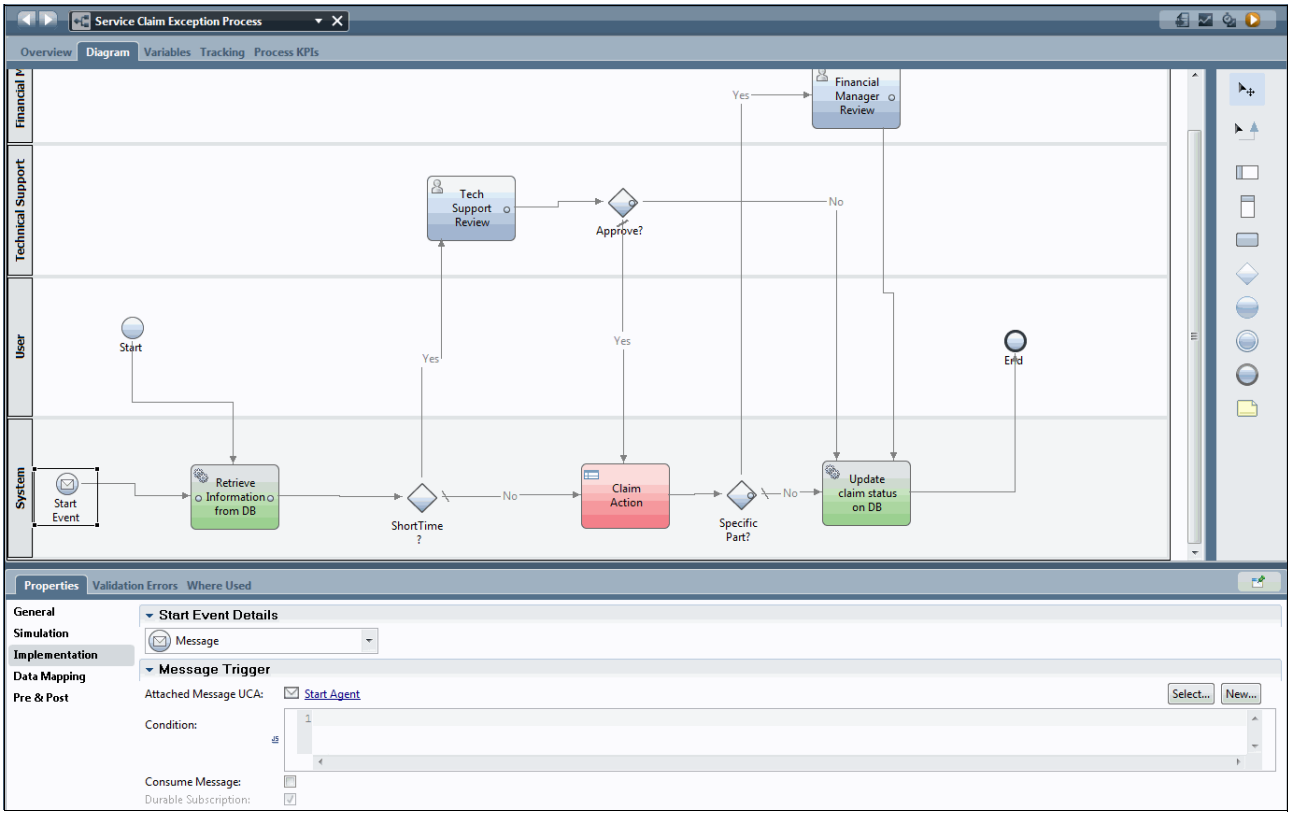


Figure 4-63 Connect the Start Event

7. Select the Start Event activity.
8. In the Properties - Implementation pane, select **Message Start Event** from the Start Event Details drop-down menu, and attach the new UCA with this activity (see Figure 4-63).

4.2.8 Add database integration

An integration service enables your processes to communicate with an external system to retrieve, update, or insert data. Because you want each claim request data to be retrieved and claims data (from both approved and rejected claims) to be persisted into a database, you will create integration services called `RetrieveInfo` and `Persist` to perform the actions.

Before starting, however, you need to have a database defined and a data source that references that database on the Process Center WebSphere Application Server. The data source JNDI name is defined as "`jdbc/serviceClaimDS`" for this solution. The data table used is the SAW_REQUESTS table in the VIRTUSER schema, which is defined in Appendix C, "Database definitions" on page 239. This section describes the detailed steps to create the `Persist` action. You can use the same steps to create `RetrieveInfo` action:

1. Open Process Designer and then you are in the Design view.
2. Click the **plus (+)** sign next to Implementation and select **Integration Service**.
3. Name the Integration Service `Persist`.
4. In the diagram, drag a Server Scriptlet, a Server Script, and a Nested Service onto the diagram from left to right.

5. From the right side palette, drag and add the Server Scriptlet, Server Script, and Nested Service activities onto the diagram from left to right and join them in sequence.
6. Name the activities that you dragged in Step 5. Name them in the order that you dragged them (see Figure 4-64):

   a. `SQL statement definition`
   b. `Set SQL parameters`
   c. `SQL execute statement`

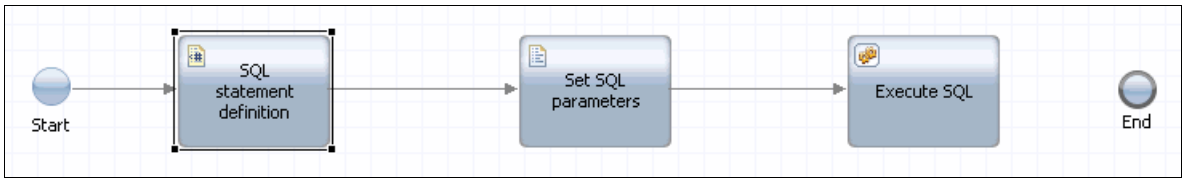


Figure 4-64 Adding activities to the service diagram

Each activity defines how an SQL statement with given parameters can be executed on Business Process Management process.

7. Click the **plus (+)** sign next to Data, select **Business Object**, and call the new Business Object `TransactionType`.
8. Add the following parameters to this type by clicking **Add**:
   - `RequestID` Integer
   - `SerialNumber` String
   - `StatusID` Integer
   - `Remarks` String

   Note: If you are using your own schema and table, these types must match those in the table you will insert into.

9. Open the Persist service that you created, select the Variables tab, and add the following:
   - An input variable named `Transaction` with type TransactionType
   - A private variable named `sqlStatement` with type String
   - A private variable named `sqlParameters` with type SQLParameter
10. Select the **Is List** option for sqlParameters, followed by the SQL statement definition.
11. Navigate to the Implementation tab and enter `UPDATE VIRTUSER.SAW_REQUESTS SET REMARKS=?, STATUS_ID=? WHERE SERIAL_NUMBER=? and STATUS_ID=0`, or an SQL update statement that matches your own schema. For binding, click **Select**, and select **tw.local.sqlStatement**.
12. Select **Set SQL parameters**.
13. Click the Implementation tab and copy in the code from Example 4-5, and shown in Figure 4-65 on page 110.

Example 4-5 SQL parameters

```
tw.local.sqlParameters = new tw.object.listOf.SQLParameter();
tw.local.sqlParameters[0] = new tw.object.SQLParameter();
tw.local.sqlParameters[0].value = tw.local.Transaction.remarks;
tw.local.sqlParameters[0].type = "VARCHAR";
tw.local.sqlParameters[1] = new tw.object.SQLParameter();
tw.local.sqlParameters[1].value = tw.local.Transaction.statusID;
tw.local.sqlParameters[1].type = "INTEGER";
```

```
tw.local.sqlParameters[2] = new tw.object.SQLParameter();
tw.local.sqlParameters[2].value = tw.local.Transaction.serialNumber;
tw.local.sqlParameters[2].type = "VARCHAR";
```

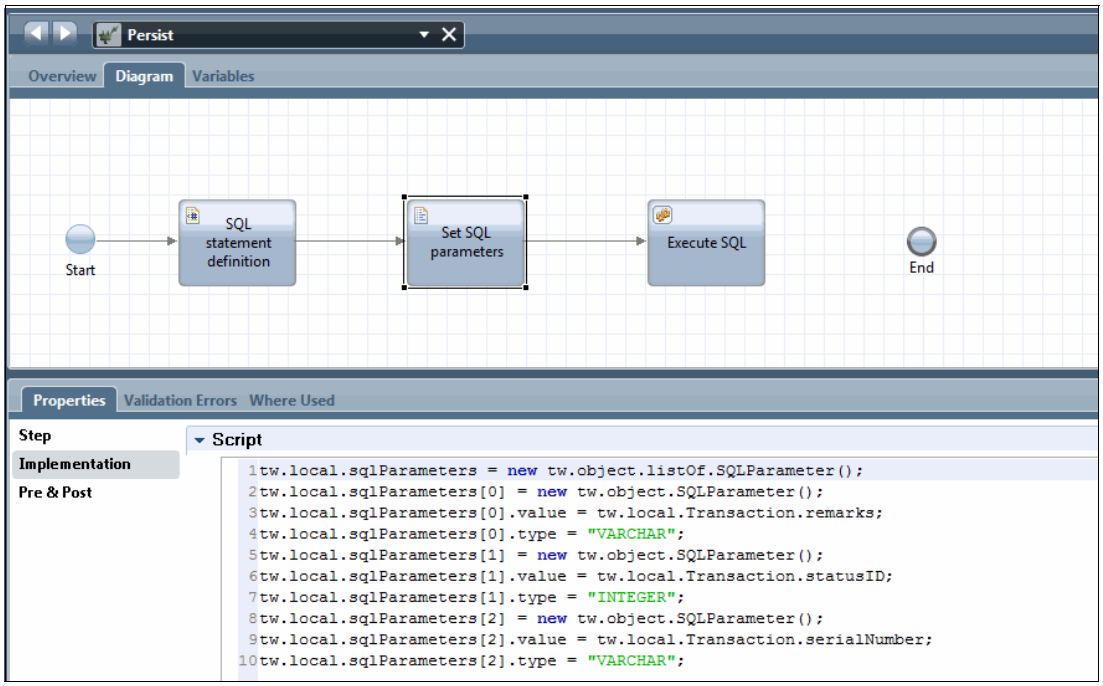


Figure 4-65 Setting values for the SQL statement using Server Script

14. Select **ExecuteSQL**.
15. Navigate to the Implementation tab, and, in attached service, click **Select**.
16. Choose **SQL Execute Statement**.
17. Navigate to the Data Mapping tab. Next to SQL, enter `tw.local.sqlStatement`, and next to parameters, clear the default box and enter `tw.local.sqlParameters` (see Figure 4-66 on page 111).

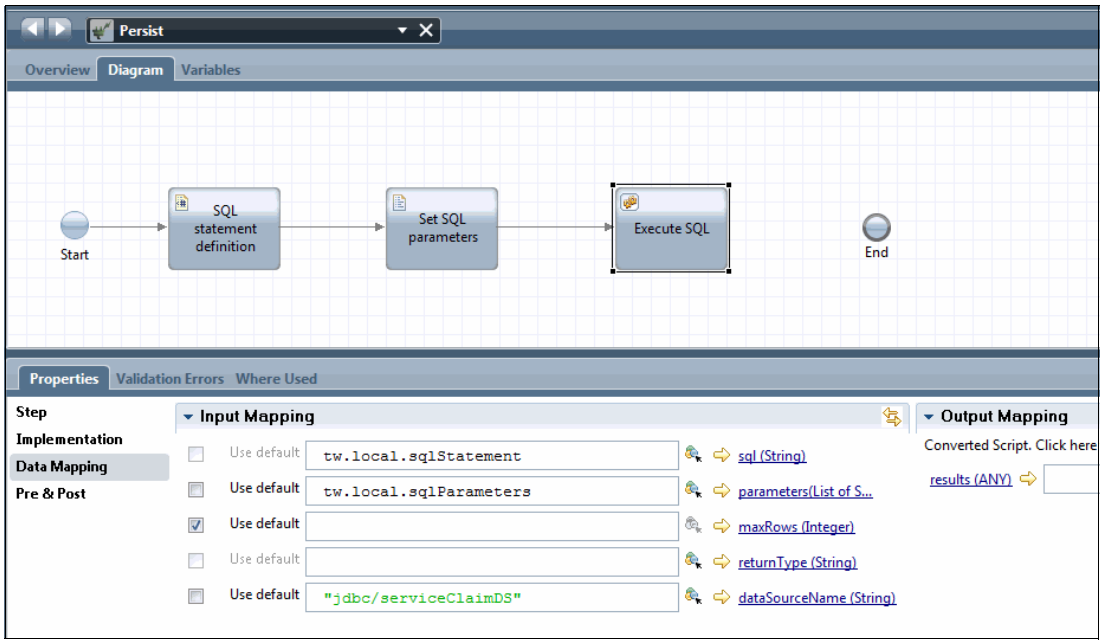


Figure 4-66 Adding data mapping for the SQL Execute Statement

18. Next to **dataSourceName**, enter the data source that you have configured in the WebSphere Application Server.
19. To create a simple test process, click the **plus (+)** sign next to processes, select **Business Process Definition,** name the process `TestDatabase`, and click **Finish**.
20. In the diagram, drag the Persist activity from the integration list into the System swimlane.
21. Connect the start node to the Persist activity with the arrow tool, and connect the Persist activity to the end node with the arrow tool.
22. Navigate to the Variables tab, click **Add Input**. Name the variable transaction and select the type `TransactionType`.
23. Click the **Has Default** option. Modify the default values to non-confidential values, for example:
    - RequestID: `0`
    - SerialNumber: `F547`
    - StatusID: `1`
    - Remarks **“”**
24. Navigate to the Data mapping tab, and for the input mapping, enter `tw.local.Transaction`.
25. Save the process.
26. Click the **Play** button, and if prompted, click **Yes** to switch views.
27. The data is now in the database. Use any database tool that you have available to verify that the data has been inserted into the database.

Apply the steps from section 4.2.8, “Add database integration” on page 108, from the first Step to Step 20. This will create the `RetrieveInfo` action and use `select SERIAL_NUMBER, TRACKING_NUMBER, name, comments from SAW_REQUESTS where SERIAL_NUMBER=? and TRACKING_NUMBER=?` SQL statement. The parameters for that activity will be similar to those in Example 4-6 on page 112.

Example 4-6 SQL parameters for the RetrieveInfo activity

```
tw.local.sqlParameters = new tw.object.listOf.SQLParameter();
tw.local.sqlParameters[0] = new tw.object.SQLParameter();
tw.local.sqlParameters[0].value = tw.local.inputRetInfo.serialNumber;
tw.local.sqlParameters[0].type = "VARCHAR";
tw.local.sqlParameters[1] = new tw.object.SQLParameter();
tw.local.sqlParameters[1].value = tw.local.inputRetInfo.transactionNumber;
tw.local.sqlParameters[1].type = "VARCHAR";
```

4.2.9 Deploy the process to Process Center

Once the process is saved on Process Designer, the current version of the process is directly deployed to Process Center. In order to keep the version tracking, you can get a snapshot of the current process from the Process Designer.

Mainly, production servers are offline servers, on which you cannot deploy the applications directly. In that case, refer to link in the note below for more information.

Note: For guidance about deploying processes to offline servers, see the following site:

http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.admin.doc/topics/releasing_installing_procs_D.html?lang=en

5

Creating solution artifacts

This chapter describes the process of creating custom artifacts for the scenario described in 1.2, “The business scenario” on page 14. We describe the creation of web and mobile applications and the complete environments that support the scenario, including the integration layer with integration flows. For the web application, both the Liberty profile and the WebSphere Application Server full profile are discussed. As the foundation for the mobile application, we will be using IBM Worklight.

In this chapter, the following topics are discussed:

- 5.1, “Prepare the integration environment” on page 114
- 5.2, “Create the web application for the scenario” on page 123
- 5.3, “Prepare the Liberty profile environment for your web application” on page 132
- 5.4, “Prepare WebSphere Application Server full profile environment for web application” on page 136
- 5.5, “Create the mobile application for our scenario” on page 139

© Copyright IBM Corp. 2014. All rights reserved.

5.1 Prepare the integration environment

In the overall solution, the integration environment plays a key role in enabling the components to communicate using their native mechanisms and protocols. The business scenario described in this book (see 1.2, “The business scenario” on page 14) is an example of the variety of commonly used technologies for communication. Follow this section for guidance in preparing and implementing the integration applications.

5.1.1 Integration architecture

Let us start by designing a solution for the scenario described in 1.2, “The business scenario” on page 14.

We described in 2.2, “Middleware products used for the solution” on page 20, that there will be two sources of incoming messages:

- One source will use the asynchronous method of delivering XML messages to the WebSphere MQ queue.
- One source will send a synchronous HTTP request with a JavaScript Object Notation (JSON) message.

The job of the IBM Integration Bus is to receive messages from both channels, persist in the database, and invoke a business event rule that will launch the business process.

This process can be split into three separate parts (three integration flows):

- The first integration flow listens for incoming XML messages to WebSphere MQ queue and processes the XML message.
- The second integration flow is invoked by an HTTP request and processes the JSON message. The result of both integration flows is then sent to an additional WebSphere MQ queue, which is not accessible by the external clients.

  The messages that are put into this internal queue will trigger the third integration flow, for which the execution logic is common to all incoming messages.
- The third integration flow persists the data in the database, writes the event to a log file, and sends a SOAP request to trigger an event rule that is deployed on the IBM Operational Decision Manager.

This concept is depicted graphically in Figure 5-1 on page 115.

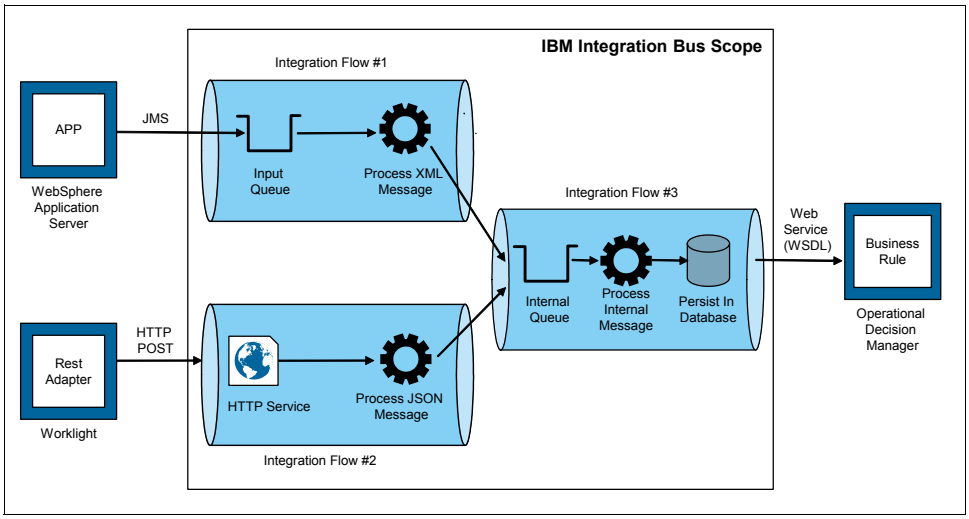


Figure 5-1 Integration flow concept for the business scenario

Tip: There are three primary reasons for using the additional integration flow with an internal queue, rather than creating two integration flows for each input:

- The common logic that triggers the same service is implemented only one time.
- Improved performance. With this approach, the work done by the Integration Bus is paralyzed, and more threads are involved in running the integration flows. This is also the reason why subflows were not used, as subflows are a logical extension of the integration flows.
- Better control over the integration process. With this design, it is possible to stop a single channel to block requests from mobile devices, or to stop the triggering flow in the case of a maintenance window (for example, administrative maintenance) that has to be done from the business process side. From the client's perspective, the solution is continuously available; however, invisible to the user is that the solution queues messages and processes them when the stopped integration flow becomes operational again.

Next, we guide you through the steps for completing the integration solution.

5.1.2 Installation

The Integration Bus consists of the following components:

- **IBM Integration Bus** runtime: This is where the integration logic executes.
- **WebSphere MQ:** This component is closely tied with Integration Bus and is used in internal processing to allow parallel execution of threads. It installs on the same system as the IBM Integration Bus runtime.
- **Integration Bus Toolkit**: Eclipse-based tool that allows to design and implement the integration flows. It can be installed on the same system with the Integration Bus or it can connect to remote Integration Bus server. It is not required to run applications on Integration Bus.
- **Integration Explorer:** This is a WebSphere MQ, Eclipse-based tool for use by administrators, enabling them to perform most administrative tasks and to monitor the

Integration Bus runtime. Like the Integration Bus Toolkit, Integration Explorer operates with local and remote runtimes.

Follow the IBM Knowledge Center installation guide for Integration Bus installation details and prerequisites. See the IBM Knowledge Center:

http://www-01.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.mft.doc/ah24100_.htm

In our scenario, we used Integration Bus Advanced version 9.

> **Note:** Integration Bus version 9 is the successor to the WebSphere Message Broker. The Integration Bus shares many features with WebSphere Message Broker, and some of the commands still refer to the old product name.
>
> For the latest details, refer to the *IBM Integration Bus technical overview* in the IBM Knowledge Center:
>
> http://www-01.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.mft.doc/ab20551_.htm

5.1.3 Configuration

With the Integration Bus runtime installed, follow these steps to begin developing the integration flows.

Create the Integration Bus instance

To create the Integration Bus instance, run the command:

```
mqsicreatebroker IB9NODE -q IB9QMGR -t
```

This command first checks to determine if there is an `IB9QMGR` defined in the system. If not, it is created, along with the necessary resources, such as internal queues that the Integration Bus requires to operate. This command produces the `IB9NODE` instance.

> **Note:** The behavior of the `mqsicreatebroker` might vary for different platforms. Refer to the IBM Knowledge Center for detailed information about its behavior on your system and the default settings it uses:
>
> http://www-01.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.mft.doc/an07080_.htm

Create the integration server

To create the integration server, follow these steps:

1. Start the Integration Bus instance, along with the WebSphere MQ queue manager by issuing the **`mqsistart`** command. This command performs initial verification tests, then starts the Integration Bus and its WebSphere MQ queue manager:

   ```
   mqsistart IB9NODE
   ```

2. With the Integration Bus running, prepare an integration server, which is a configurable runtime environment in which the integration flows are being executed. Integration servers (named execution groups in the previous WebSphere Message Broker product) can be, for example, configured with separate listener ports or connection pool settings, which allows for sharing or separating of application resources in the Integration Bus.

3. Create the default integration server by issuing the following command on the system on which you installed the Integration Bus runtime environment:

```
mqsicreateexecutiongroup -i localhost -p 1414 -q IB9QMGR -e default
```

Note: You can also create the integration server, and many other configurations, using the Integration Bus Toolkit or the Integration Explorer graphical interface. Refer to the IBM Knowledge Center for instructions and examples:

http://www-01.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.mft.doc/be10020_.htm

Create additional resources

To create additional queues used by the solution, use the commands in Example 5-1, or create them using the Integration Explorer.

Example 5-1 Creating WebSphere MQ queues

```
echo "DEFINE QLOCAL('WEB_IN') PUT(ENABLED) GET(ENABLED)" | runmqsc IB9QMGR
echo "DEFINE QLOCAL('TRIGGER_PROCESS_Q') PUT(ENABLED) GET(ENABLED)" | runmqsc
IB9QMGR
```

To enable connectivity from the WebSphere Application Server to the WebSphere MQ queue manager, define an additional access channel. Example 5-2 shows the commands for creating a channel named `WAS.CONN` on a Linux system.

Example 5-2 Creating and configuring a channel on the WebSphere MQ queue manager

```
echo "DEFINE CHANNEL(WAS.CONN) CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER('mqm')" |
runmqsc IB9QMGR
echo "SET CHLAUTH(WAS.CONN) TYPE(BLOCKUSER) USERLIST(ALLOWANY)" | runmqsc IB9QMGR
```

Note: The default behavior in WebSphere MQ version 7.5, which is used in our business scenario, is that it refuses any remote connection to a channel. This is why the second command (in Example 5-1) is used. For production environments, consider using mutual SSL/TLS authentication or WebSphere MQ Security Exits.

For more security considerations and setup, refer to the IBM technote:

http://www-01.ibm.com/support/docview.wss?uid=swg21636093

Configure open database connectivity (ODBC) for database access

The Integration Bus can use ODBC or Java Database Connectivity (JDBC) to access relational databases. In this solution, we used ODBC because it is a faster and more efficient way of accessing data. This is important for the integration component because it will handle high volumes of data. The ODBC is also easily accessible directly from embedded structure query language (ESQL) computing nodes, which will transform the messages.

The following IBM Knowledge Center article covers the steps and requirements for configuring ODBC connectivity for the Integration Bus:

http://www-01.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.mft.doc/ah14440_.htm

For example, to configure a remote DB2 database, follow this procedure:

1. Install the DB2 client and catalog the DB2 database.

2. Configure the DB2 stanza in the `odbc.ini` configuration file that is shipped with the Integration Bus.
3. Create and test the data source from the Integration Bus, as shown in Example 5-3.

Example 5-3 Configuring and testing the data source in the Integration Bus

```
mqsisetdbparms IB9NODE -n <datasource name> -u <dbuser> -p <dbuser password>
mqsicvp IB9NODE -n <datasource name>
```

Important: For our business scenario, a DB2 database was used. Refer to Appendix C, “Database definitions” on page 239 for information about the database structure and how to set up the database and import the tables that are required to run the integration flows.

5.1.4 Working with integration flows

This section walks you through the integration flows that are implemented for the scenario. If you want to inspect the details and configurations of the integration flows used in these examples, follow Appendix B, “Importing solution applications” on page 237. That Appendix guides you in importing the complete integration flows to your Integration Bus Toolkit workspace and browsing the artifacts.

Note: The integration application used in this book does not cover or provide any validation or error handling procedures.

After importing the projects, you will see the window shown in Figure 5-2.

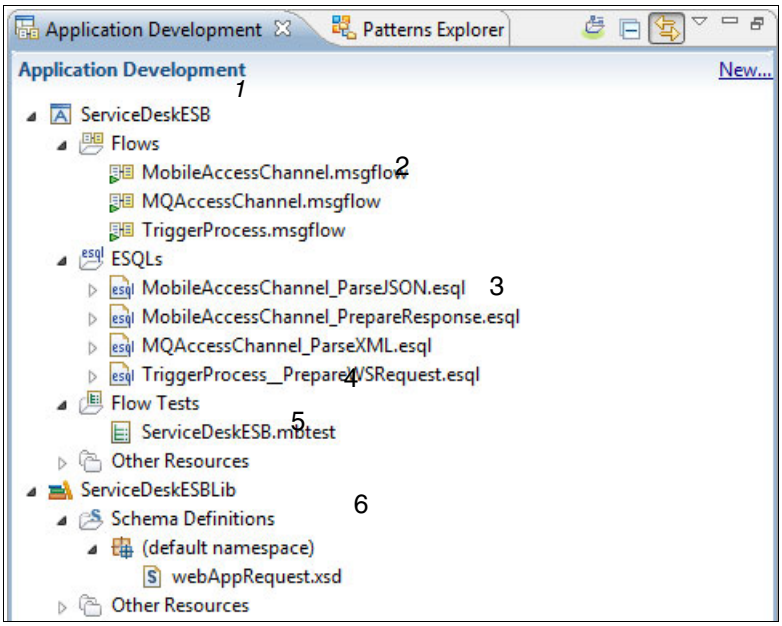


Figure 5-2 Projects, as viewed in the Integration Bus Toolkit

The most important elements of the integration application are shown in Figure 5-2:

1. The main integration application project, which packages the source code and integration logic.
2. Integration flows used to perform the protocol and messaging mediations that integrate the separate business systems.

3. Integration Bus-optimized ESQL source code for implementation of the custom compute nodes of the integration flows.
4. Unit testing flows.
5. An integration library project for additional artifacts.
6. An XSD schema and other files that are accessible from the integration flows.

Note: To keep track of the source code and enforce a common development policy, you can use IBM Rational Collaborative Lifecycle Management Solution tools. Appendix A, “Integrating the Eclipse environment with IBM Rational Collaborative Lifecycle Management” on page 223 shows the procedure for sharing this piece of solution in a Collaborative Lifecycle Management environment.

WebSphere MQ access channel integration flow

Figure 5-3 illustrates the WebSphere MQ access channel provided by `MQAccessChannel.msgflow` integration flow.

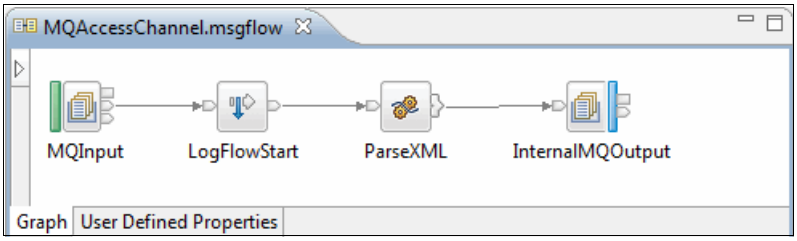


Figure 5-3 WebSphere MQ access integration flow

This integration flow listens on the `WEB_IN` queue for any XML messages. When it receives one, it writes the content of the message to a local file. Then, in the ParseXML computing node, it transforms the XML message that was sent using the web application to a custom, internal XML message format, as shown in Example 5-4, and used later to persist data in the database.

Example 5-4 XML message used in the integration flows

```
<?xml version="1.0" encoding="UTF-8"?>
<DBData>
   <SerialNumber>S0001</SerialNumber>
   <Name>Jonny</Name>
   <Address>Cach</Address>
   <TrackingNumber>WEB3FEFBBF</TrackingNumber>
   <Comments>Maintenance replace</Comments>
</DBData>
```

Note: The message transformation logic is done using compute nodes, which can be implemented using several methods, such as ESQL, Java, .net, or even PHP Hypertext Preprocessor (PHP). In this example, we use the ESQL compute nodes because they are the integral part of the IBM Integration Bus and their execution is the fastest.

To learn more about ESQL, visit the IBM Knowledge Center:

http://www-01.ibm.com/support/knowledgecenter/SSMKHH_9.0.0/com.ibm.etools.mft.doc/ak00990_.htm

The last step of this flow is to send the transformed message to an internal Integration Bus `TRIGGER_PROCESS_Q` queue to trigger other integration flows that will finish the process.

Mobile access channel integration flow

The mobile access channel shown on Figure 5-4 is invoked with an HTTP request, so the HTTP node is used at the beginning to capture it.

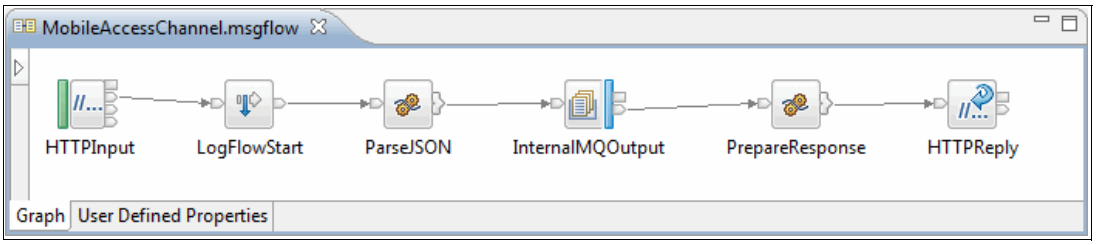


Figure 5-4 Mobile access integration flow

Similar to the WebSphere MQ access channel integration flow, this integration flow also writes the content of the incoming message to a local file. The use of ESQL code on the ParseJSON node strips the data from the JSON message body and generates a tracking number. Then, it puts the message into an internal XML message (see Example 5-4 on page 119) and sends it to an internal queue for further processing. Example 5-5 shows the sample ESQL code for this message transformation.

Example 5-5 Transformation of JSON to XML message using ESQL code

```
CREATE FIELD OutputRoot.XMLNSC.DBData;
      CREATE FIRSTCHILD OF OutputRoot.XMLNSC.DBData NAME 'SerialNumber' VALUE
InputRoot.JSON.Data.MobileAppRequest.SerialNum;
      CREATE FIRSTCHILD OF OutputRoot.XMLNSC.DBData NAME 'Name' VALUE
InputRoot.JSON.Data.MobileAppRequest.Name;
      CREATE FIRSTCHILD OF OutputRoot.XMLNSC.DBData NAME 'Address' VALUE
InputRoot.JSON.Data.MobileAppRequest.Address;
      CREATE FIRSTCHILD OF OutputRoot.XMLNSC.DBData NAME 'Comments' VALUE
InputRoot.JSON.Data.MobileAppRequest.Comments;
      CREATE FIRSTCHILD OF OutputRoot.XMLNSC.DBData NAME 'TrackingNumber' VALUE
Environment.TRACKING_NUMBER;
```

Tip: You can also use a graphical *Mapping* node to perform the message transformation instead of using ESQL. The Mapping node is the easier method when working with large XML files.

Because this is a synchronous call, the Integration Bus has to provide a response to the client. After a successful message submission to an internal queue, the Integration Bus prepares an HTTP response. This response contains a JSON payload, including the tracking number. The request is then sent by the HTTPReply node (see Figure 5-5 on page 121).

Trigger process integration flow

This integration flow shares a common logic for both access channels and is shown in Figure 5-5 on page 121.

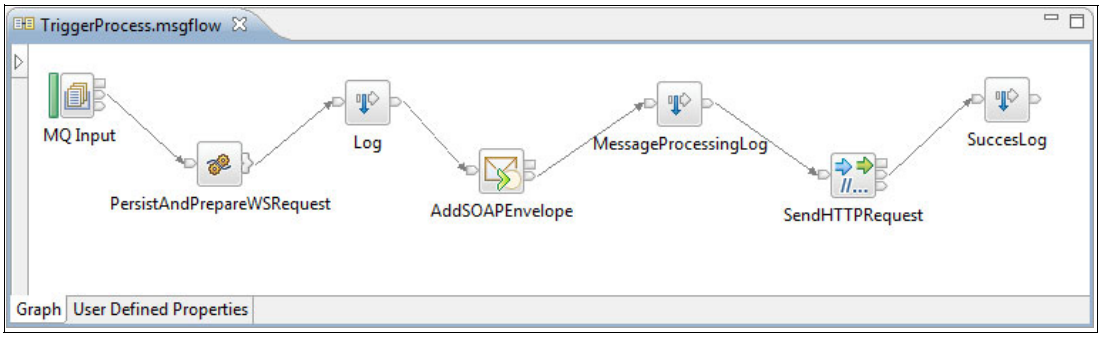


Figure 5-5 Integration flow for triggering the business event

This integration flow reads the incoming XML message and extracts the data to persist it in the database. Example 5-6 shows a sample of the ESQL code that persists the data. Some of the data comes directly from the XML message and some use variables.

Example 5-6 ESQL code used to persist data in the database

```
INSERT INTO Database.SAW_REQUESTS (SERIAL_NUMBER, NAME, TRACKING_NUMBER, ADDRESS,
CREATION_TIME, COMMENTS, STATUS_ID) VALUES (InputRoot.XMLNSC.DBData.SerialNumber,
InputRoot.XMLNSC.DBData.Name, InputRoot.XMLNSC.DBData.TrackingNumber,
InputRoot.XMLNSC.DBData.Address, VAR_CREATION_TIME,
InputRoot.XMLNSC.DBData.Comments, INITIAL_STATUS);
```

In the same PersistAndPrepareWSRequest compute node, the integration flow also creates the body of the SOAP request to call the Operational Decision Manager event rule. If persisting the data is successful, the integration flow writes this information to the custom log, then puts the body of the prepared message request in a generated SOAP envelope. It logs the request in the custom log, and sends it using the HTTP Request node. Because we are not interested in parsing the response of the event rule, the integration flow just adds a line to the custom log if the web service was invoked successfully.

5.1.5 Unit testing

The Integration Bus Toolkit provides a unit testing environment that enables you to test the integration before deploying it to other environments.

To use this utility, follow these steps:

1. Right-click the flow to test, for example the `MobileAccessChannel.msgflow`.
2. Select **Test Message Flow**.
3. In the new window that displays, import or paste your message body. Example 5-7 shows JSON content that can be pasted in the **Message** section.

Example 5-7 Sample JSON body for testing the mobile channel integration flow

```
{"MobileAppRequest": {
   "SerialNum": "S123",
   "Name": "Johny",
   "Address": "West",
   "Comments": "N/A"}}
```

4. Click **Send Message** to run the test.

5. After execution of the test, browse through the recorded steps of the integration flow to inspect the input and output messages. Figure 5-6 shows the input message and the URL of the service that was invoked on the Integration Bus.

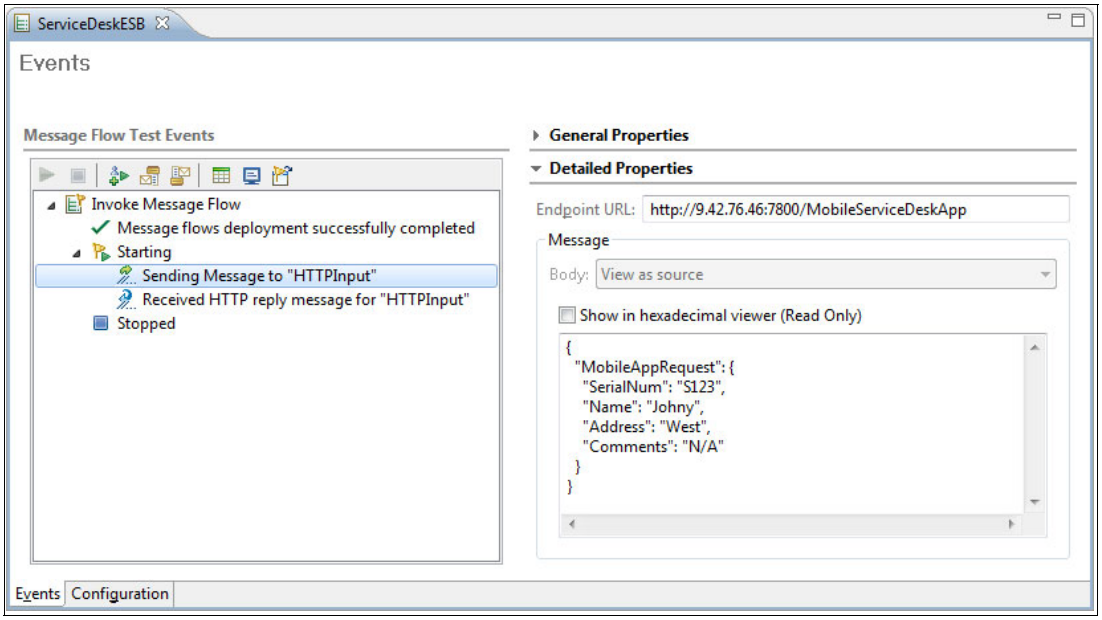


Figure 5-6 Performing unit testing with the Integration Bus: Simulating an incoming JSON message

6. Figure 5-7 shows the result generated by the Integration Bus that is normally sent to the client.

   Optionally, you can save the test case as a file with an `.mbtest` extension in the integration application for later reuse.

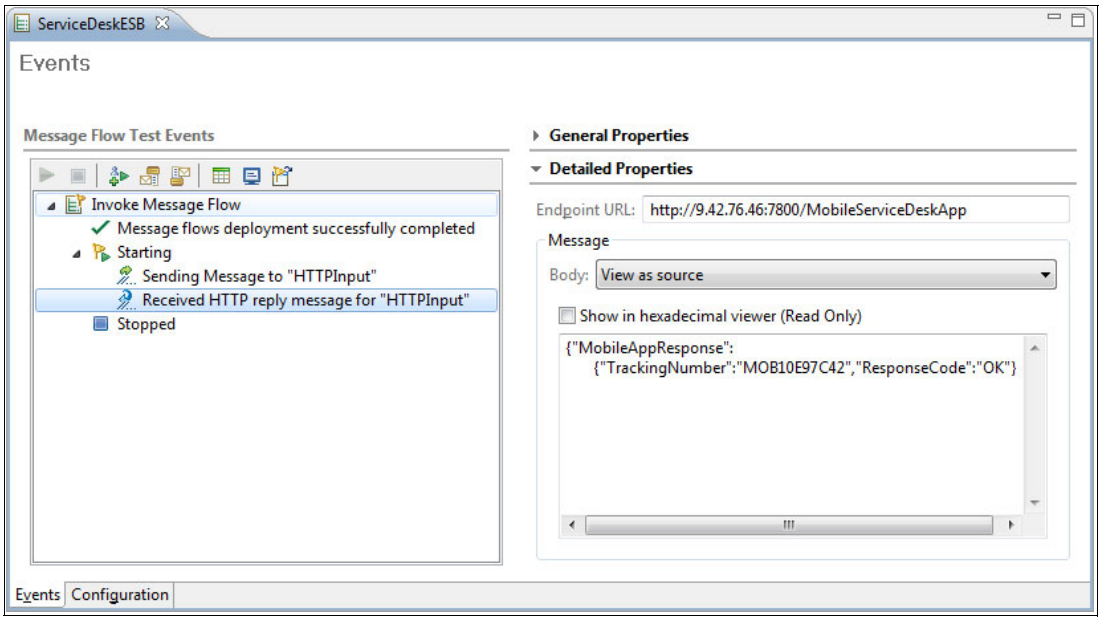


Figure 5-7 Performing unit testing with the Integration Bus: A JSON result generated for the client

5.2 Create the web application for the scenario

In this section, we create a web application that performs one of the front-end access channels for our scenario. We use Eclipse integrated development environment (IDE) for Java Platform, Enterprise Edition (Java EE) developers to create the application. These tools make the development of applications for WebSphere Application Server faster and simpler, and include features, such as hot-deployment.

Note: If you are using the Rational Application Developer IDE family that supports your WebSphere Application Server:

- Skip section 5.2.1, “Install WebSphere Developer Tools in the Eclipse IDE” on page 123, which describes tools that are bundled with WebSphere Developer Tools and offer even more features. To learn more about the Rational development tools platform, see the following site:

  http://www-03.ibm.com/software/products/en/application

- Proceed to 5.2.2, “Develop the web application” on page 123.

5.2.1 Install WebSphere Developer Tools in the Eclipse IDE

The easiest way to install the WebSphere Development Tools for your Eclipse IDE is to follow this procedure:

1. Open your Eclipse IDE and click **Help** → **Eclipse Marketplace**.
2. Search for the term *WebSphere* and click **Go**.

   In the search result list, there are several versions of the IBM WebSphere Application Server Developer Tool plug-in that support different versions of WebSphere Application Server, including Liberty profile.
3. Select the version of the plug-in in your environment.
4. Click **Install**.

   You can install more than one plug-in to support multiple versions of the WebSphere Application Server runtime environment.
5. With the installation or installations complete, restart the Eclipse IDE.

You can also download this plug-in, access documentation, and locate other resources on the *WASdev* web page. This page also contains information about Eclipse and WebSphere Development Tools compatibility:

https://developer.ibm.com/wasdev/downloads

5.2.2 Develop the web application

Follow this section to build and prepare your web application for our scenario.

Create the Java EE project

To create a dynamic web project follow these steps:

1. In the Eclipse IDE, select **File** → **New** → **Dynamic Web Project**.
2. Define the project:
   a. Enter the name for your project, for example, `ServiceDeskWebApp`.

b. Select the runtime for your server, for example the `Liberty` profile.
c. Ensure that the **Add project to an EAR** check box is cleared.

These selections are shown in Figure 5-8.

3. In the same window, click **Finish** and the project will be generated.

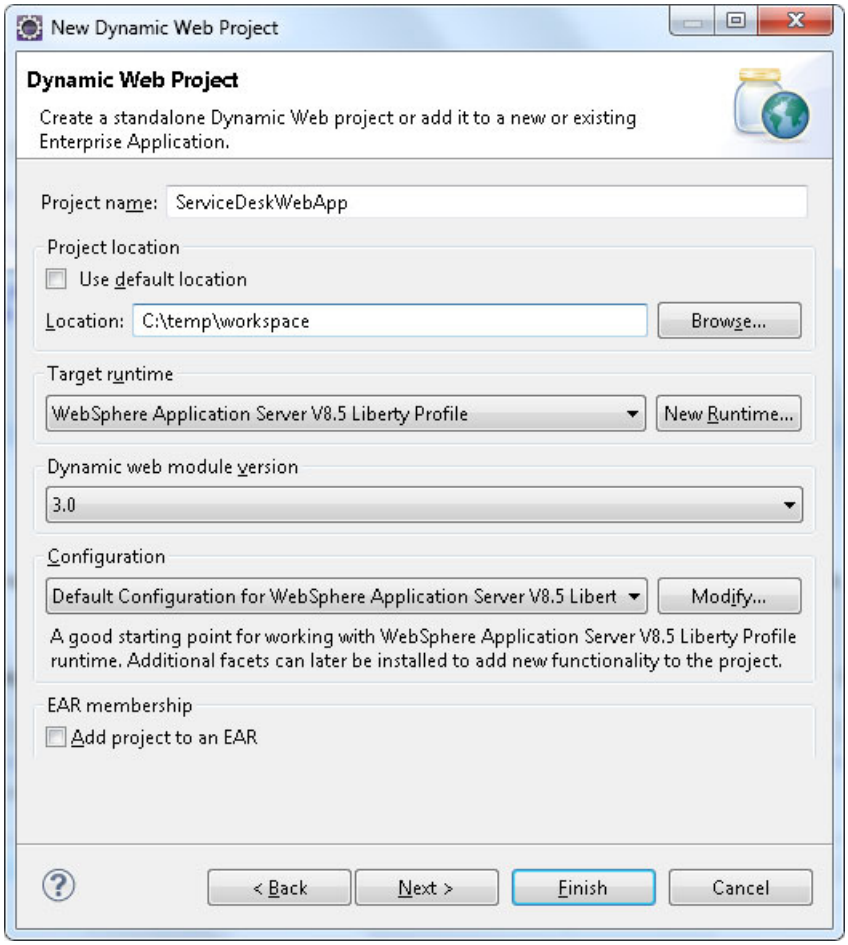


Figure 5-8 Creating the web application project

Note: To keep track of the source code and enforce a common development policy, you can use IBM Rational Collaborative Lifecycle Management Solution tools. Appendix A, “Integrating the Eclipse environment with IBM Rational Collaborative Lifecycle Management” on page 223, shows the procedure for sharing this piece of solution in a Collaborative Lifecycle Management environment.

Create the application logic

With the dynamic web project created, create the controller with the application logic. For our scenario, this role is filled by a servlet with a helper class. Follow this procedure to create both of these artifacts:

1. In the Eclipse **Enterprise Explorer** project view, right-click the newly created project and select **New** → **Servlet.**
2. In the wizard window, in the **Java package** field, enter `com.ibm.redbooks.supportDesk.servlets`.
3. In the **Class name** field, enter `ClaimServlet` and click **Finish**.

4. In the newly created `ClaimServlet` class, replace the code with that in Example 5-8 and save your changes.

Example 5-8 ClaimServlet source code

```
package com.ibm.redbooks.supportDesk.servlets;

import java.io.IOException;
import java.io.PrintWriter;

import javax.annotation.Resource;
import javax.jms.Connection;
import javax.jms.ConnectionFactory;
import javax.jms.DeliveryMode;
import javax.jms.Destination;
import javax.jms.JMSException;
import javax.jms.MessageProducer;
import javax.jms.Session;
import javax.jms.TextMessage;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import com.ibm.redbooks.supportDesk.helpers.ClaimHelper;

/**
 * Servlet implementation class ClaimServlet
 */
@WebServlet("/ClaimServlet")
public class ClaimServlet extends HttpServlet {

   private static final long serialVersionUID = -8909793799230592821L;

   private static final String NAME_PARAM = "name";
   private static final String ADDRESS_PARAM = "address";
   private static final String SERIAL_NUMBER_PARAM = "serialNumber";
   private static final String COMMENTS_PARAM = "comments";

   @Resource(name = "jms/queueCF")
   private ConnectionFactory qcf;

   @Resource(name = "jms/inputQueue")
   private Destination inputQueue;

   public ClaimServlet() {
   }

   protected void doPost(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
      doGet(request, response);
   }

   protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
```

```
      System.out.println("ClaimServlet invoked!");

      PrintWriter out = response.getWriter();
      out.println("<html><head><title>Acknowledgement Of Claim
Form</title></head><body>");

      String name = request.getParameter(NAME_PARAM);
      String address = request.getParameter(ADDRESS_PARAM);
      String serialNumber = request.getParameter(SERIAL_NUMBER_PARAM);
      String comments = request.getParameter(COMMENTS_PARAM);

      out.println("<h2>Sending claim request for item with serial number: " +
serialNumber + " </h2>");

      String trackingNum = ClaimHelper.generateWebTrackingNumber();

      String message = ClaimHelper.generateXML(serialNumber, name, address,
comments, trackingNum);
      System.out.println("Message:\n" + message);

      if (putMsgOnQueue(message))
         out.println("<h2>Your request have been successfully sent to the
Redbooks Support Desk! Your tracking number is " + trackingNum + "</h2>");
      else
         out.println("<h2>Your request was not successfully sent, please
contact with the administrator</h2>");

      out.println("<h3><a href=\"/ServiceDeskWebApp\">Click</a> to return to
the claim form</h3></body></html>");
   }

   /**
    * Puts the message on the MQ queue using JMS
    *
    * @param message
    *            - XML message to be sent to the queue
    * @return true if the sent was succesful
    */
   private boolean putMsgOnQueue(String message) {

      boolean sent = false;
      Connection con = null;
      MessageProducer sender = null;

      try {
         con = qcf.createConnection();
         Session session = con.createSession(false, Session.AUTO_ACKNOWLEDGE);
         sender = session.createProducer(inputQueue);

         TextMessage textMessage = session.createTextMessage();
         textMessage.setText(message);

         sender.send(textMessage, DeliveryMode.PERSISTENT, 1, 0);
         sent = true;
```

```
         System.out.println("Message sent with ID: " +
textMessage.getJMSMessageID().trim());

      } catch (Exception e) {
         e.printStackTrace();
      } finally {

         try {
            sender.close();
         } catch (JMSException e) {
            e.printStackTrace();
         }

         try {
            con.close();
         } catch (JMSException e) {
            e.printStackTrace();
         }
      }

      return sent;
   }
}
```

With the servlet ready, create the helper class:

1. In the Eclipse **Enterprise Explorer** project view, right-click the project and select **New** → **Class**.
2. In the wizard window, in **Java package** field, enter:

   `com.ibm.redbooks.supportDesk.helpers`
3. In the **Class name** field, enter `ClaimHelper` and click **Finish**.
4. In the newly created `ClaimHelper` class, replace the code with that in Example 5-9 and save your changes.

Example 5-9 ClaimHelper source code

```
package com.ibm.redbooks.supportDesk.helpers;

import java.io.StringWriter;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerException;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;

import org.w3c.dom.Attr;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
```

```
public class ClaimHelper {

   private static final String WEB_PREFIX = "WEB";

   public static String generateXML(String serialNumber, String name,
         String address, String comments, String trackingNum) {

      String xml = null;
      StringWriter writer = null;

      try {
         DocumentBuilderFactory docFactory = DocumentBuilderFactory
               .newInstance();
         DocumentBuilder docBuilder = docFactory.newDocumentBuilder();

         // root elements
         Document doc = docBuilder.newDocument();
         doc.setXmlStandalone(true);
         Element rootElement = doc.createElement("WebAppRequest");

         Attr attr1 = doc.createAttribute("xmlns:xsi");
         attr1.setValue("http://www.w3.org/2001/XMLSchema-instance");
         Attr attr2 = doc.createAttribute("xsi:noNamespaceSchemaLocation");
         attr2.setValue("webAppRequest.xsd");

         rootElement.setAttributeNode(attr1);
         rootElement.setAttributeNode(attr2);

         doc.appendChild(rootElement);

         Element serialNum = doc.createElement("SerialNum");
         serialNum.appendChild(doc.createTextNode(serialNumber));
         rootElement.appendChild(serialNum);

         Element nameEl = doc.createElement("Name");
         nameEl.appendChild(doc.createTextNode(name));
         rootElement.appendChild(nameEl);

         Element trackingEl = doc.createElement("TrackingNum");
         trackingEl.appendChild(doc.createTextNode(trackingNum));
         rootElement.appendChild(trackingEl);

         Element addressEl = doc.createElement("Address");
         addressEl.appendChild(doc.createTextNode(address));
         rootElement.appendChild(addressEl);

         Element commentsEl = doc.createElement("Comments");
         commentsEl.appendChild(doc.createTextNode(comments));
         rootElement.appendChild(commentsEl);

         DOMSource domSource = new DOMSource(doc);
         writer = new StringWriter();
         StreamResult result = new StreamResult(writer);
         TransformerFactory transformerFactory = TransformerFactory
               .newInstance();
```

```
            Transformer transformer = transformerFactory.newTransformer();

            transformer.transform(domSource, result);
            xml = writer.toString();

         } catch (ParserConfigurationException pce) {
            pce.printStackTrace();

         } catch (TransformerException tfe) {
            tfe.printStackTrace();
         }
         return xml;
      }

      public static String generateWebTrackingNumber() {

         return WEB_PREFIX
               + Long.toHexString(Double.doubleToLongBits(Math.random()))
                     .toUpperCase().subSequence(0, 7);
      }

   }
```

The code for the servlet in Example 5-8 on page 125 does the following:

- Generates a unique tracking number using the `ClaimHelper` class.
- Generates an XML message that contains the user input and the tracking number.
- Sends a message to the WebSphere MQ queue manager using the `putMsgOnQueue` method.
- Generates HTML output for the user.

The important parts of code are:

`@WebServlet("/ClaimServlet")`	Annotation that configures the servlet mapping URL.
`@Resource(name = "jms/queueCF")`	This annotation is a reference that is used to obtain the connection to the WebSphere MQ queue manager.
`@Resource(name = "jms/inputQueue")`	This annotation is a reference that is used to define the destination queue.

The code of the helper class shown in Example 5-9 on page 127 implements two static methods:

`generateWebTrackingNumber()`	This is used to generate a random string that represents the tracking number.
`generateXML()`	`This` uses a DOM parser to generate an XML message.

Create a claim form

To create the front-end page with a claim form, follow these steps:

1. Right-click the project and select **New** → **JSP File**.
2. In the wizard window, in the **File Name** field, enter `index.jsp` and click **Finish**.

3. In the newly created JavaServer Pages (JSP) file, replace the code with that in Example 5-10 and save your changes.

Example 5-10 Claim form source code

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-1"
   pageEncoding="ISO-8859-1"%>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<title>Redbooks Support Desk System</title>
</head>
<body>
   <h2>Claiming Form</h2>
   <form method="POST" action="/ServiceDeskWebApp/ClaimServlet">
      <table>
         <tbody>
            <tr>
               <td>Name:</td>
               <td><input type="text" name="name" size="20"></td>
            </tr>
            <tr>
               <td>Address:</td>
               <td><input type="text" name="address" size="30"></td>
            </tr>
            <tr>
               <td>Serial Number Of The Item:</td>
               <td><input type="text" name="serialNumber" size="20"></td>
            </tr>
            <tr>
               <td>Additional comments:</td>
               <td><textarea name="comments" cols="20" rows="8"></textarea></td>
            </tr>
            <tr>
               <td></td>
               <td><input type="submit" value="Submit" name="SUBMIT"></td>
            </tr>
         </tbody>
      </table>
   </form>

   <h6>Powered by IBM Redbooks Team</h6>
</body>
</html>
```

The role of the form presented in Example 5-10 is simple. It gathers the information from the user and sends an HTTP POST request to ClaimServlet.

Note: Neither the servlet or the JSP implement any security or validation functionality.

Browse through application resources

Figure 5-9 on page 131 shows the view of the created application in the Eclipse Java EE perspective.

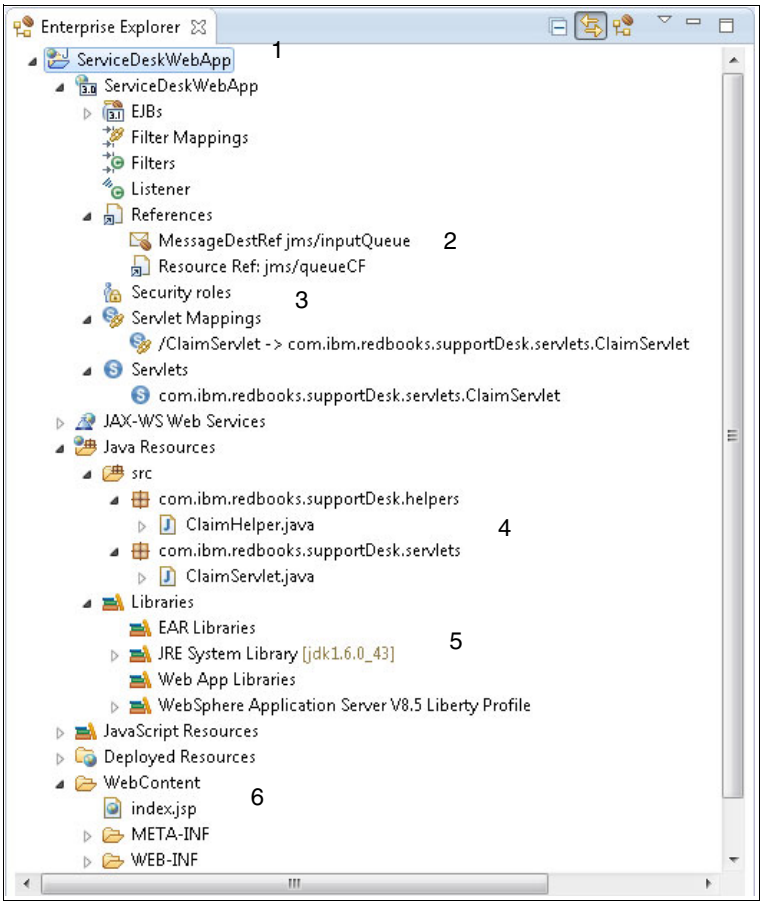


Figure 5-9 Web application Java EE projects view

The most important elements of the application are highlighted in Figure 5-9 and defined as follows:

1. The main project packaging of all application code and resources.
2. Resource references used by the application for the connection factory and the queue.
3. Servlet mapping used to invoke the application controller logic.
4. The source code of the servlet and a helper class with static methods.
5. Libraries and runtimes used by the application in the Eclipse workspace.
6. The main JSP page with the claim form.

Figure 5-10 on page 132 presents the generated HTML output from the application in the browser.

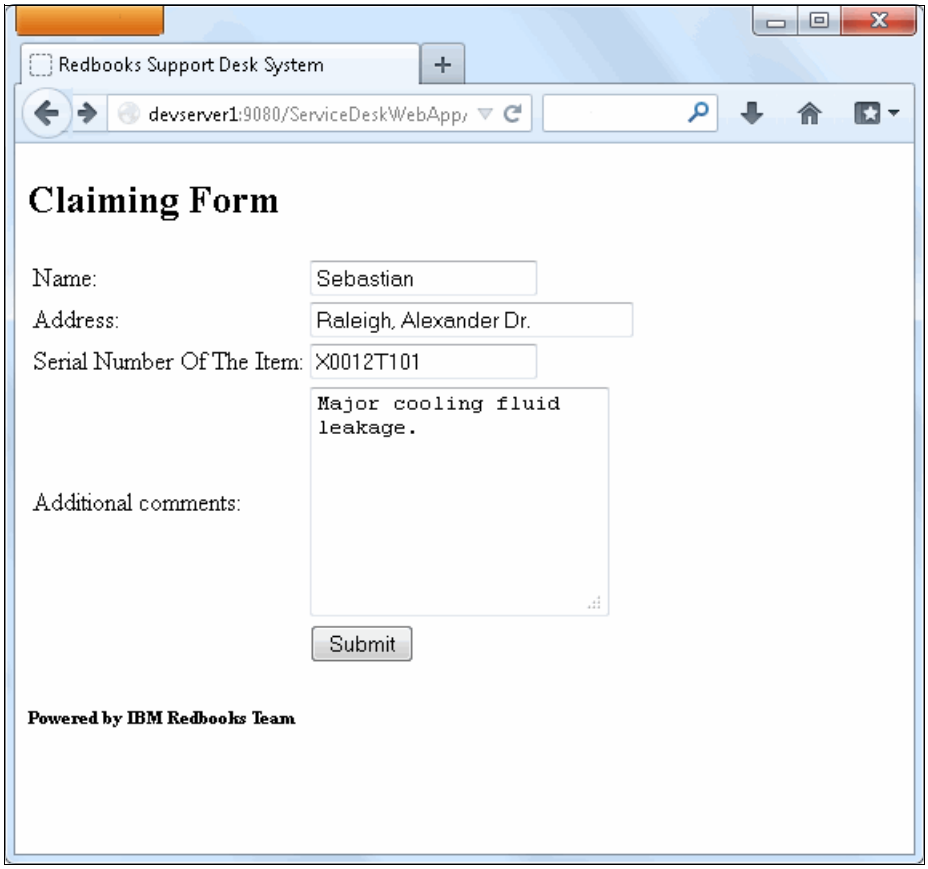


Figure 5-10 Claim form presented by the web application

5.2.3 Export the application archive

To export the application for deployment, follow these steps:

1. Right-click the project and select **Export** → **WAR**.
2. In the wizard, click **Browse** and select the output location.
3. Ensure that the **Optimize for a specific server runtime** check box is cleared and click **Finish**.

The Application is saved to your chosen location in Java `.war` format.

5.3 Prepare the Liberty profile environment for your web application

The WebSphere Liberty profile was introduced with WebSphere Application Server version 8.5 and provides dynamic and simplified stand-alone runtime for web applications. It supports a subset of the programming model and runtime features that are available with WebSphere Application Server full profile. Any application that runs on the Liberty profile will also run on the full profile.

Note: For more information about the Liberty profile features and capabilities, refer to: *WebSphere Application Server V8.5 Administration and Configuration Guide for Liberty Profile*, SG24-8170.

To prepare your Liberty profile environment, follow these steps:

1. Configure the Java Runtime environment.
2. Install the Liberty profile runtime environment.
3. Create the server, and (optionally) configure features and add-ons for your application.

Obtain the latest Liberty profile package from the WASdev website:

https://developer.ibm.com/wasdev/downloads

5.3.1 Configure the Java runtime environment

The Liberty profile is not bundled with a Java Runtime Environment, but it needs one to run. It searches for the **`java`** command in the following order of properties: `JAVA_HOME`, `JRE_HOME`, and `PATH`.

Example 5-11 shows the commands for setting these properties on a Linux system.

Example 5-11 Setting the Java runtime environment on Linux

```
export JAVA_HOME=/opt/IBM/Java6
export PATH=$JAVA_HOME/bin:$PATH
```

Important: On distributed platforms, the minimum supported level for the IBM Java Development Kit (JDK) is 6.0 (J9 2.6) SR1 and Java 6 update 26 for Oracle.

5.3.2 Installation

To install the Liberty profile environment in text mode, follow these steps. The Liberty profile environment is one of the fastest ways to start working with the server:

1. Run the command to extract the contents of the Liberty archive, for example:

   ```
   java -jar wlp-developers-runtime-8.5.5.3.jar
   ```

2. Accept the license agreement.
3. Press **1** to agree to the license terms and proceed.
4. Provide the installation path for the Liberty profile, for example `/opt/IBM/WebSphere`, and press **Enter**.

Note: The Liberty profile can be also installed using the IBM Installation Manager in either graphical and text mode. For more information, see the following site:

http://www-01.ibm.com/support/knowledgecenter/SSAW57_8.5.5/com.ibm.websphere.wlp.nd.doc/ae/twlp_inst_top.html?cp=SSAW57_8.5.5%2F1-3-11-0-1

5.3.3 Create and configure the server

To create a new Liberty profile server, run the **server** command from the `bin` directory of the Liberty profile runtime environment. For example, on a Linux or UNIX system, run following command:

```
./server.sh create server1
```

This command creates the server1 instance in the `/usr/servers` subdirectory, along with the `server.xml` configuration file, which we will use to configure the Liberty profile server.

For our scenario, the web application requires that the runtime server provide additional Java Message Service (JMS) and Java Naming and Directory Interface (JNDI) resources on the server. This is because the application connects to WebSphere MQ. The Liberty profile server also has to be configured to use an external JMS messaging provider. For that purpose, we use the WebSphere MQ resource adapter. You can read about this adapter and download it from following web page:

http://www-01.ibm.com/support/docview.wss?uid=swg21633761

After downloading and extracting the WebSphere MQ resource adapter, you need to refer to its installation path in the Liberty profile configuration file, using `variable` property named `wmqJmsClient.rar.location`:

```
<variable name="wmqJmsClient.rar.location" value="YOUR_PATH/wmq.jmsra.rar" />
```

Example 5-12 shows a completed configuration file for the Liberty profile server to run the application for the scenario.

Example 5-12 Complete Liberty profile configuration to run the application

```
<server description="new server">

    <!-- Enable features -->
    <featureManager>
        <feature>jsp-2.2</feature>
        <feature>wmqJmsClient-1.1</feature>
        <feature>jndi-1.0</feature>
    </featureManager>

    <httpEndpoint id="defaultHttpEndpoint"
                  host="*"
                  httpPort="9080"
                  httpsPort="9443" />

    <variable name="wmqJmsClient.rar.location"
value="/opt/ibm/liberty/wmqadapter/wmq/wmq.jmsra.rar" />

    <connectionManager id="ConMgr" maxPoolSize="2" />

    <jmsConnectionFactory jndiName="jms/queueCF" connectionManagerRef="ConMgr">
                <properties.wmqJms
                        transportType="CLIENT"
                        hostName="iib.raleigh.ibm.com"
                        port="1414"
                        channel="WAS.CONN"
                        queueManager="IB9QMGR" />
    </jmsConnectionFactory>
```

```
    <jmsQueue id="jms/inputQueue" jndiName="jms/inputQueue">
                <properties.wmqJms
                        baseQueueName="WEB_IN"
                        baseQueueManagerName="IB9QMGR" />
    </jmsQueue>

</server>
```

As you can see from Example 5-12 on page 134, the Liberty profile uses the `jsp-2.2` feature to run the servlet that processes the XML message. The `Jndi-1.0` and `wmqJmsClient-1.1` features are used for messaging and connectivity with the WebSphere MQ. The `jmsConnectionFactory` and `jmsQueue` properties are set to point to the correct resources and destination on the WebSphere MQ queue manager.

5.3.4 Deploy the application manually

To deploy manually, start the server and deploy the application as follows. Use the following command to start the Liberty profile server1 runtime environment:

```
server start server1
```

To install the application, use the archive that was generated in 5.2.3, “Export the application archive” on page 132.

Copy the archive to the `dropins` folder, located in the `server1` directory structure. Alternatively, add the following property to the server.xml configuration file, indicating the location of your application:

```
<webApplication id="ServiceDeskWebApp"
location="${server.config.dir}/apps/ServiceDeskWebApp.war"
name="ServiceDeskWebApp"/>
```

Note: This chapter covers only the basic Liberty profile configuration capabilities. To learn more about using the Liberty profile, see:

WebSphere Application Server V8.5 Administration and Configuration Guide for Liberty Profile, SG24-8170.

Also see the IBM Knowledge Center at the following location:

http://www-01.ibm.com/support/knowledgecenter/SSEQTP_8.5.5/com.ibm.websphere.wlp.doc/ae/cwlp_about.html?lang=en

Now you can run the application and see the user claim form shown in Figure 5-10 on page 132. Enter the following URL in a browser (port 9080 is a default HTTP inbound port for Liberty profile, but it can be dynamically changed by updating the `httpEndpoint` property in the `server.xml` configuration file):

http://<hostname>:9080/ServiceDeskWebApp

5.4 Prepare WebSphere Application Server full profile environment for web application

To prepare the WebSphere Application Server full profile environment for the web application prepared in 5.2, “Create the web application for the scenario” on page 123, follow this procedure:

1. Install the WebSphere Application Server runtime environment.
2. Create a profile.
3. Start and configure the server.
4. Deploy the application.

5.4.1 Install and create the server profile

The WebSphere Application Server can be installed in various configurations, including Base (stand-alone) and network deployment topologies. For our scenario, we use a stand-alone configuration of WebSphere Application Server profile, version 8.5.5.1.

Detailed process steps for installing the WebSphere Application Server and for creating an application server profile are included in *WebSphere Application Server V8.5 Administration and Configuration Guide for the Full Profile*, SG24-8056.

5.4.2 Configure the server

With your WebSphere Application Server installed and stand-alone profile ready, start the server by running the following command from the installed profile `bin` directory:

```
./serverStart.sh server1
```

To display the administrative console, go to the following URL (port 9060 is a default administrative port, but it can be changed during or after profile creation):

```
http://<hostname>:9060/ibm/console
```

> **Note:** If you installed a WebSphere Network Deployment Cell environment, connect to the deployment manager server host to access the console.
>
> If you did not enable security during profile creation, click **Log in** on the Welcome page without providing any credentials. If security is enabled, enter the administrative user name and password, then click **Log in**.

To prepare the environment for the application that was prepared in 5.2, “Create the web application for the scenario” on page 123, create and configure the messaging providers and destination queue in the server runtime environment:

1. In the administrative console, click **Resources** → **JMS** → **Connection factories**.
2. Select the scope for the connection factory. If you are using network deployment topology to access resources, this setting provides access to other servers. For this scenario, select the cell scope and click **New**.
3. Select WebSphere MQ messaging provider and click **OK**.
4. Enter `queueCF` in the **Name** field, `jms/queueCF` in the **JNDI name** field, and click **Next**.
5. Use the default option to **Enter all required information into this wizard** and click **Next**.

6. Enter the name of your queue manager in the **Queue manager or queue sharing group name** field. In our scenario, we use `IB9QMGR` for the queue manager name.
7. Click **Next**.
8. Select `Client` from the **Transport** drop-down menu, then supply the **Hostname**, **Port**, and **Server connection channel** with the parameters of your queue manager. An example is shown in Figure 5-11.

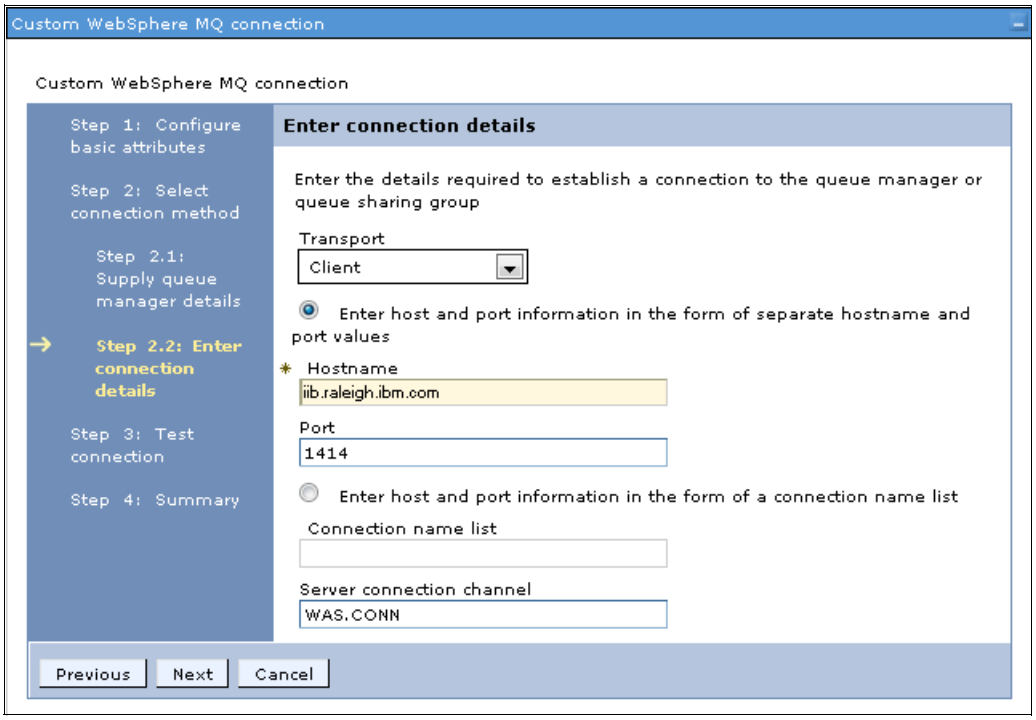


Figure 5-11 Configuring the connection factory for the WebSphere MQ messaging provider

9. Click **Next**.
10. Click **Test connection**. If the test is successful, click **Next**.
11. In the summary window, confirm that the parameters you provided match those of your WebSphere MQ queue manager.
12. Click **Finish**.

With the connection factory defined, create the destination definition:

1. In the administrative console, click **Resources** → **JMS** → **Queues**.
2. Select the same scope that you chose for the connection factory in Step 2 on page 136, and click **New**.
3. Select **WebSphere MQ messaging provider** and click **OK**.
4. Enter `inputQueue` in the **Name** field, `jms/inputQueue` in the **JNDI name** field, `WEB_IN` in the **Queue name** field, and click **OK**.

To commit your changes to the WebSphere Application Server runtime environment, click **Save** in the message box that displays in the top of the window, as shown in Figure 5-12 on page 138.

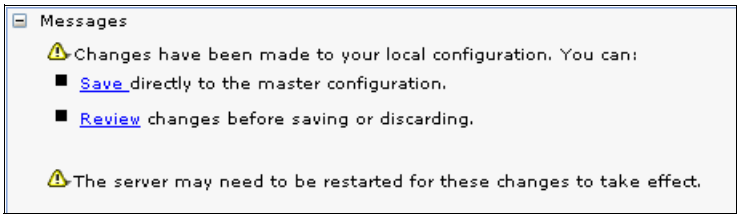


Figure 5-12 Saving any configuration change on WebSphere Application Server full profile

This is especially important when working with network deployment environments because changes made in the deployment manager have to be synchronized across the whole cell where multiple servers can be defined.

> **Tip:** You can manually enforce synchronization by clicking **System administration** → **Nodes**, then selecting the nodes and clicking the **Synchronize** button.

5.4.3 Deploy the application manually

With the configuration created (see 5.4.2, “Configure the server” on page 136), the WebSphere Application Server runtime environment is ready to host the application for our scenario. Follow these steps to deploy the application that was created in 5.2.3, “Export the application archive” on page 132:

1. In the administrative console, click **Applications** → **New Application**.
2. Click **New Enterprise Application**.
3. Click **Browse** and locate the WAR file for the application (ServiceDeskWebApp.war) and click **Next**.
4. Select the **Fast Path** option and click **Next**.
5. In the Install New Application wizard, map the JNDI resources to application references. Use the **Browse** button to look up the resources you defined in “Create the application logic” on page 124, or type their names manually. In the application in our scenario, we used the same names for references and resources.
6. Click the **Summary step** link (or click Next until you reach the last step of the wizard), and click **Finish**.
7. Wait for changes to synchronize on the WebSphere Application Server, and click **Save** to commit the changes to the runtime environment.

The default behavior of the WebSphere Application Server full profile is that a new application is in stop state after installation. To start it, follow these steps:

1. In the administrative console, click **Applications** → **Application Types** → **WebSphere enterprise applications**.
2. Select the `ServiceDeskWebApp` application and click **Start**.

> **Tip:** After deploying the application, you can still access and modify the resource references. To do so, click your application, and select the **Resource references** link in the **References** section.

Now you can run the application and see the user claim form shown in Figure 5-10 on page 132. Enter the following URL in the browser (port 9080 is a default HTTP inbound port

for the WebSphere Application Server full profile, but it can be changed during and after profile creation):

http://<hostname>:9080/ServiceDeskWebApp

Figure 5-10 on page 132 shows the application form that should be displayed.

5.5 Create the mobile application for our scenario

This section describes the development of the mobile application using Worklight Studio.

In our scenario, the mobile application implements a basic functionality by enabling the user to create a claim and retrieve a tracking number.

To communicate with the back-end system, the application uses a Worklight HTTP adapter, which calls a Representational State Transfer (REST) service implemented in the Integration Bus.

Worklight Studio enables the creation of mobile applications for multiple platforms, such as Android, iOS, and Windows phones, among others. For a complete list of supported platforms and corresponding system requirements, see the following site:

http://www-01.ibm.com/support/knowledgecenter/SSZH4A_6.2.0

In our scenario, we use Worklight Studio to create both a mobile website and an Android application.

5.5.1 Prepare the development environment

For developing the mobile application for our scenario, we need to install Worklight Studio and the Android software development kit (SDK).

Install Worklight Studio

Worklight Studio is an Eclipse installation, which is enhanced with specific plug-ins.

At the time of this writing, the supported versions of Eclipse are Juno SR2 (4.2.2), Kepler SR1 (4.3.1), and Luna (4.4). Any other version of Eclipse is not officially supported and can present unsolved dependencies that might compromise the development process.

For our scenario, we use Eclipse Juno SR2 (4.2.2) for easy integration with Rational Team Concert plug-ins.

If you are planning to use Rational Team Concert, refer to the plug-in installation at “Install the Rational Team Concert Eclipse plug-in” on page 225. Also, an additional process is required to enable the plug-in coexistence. Refer to the product documentation at the following link:

http://www-01.ibm.com/support/knowledgecenter/SSZH4A_6.2.0/com.ibm.worklight.installconfig.doc/devenv/t_extrastepsforrtcv4.0.html

To install Worklight Studio, follow these steps:

1. Download Eclipse Juno SR2 (4.2.2) from the Eclipse Foundation site, choosing the Eclipse IDE for Java EE Developers version:

   https://www.eclipse.org/downloads/packages/release/juno/sr2

2. Extract the downloaded file.

3. Open Eclipse by clicking the executable file in the installation folder.
4. Choose the workspace folder. You can leave the default location as-is, or change it.
5. From the Eclipse menu, select **Help** → **Eclipse Marketplace**.
6. In the Find box, enter **Worklight** and click **Go**.
7. In the search results, select IBM Worklight Studio.
8. Click **Install**.

Install the Android SDK

The Android SDK is needed to generate Android applications. Download it at:

`http://developer.android.com`

At the time of writing this book, three installation options are available:

- *Download Eclipse ADT:* This option downloads the full development environment, including a customized version of Eclipse, the Android SD, and tools.
- *Download Android Studio:* An integrated development environment, currently in Beta.
- *Download the stand-alone Android SDK Tools:* This downloads the SDK for an existing IDE. This is the *preferred choice* for adding Android application development capabilities to an existing Worklight Studio installation.

After installing the Android SDK, at least one Android platform environment needs to be downloaded. This is achieved by using the Android Manager and selecting the platform version to download. In our scenario, we used platform version 19, but other versions can be selected for developing applications for specific platform versions.

Install the Android development tool (ADT) plug-in for Eclipse

To install the ADT plug-in for Eclipse, follow these steps:

1. From the Eclipse menu, select **Help** → **Eclipse Marketplace**.
2. In the **Find** text box, enter `ADT` and click **Go**.
3. Select **Android Development Tools for Eclipse** and click **Install**.
4. Restart the Eclipse IDE after the installation is complete.

5.5.2 Develop the mobile application

Developing the mobile application consists of the following steps:

- See “Create the application” on page 140.
- See “Create the UI” on page 144.
- See “Create the HTTP adapter” on page 148.
- See “Call the adapter from the UI” on page 155.

Create the application

To create the UI for the application, follow these steps:

1. Open Worklight Studio.
2. When prompted, enter a folder name to be used as the workspace location.

3. Before starting the development process, it is recommended to switch to the Design perspective. Even if it is not strictly necessary to use this perspective, it will ease the Worklight development process.

   To change to the Design perspective, select **Window** → **Open Perspective** → **Other** and select **Design** from the list.

4. To create a new project, select **File** → **New** → **Worklight Application Project.** You can start defining the Worklight project properties as shown in Figure 5-13.

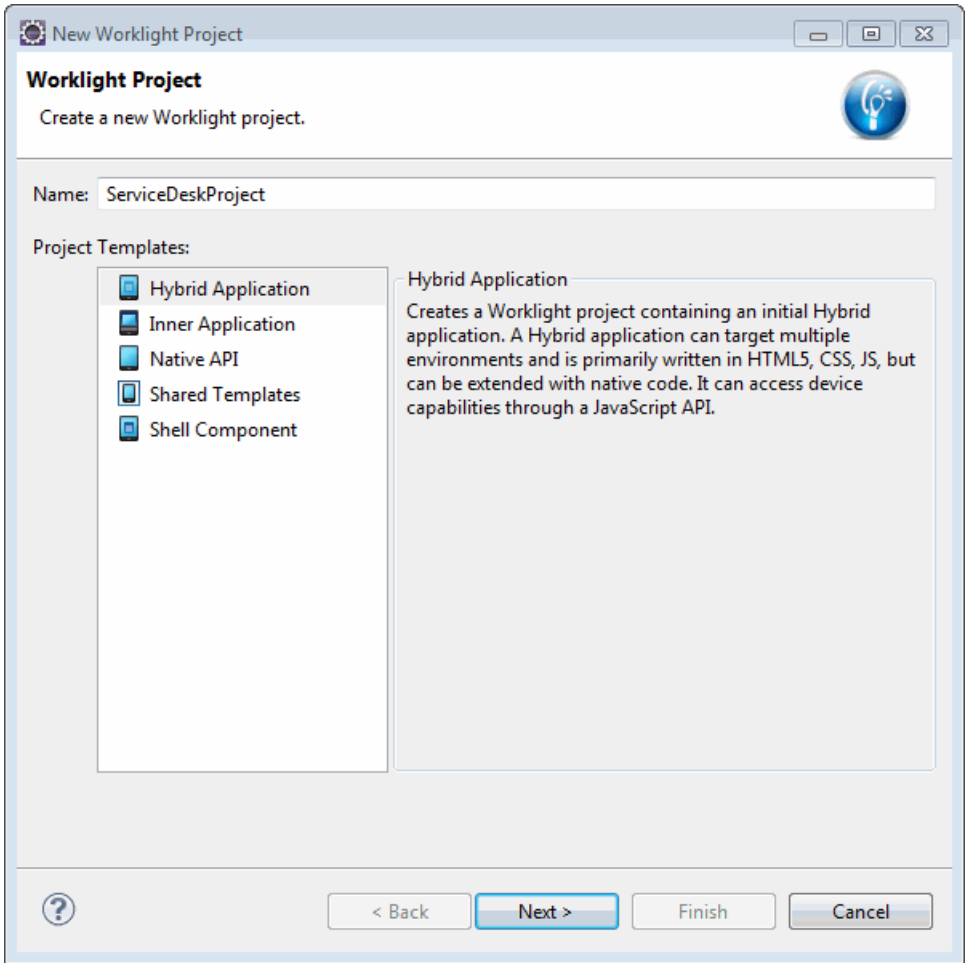


Figure 5-13 Defining properties for a new Worklight project

5. Enter a name for the project, select the **Hybrid Application** type, and click **Next**.

   When choosing the project name, keep in mind that a single project can contain many applications. For our scenario, we named the project `ServiceDeskProject`.

6. Enter the application name, as shown in Figure 5-14 on page 142.

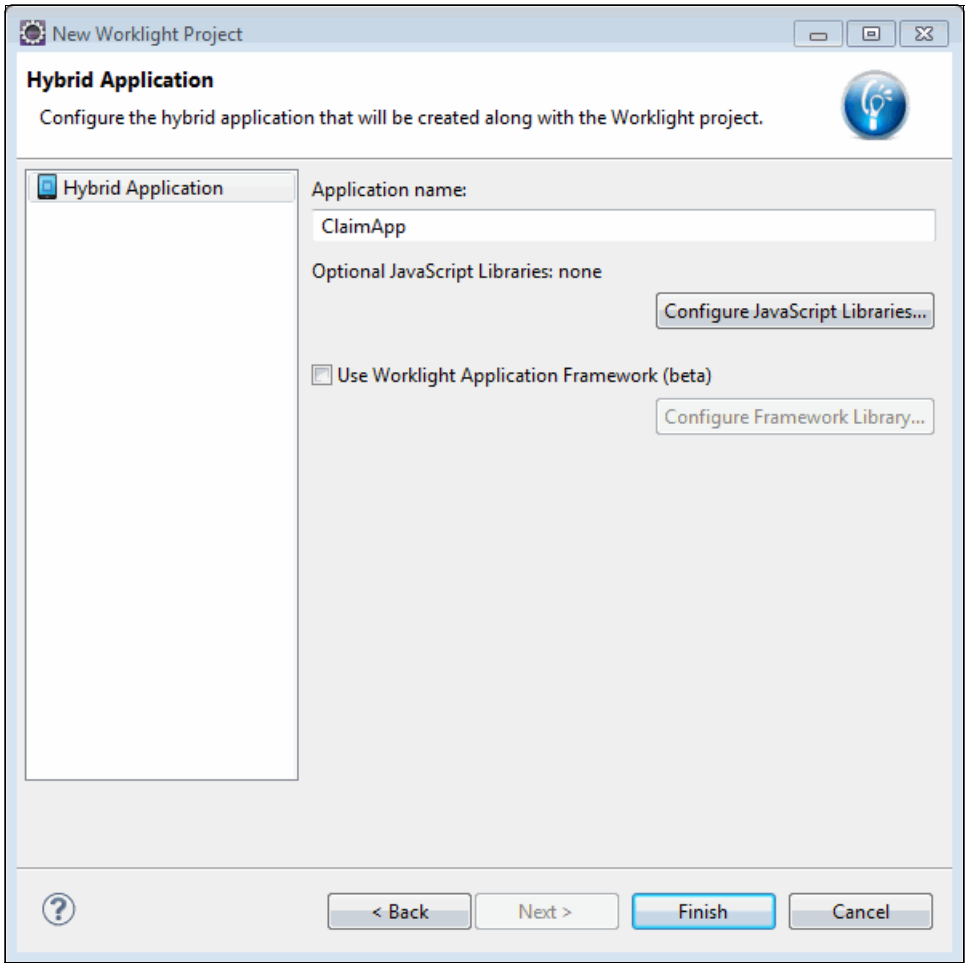


Figure 5-14 Defining the applications for a new project

7. Configure the necessary JavaScript libraries by clicking **Configure JavaScript Libraries**, as shown in Figure 5-15 on page 143.

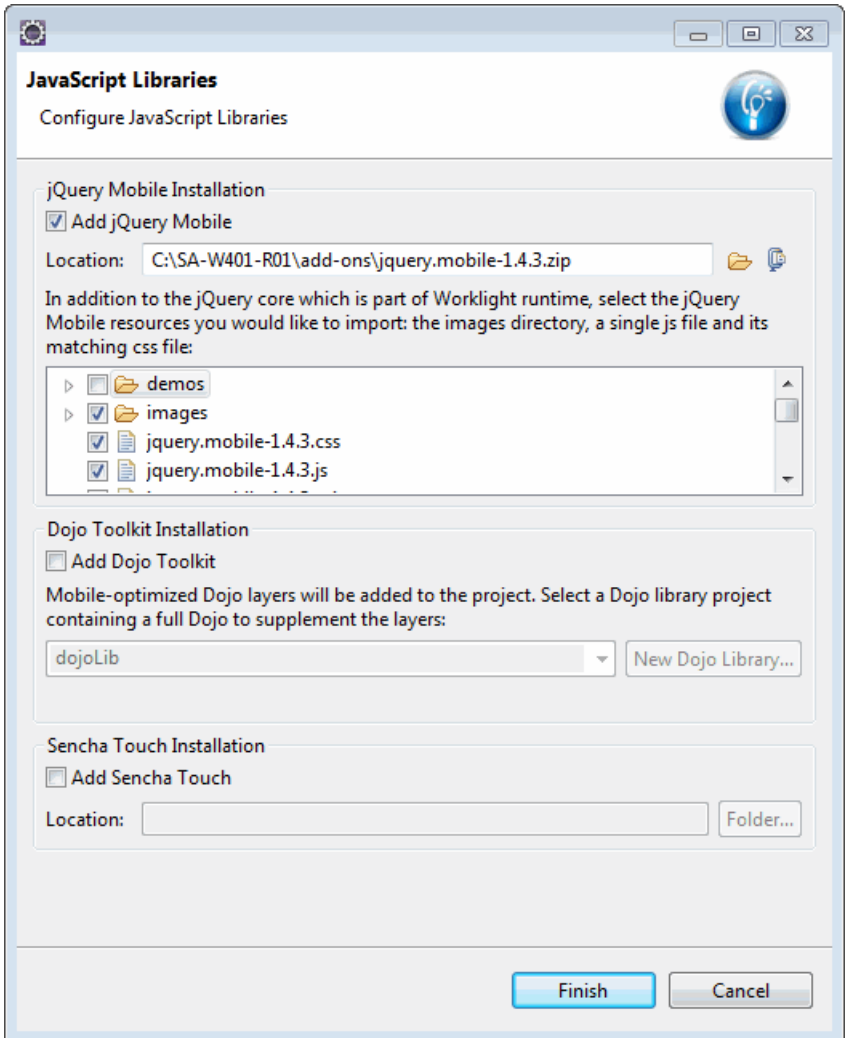


Figure 5-15 Configuring JavaScript Libraries

8. For our application, we used jQuery Mobile as the JavaScript framework. Check the **Add jQuery Mobile** option to enable the additional configuration.
9. To use jQuery Mobile in the application, point to a local copy of the framework. To do that:
   a. Download a stable version of the framework from `http://jquerymobile.com`.
   b. Save the framework as a compressed file in a local folder. For our scenario, and at the time of this writing, we downloaded version 1.4.3, which was marked as the latest stable version.
10. In the **Location** text box, enter or browse for the path to the downloaded compressed file.
11. The files that are contained in the compressed file are available for selection. For our sample application, it is enough to select the **images** folder and the main **.js** and **.css** files for the corresponding jQuery Mobile version. So, for our scenario, we selected **jquery.mobile-1.4.3.css** and **jquery.mobile-1.4.3.js**.

> **Note:** Remember that the file names might change with different releases of the framework. If the required files are not included, some jQuery Mobile features might not work properly in the application.

12. Click **Finish** to close the JavaScript configuration windows.
13. Click **Finish** in the application creation window. After some processing, the default content for the project and application is created, as shown in Figure 5-16.

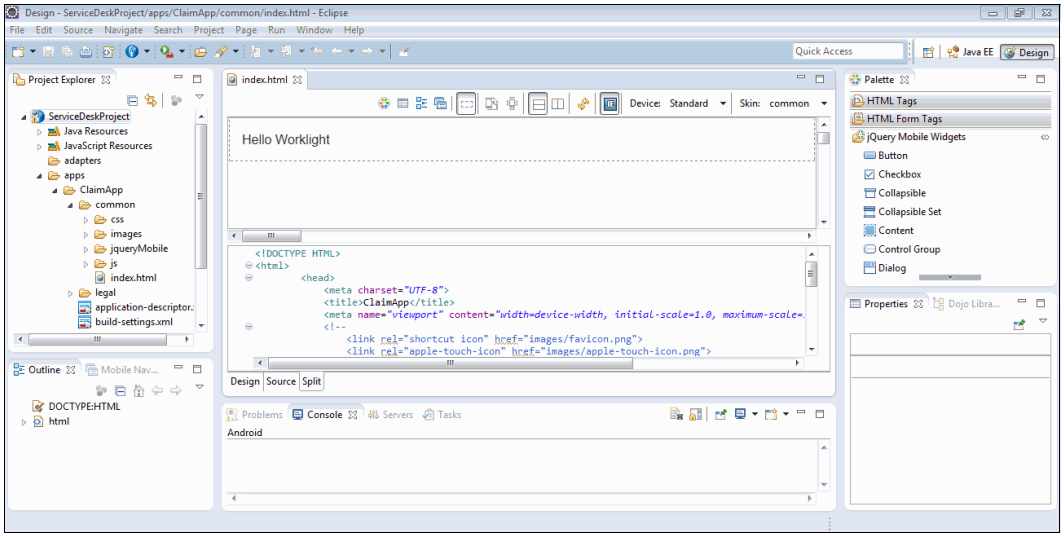


Figure 5-16 Default workspace view for a new project

Note: To keep track of the source code and enforce a common development policy, you can use IBM Rational Collaborative Lifecycle Management Solution tools. Appendix A, “Integrating the Eclipse environment with IBM Rational Collaborative Lifecycle Management” on page 223, shows the procedure for sharing this project in a Collaborative Lifecycle Management environment.

Create the UI

To create the UI, we use the main application HTML file (`index.html`) and create two content sections: one for sending claim information and one for displaying the associated tracking number. Each section displays dynamically, depending on the application logic.

Set up the claim form and tracking number windows

To set up and define the claim form and tracking number windows, follow these steps:

1. Edit the `index.html` file located in the `ServiceDeskProject\apps\ClaimApp\common` folder.
2. Locate the `<body>` area of the source code, as shown in Figure 5-17 on page 145.

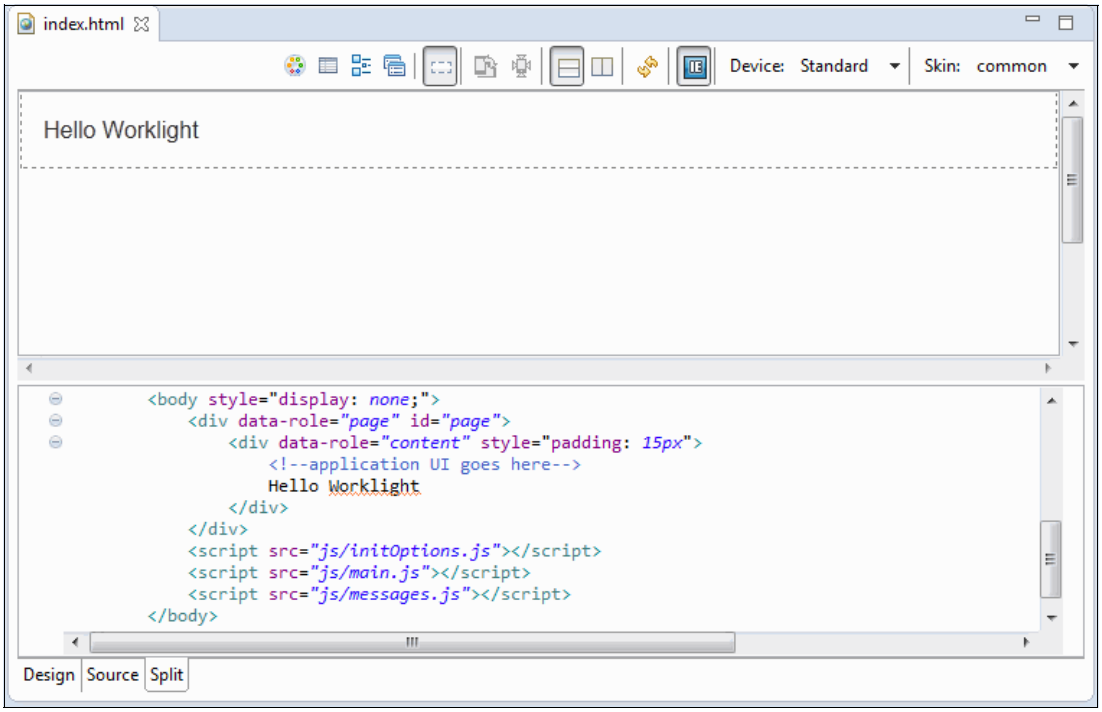


Figure 5-17 Editing the index.html file

3. To create the claim form and tracking number windows, start by duplicating the division (div) marked `data-role="page"` and editing the corresponding `id` value of each division.
4. Name the first div, `id="claim_form_page"` and the second div, `id="tracking_number_page"`. The Hello Worklight content can be deleted. Figure 5-18 on page 146 shows the edited text.

> **Note:** The Mobile Navigation panel in the lower left of the window can be used for easy navigation among pages.

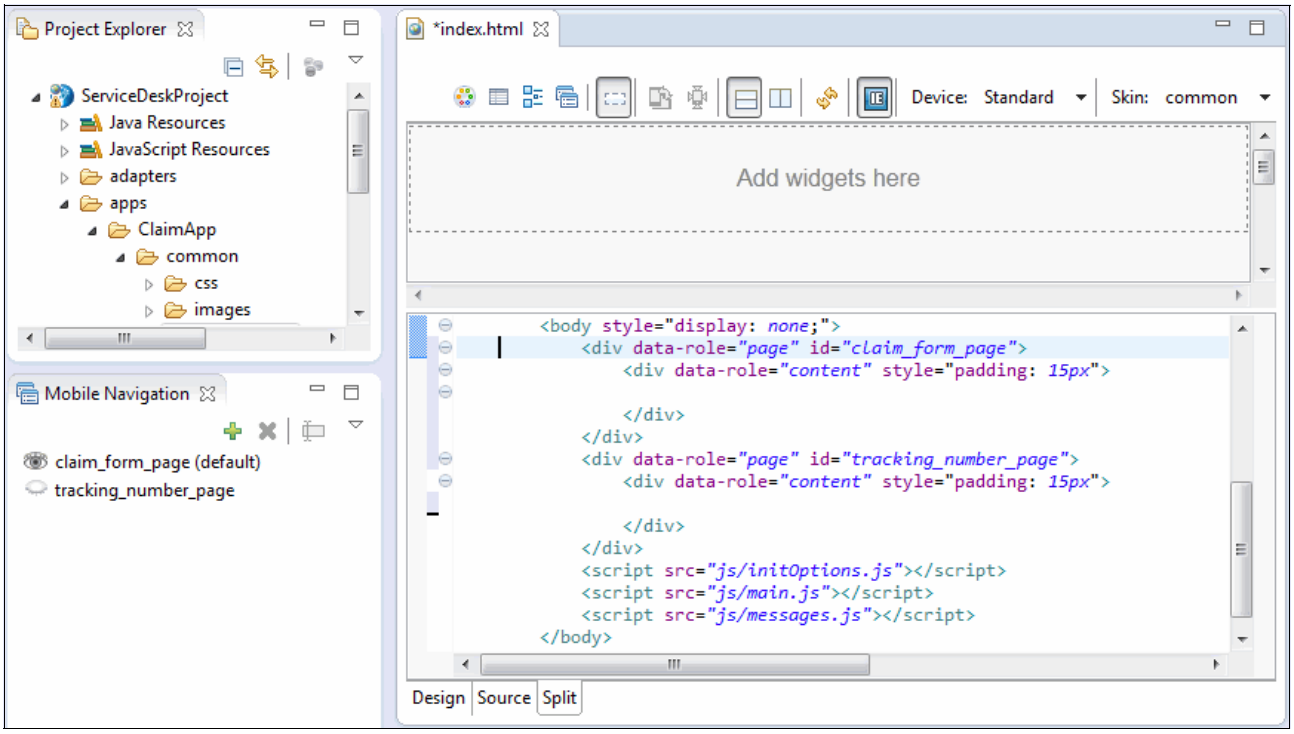


Figure 5-18 Adding pages to index.html

Define the Claim Form window

Continue by defining the claim form window:

1. To create the content for the claim form, select the current content in the Mobile Navigation view. Click the Design tab in the source code section of the window.
2. Locate the `jQueryMobile Widgets` section in the Palette view, and scroll to the **Header** component.
3. Drag the **Header** component into the Design view, in the `claim_form_page` div.
4. Replace the default header text with **Claims**.
5. Create the fields to complete the claim form. Begin by inserting a Form in which to group all of the input fields. This is only used by the Reset button, usually there is no need to insert a form to process the fields, given that we will be accessing these values using jQuery.
6. Locate the Form widget in the `jQueryMobile Widgets` section, and drag it underneath the header section. Change the `id` value to a unique object id, such as `claimForm`, as shown in Example 5-13.

Example 5-13 Form definition

```
<form id="claimForm" action="">
```

7. Locate the Text widget in the `jQueryMobile Widgets` section, and drag it into the `claim_form_page` div in the `form` section. If you look into the source section, you will see that two components have been added: one label tag and one input tag. After editing, these objects will be tied together. The value of the `for` attribute of the label will also be the `id` value of the input. The default drag and drop generates the code shown in Example 5-14 on page 147.

Example 5-14 Text input default definition

```
<label for="text">Text Input:</label><input type="text" name="text" id="text">
```

8. Edit the label code using the `name` value for the `for` attribute, and `Name` as label text.
9. Edit the `input name` and `input id` values using `name` for each of these values. The final code is shown in Example 5-15.

Example 5-15 Text input for Name field

```
<label for="name">Name:</label><input type="text" name="name" id="name">
```

10. Repeat the process to add two more text fields and a text area: `address, serial`, and `comments`, respectively. You can also copy and paste the previous line three times to define these remaining input fields. The last field (`comments`) is a text area, rather than a simple text field, so the syntax is different. The completed, edited fields are shown in Example 5-16.

Example 5-16 Modified text inputs

```
<label for="name">Name:</label><input type="text" name="name" id="name">
<label for="address">Address:</label><input type="text" name="address"
id="address">
<label for="serial">Serial #:</label><input type="text" name="serial" id="serial">
<label for="comments">Comments:</label><textarea id="comments"></textarea>
```

11. Finally, add two buttons: one for submitting the form and one for clearing form contents. To begin, drag a Button widget, and place it below the text area. Edit the Button `href`, `text`, and `button id` values.
12. Add an `onClick` event definition to call the JavaScript that calls the HTTP adapter. This will send the form data to the back end. In section “Code the JavaScript implementation” on page 154, we define the JavaScript function named `submitClaimForm`, which receives the obtained values from the window, as parameters, as shown in Example 5-17.

Example 5-17 Submit button

```
<a href="#tracking_number_page" data-role="button" id="submit"
   onClick='submitClaimForm(claimForm.serial.value, claimForm.name.value,
claimForm.address.value, claimForm.address.value);' data-inline="true">Submit</a>
```

13. Drag and drop another button and edit the code, adding the data shown in Example 5-18.

Example 5-18 Reset button

```
<a href="#" data-role="button" id="button0" data-inline="true"
onClick='claimForm.reset();'>Reset</a>
```

14. Add a `data-inline=”true”` attribute for both buttons. This is for cosmetic reasons only.
15. The input form in the UI is ready. Click **Save**.

Define the Tracking Number window

Continue by defining the Tracking Number window, which will show the tracking number retrieved from the back end:

1. In the Mobile Navigation panel, double-click the **tracking_number_page** option to display the current state of the page in the design panel.
2. Drag and drop a Header widget and use `Tracking Info` as header text.

3. Drag and drop a Label widget, and place it into the content area of the `tracking_number_page` below the header.
4. Edit the object using `resultMessage` as the `id`, and leave the text area of the label blank. This space will be used by the scripts to display the tracking number.
5. Drag and drop a Button widget, using the `href` value that references the claim form page, and `Back` as text. For our scenario, we also defined the button as inline and used the back icon. This section of code is shown in Example 5-19.

Example 5-19 Tracking number page content

```
<div data-role="page" id="tracking_number_page">
   <div data-role="header" id="header0" data-position="fixed">
      <h3>Tracking Info</h3>
   </div>
   <div data-role="content" style="padding: 15px">
      <label id="resultMessage"></label>
      <a href="#claim_form_page" data-role="button" id="button" data-icon="back"
            data-inline="true" data-mini="true" onClick="showFormPage();">Back</a>
   </div>
</div>
```

6. Click **Save** and close the page editor.

Create the HTTP adapter

The HTTP adapter is used to communicate the mobile application with the back-end server.

To create an HTTP adapter, follow these steps:

1. Select **File** → **New** → **Worklight Adapter**. Figure 5-19 on page 149 shows the adapter creation window.

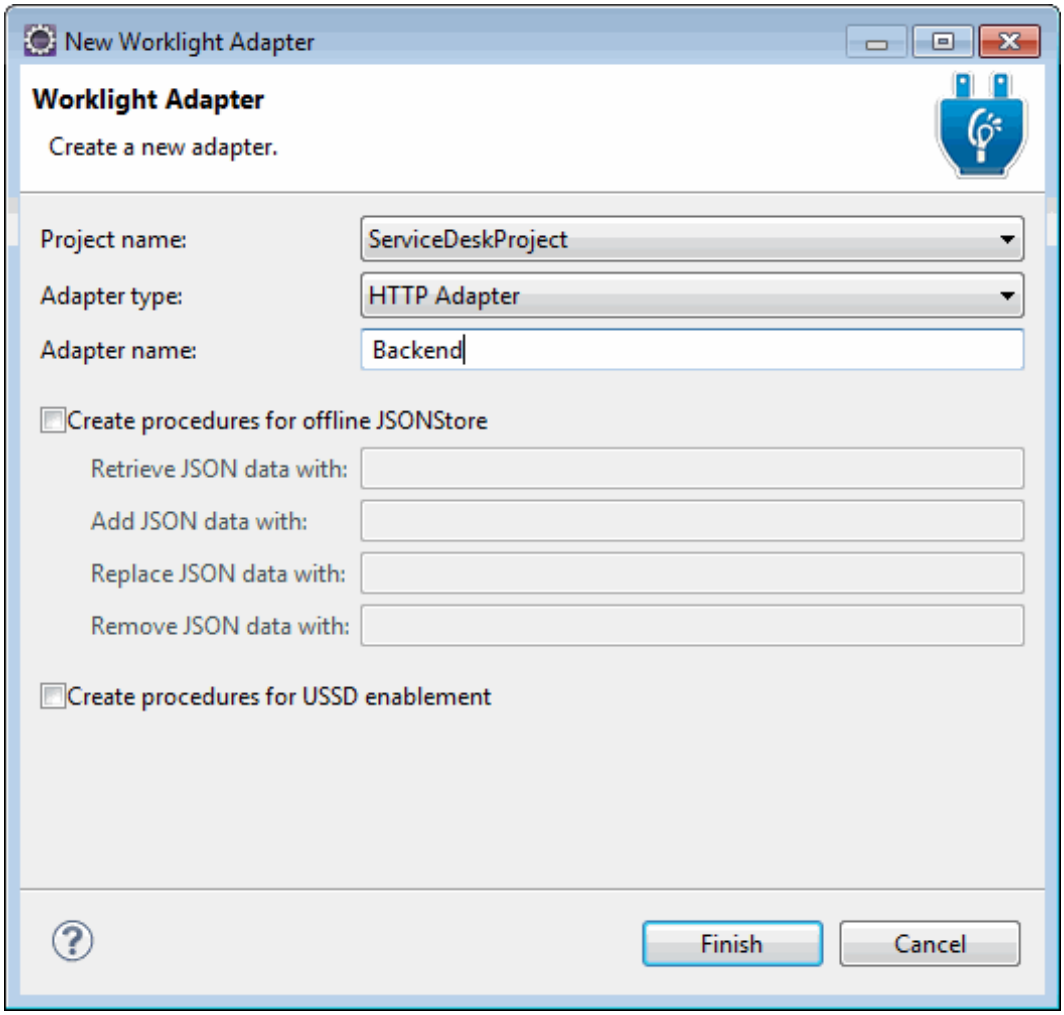


Figure 5-19 New Worklight adapter

2. Populate the form using the following data:
   - *Project name:* The name of the project, in our case **ServiceDeskProject**.
   - *Adapter type:* The type of connection to establish, in our case **HTTP Adapter**.
   - *Adapter name:* A unique name for the adapter, in our case, **Backend**.
3. Click **Finish**. A new folder named `Backend` is now created in the Adapters folder and contains the following files:
   - Backend.xml: Contains the adapter configuration, such as destination IP, or port.
   - Backend-impl.js: Contains the JavaScript that implements the adapter logic.
   - filtered.xsl: Contains the data transformation definition. This file is not needed in our scenario, and can be deleted.

Figure 5-20 on page 150 shows the new adapter with its default values.

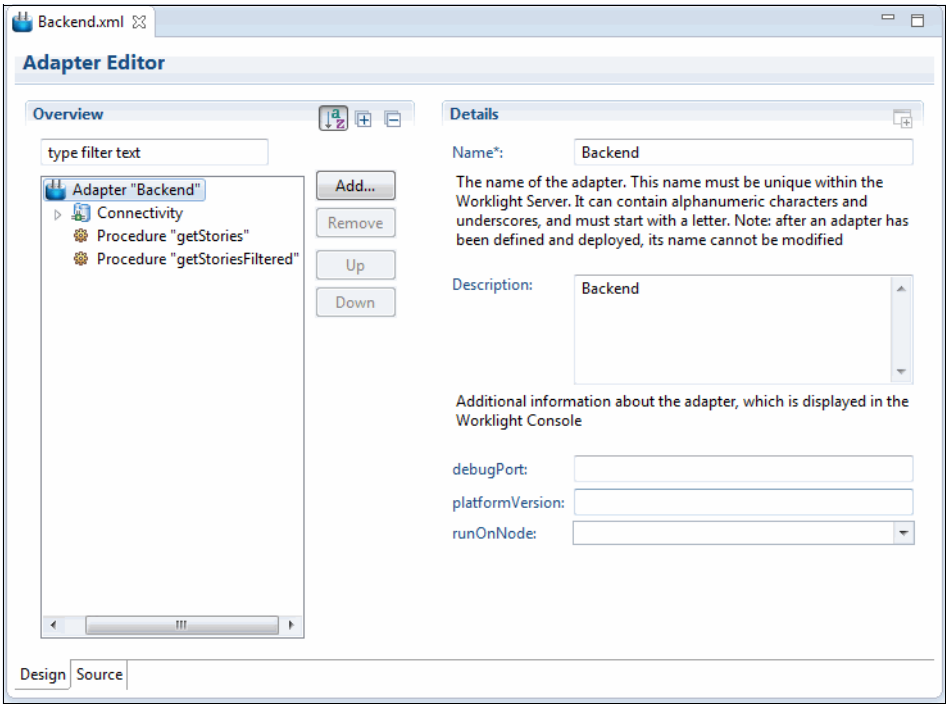


Figure 5-20 Default values for the new adapter

The new adapter is created with sample values, and these need to be modified. To change the adapter values, follow these steps:

1. Expand the **Connectivity** section to edit the Connection Policy values, as shown in Figure 5-21 on page 151.

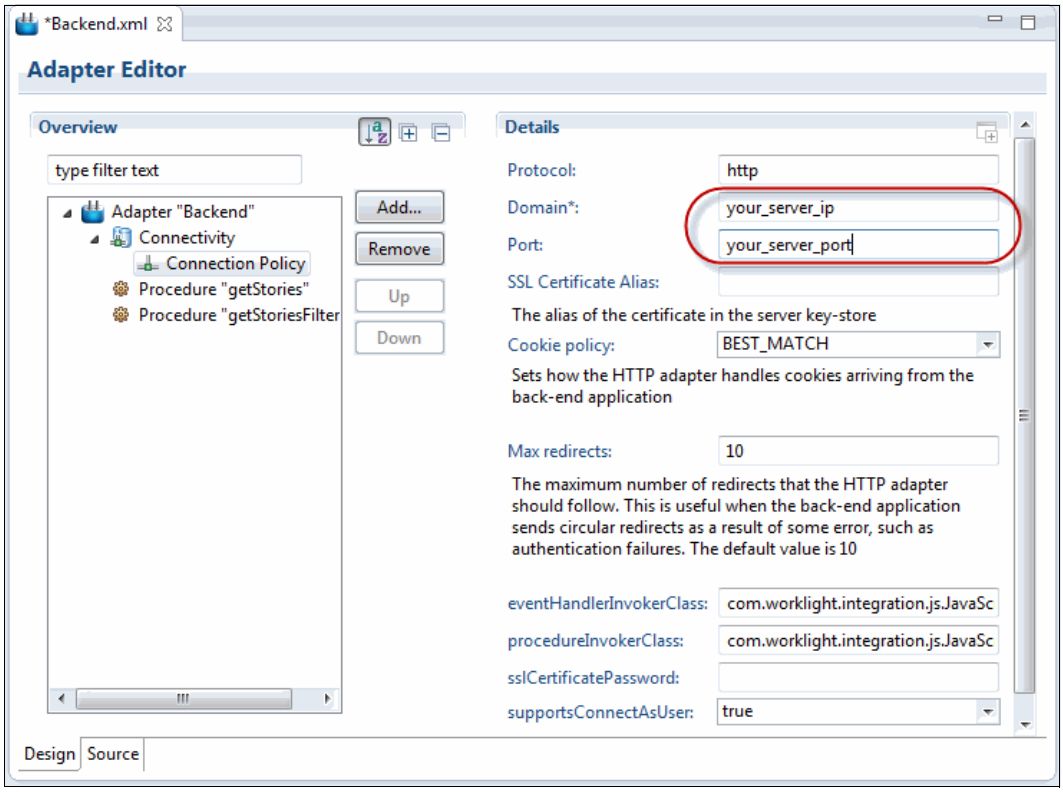


Figure 5-21 Editing the adapter

2. Populate the **Domain** and **Port** values, using the address and port of the back-end server. These values are used by the adapter as the base URL for communicating with the back-end.
3. Delete the predefined procedures **getStories** and **getStoriesFilter**, shown in Figure 5-22 on page 152 by selecting each one and clicking **Remove**.

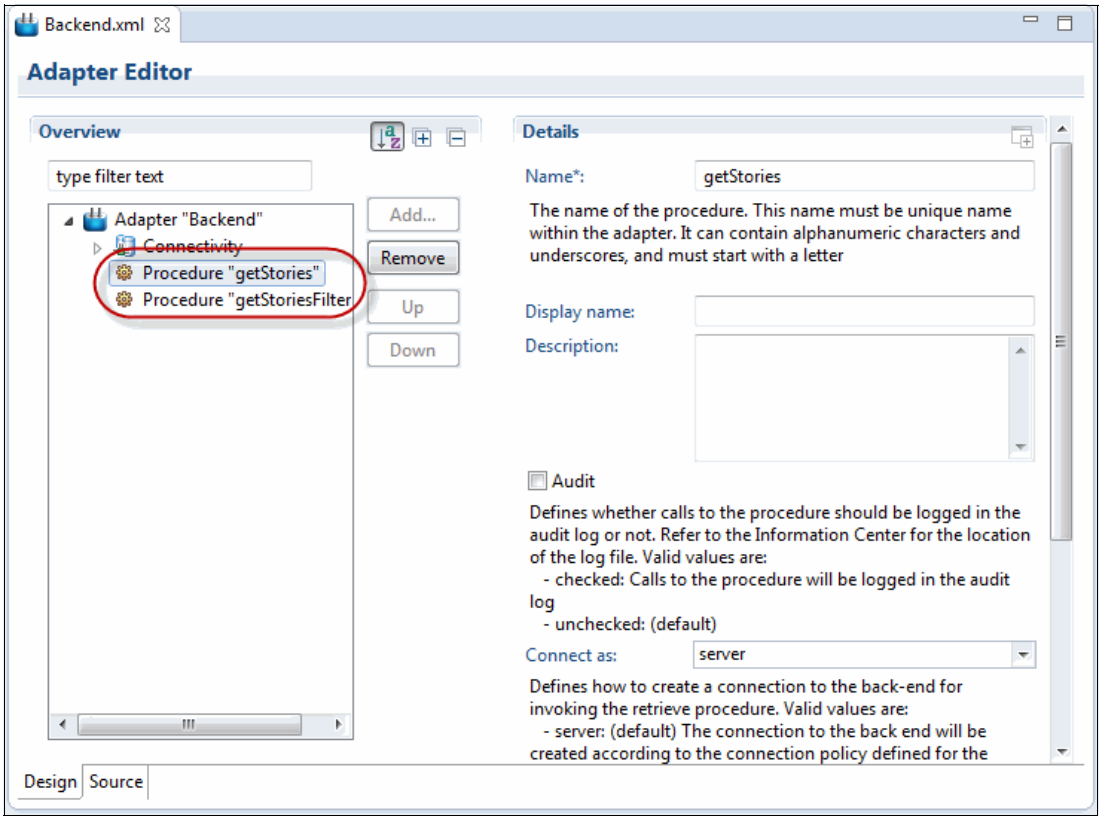


Figure 5-22 Deleting the getStories and getStoriesFilter default procedures

4. Add a new procedure by clicking the **Add** button, and naming it **submitClaim**, as shown in Figure 5-23 on page 153.

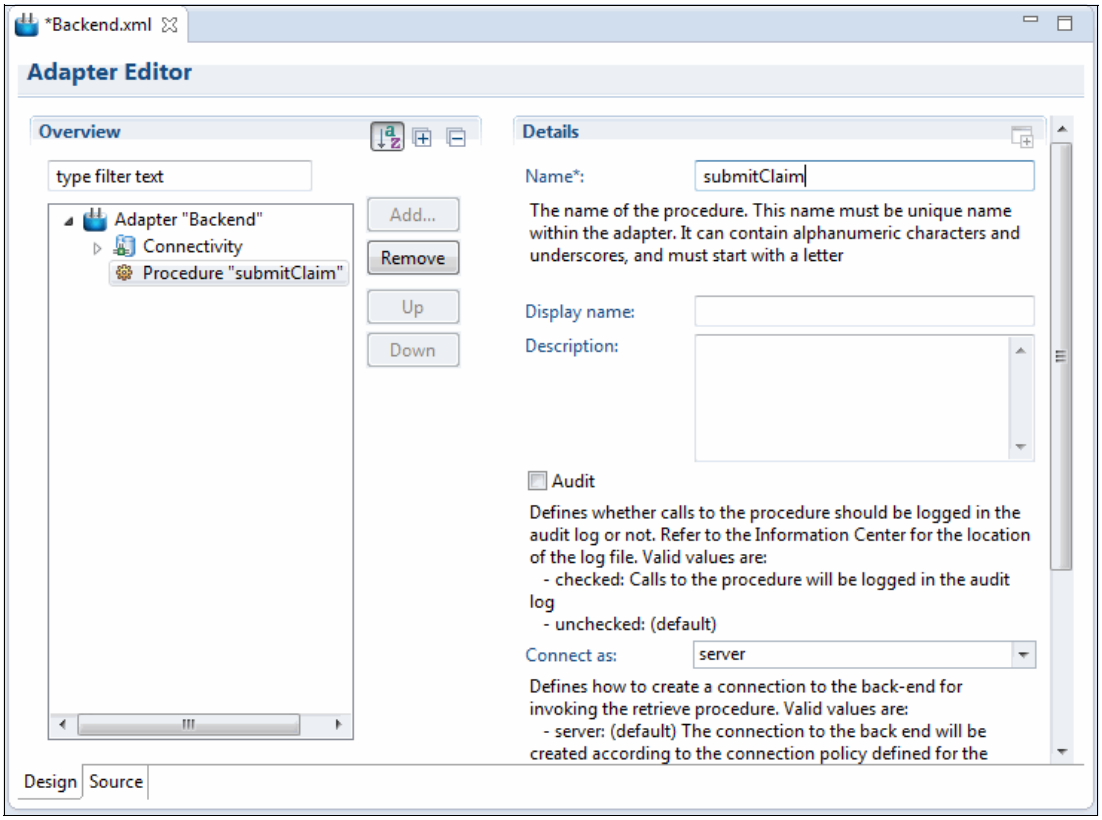


Figure 5-23 Defining adapter methods

5. Switch to the Source view of the adapter. The code displays as shown in Example 5-20.

Example 5-20 Backend adapter source view

```
<?xml version="1.0" encoding="UTF-8"?>
<!--
    Licensed Materials - Property of IBM
    5725-I43 (C) Copyright IBM Corp. 2011, 2013. All Rights Reserved.
    US Government Users Restricted Rights - Use, duplication or
    disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

-->
<wl:adapter name="Backend"
   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>Backend</displayName>
   <description>Backend</description>
   <connectivity>
      <connectionPolicy xsi:type="http:HTTPConnectionPolicyType">
         <protocol>http</protocol>
         <domain>your_server_ip</domain>
         <!-- Following properties used by adapter's key manager for choosing
specific certificate from key store
         <sslCertificateAlias></sslCertificateAlias>
         <sslCertificatePassword></sslCertificatePassword>
```

```
        -->
        <port>your_server_port</port>
      </connectionPolicy>
      <loadConstraints maxConcurrentConnectionsPerNode="2" />
   </connectivity>

   <procedure name="submitClaim" />
</wl:adapter>
```

6. Click **Save** and close the adapter file.

Code the JavaScript implementation

With the adapter defined, we need to write the JavaScript code for the actual implementation:

1. Open the `Backend-impl.js` file for editing. The file contains commented code for reference.
2. Delete all non-commented code before writing the adapter code.

   The code in Example 5-21 implements the call to the backend server.

Example 5-21 Server adapter call

```
function submitClaim(serial, name, address, comments) {

   var input = {
       method : 'post',
         returnedContentType : 'json',
         path : '/MobileServiceDeskApp',

         body:{
            contentType:'application/json; charset=UTF-8',
            content:
               JSON.stringify({
                  "MobileAppRequest": {
                     SerialNum": serial,
                     Name": name,
                     Address": address,
                     Comments": comments
                  }
               })
         }
      };

   return WL.Server.invokeHttp(input);

}
```

By using this code, we are invoking a REST service at: `http://<your_server_ip>:<your_server_port>/MobileServiceDeskApp`, using the HTTP POST method and encoding the received parameters in JSON format.

Deploy the adapter to the server and test the adapter

Deploy the adapter to the local server. Following that, we can save and test the adapter:

1. Go to the Project Explorer view, the `adapters` folder.

2. Right-click the adapter name, and select **Run As** → **Deploy Worklight Adapter**. The adapter is now deployed to the local server.
3. Test the adapter using sample data. To do so, select the adapter name in the `adapters` folder in the `Project Explorer` view.
4. Right-click the adapter and select **Run As -> Invoke Worklight Procedure** (Figure 5-24).

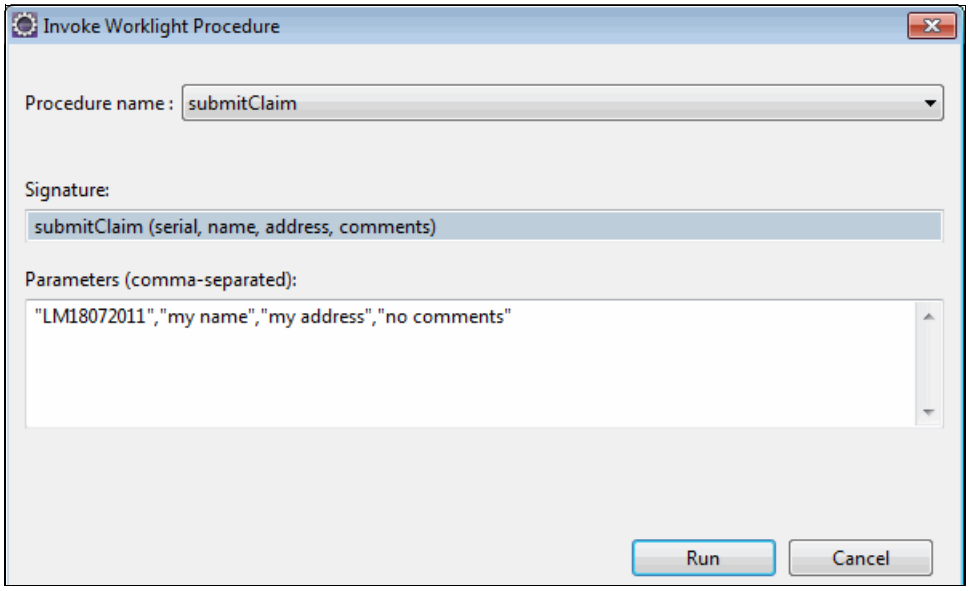


Figure 5-24 Testing the adapter

5. Adapter invocation requires that four, comma-separated input parameters be executed. For testing purposes, any values can be provided. Click **Run** to execute the adapter.

   If the adapter is well defined and the back-end server is running, you obtain a response similar to that in Figure 5-25.

Invocation Result of procedure: 'submitClaim' from the Worklight Server:


```
{
   "MobileAppResponse": {
      "ResponseCode": "OK",
      "TrackingNumber": "MOBFCD205F6"
   },
   "isSuccessful": true,
   "responseHeaders": {
      "Content-Length": "74",
      "Content-Type": "application\/json; charset=utf-8",
      "Date": "Thu, 04 Sep 2014 19:38:04 GMT",
      "Server": "Apache-Coyote\/1.1"
   },
   "responseTime": 31,
   "statusCode": 200,
   "statusReason": "OK",
   "totalTime": 31
}
```


Figure 5-25 Adapter invocation result

Call the adapter from the UI

The last step in mobile application development is to link the UI and the adapter, so that when the user clicks the **Submit** button, the adapter is called using the values captured on the input form.

During UI development, a JavaScript method named `submitClaimForm()` was invoked in the onClick event of the **Submit** button. This method is defined in the JavaScript portion of the project. Because it is the same for all environments, adding the corresponding code to the common components is enough.

1. Open the `main.js` file located in `apps/ClaimApp/common/js`. At this time, the file contains only the default initialization code of the `wlCommonInit()` function.
2. After the `wlCommonInit()` function, create a new function, as shown in Example 5-22.

Example 5-22 Adapter invocation in main.js

```
function submitClaimForm(serial, name, address, comments) {

   var invocationData = {
           adapter : 'Backend', // adapter name
           procedure : 'submitClaim', // procedure name
           parameters : [serial, name, address, comments] // parameters if any
   };

   WL.Client.invokeProcedure(invocationData, {
      onSuccess : dataSentOK,
      onFailure : dataSentError
   });

}
```

3. In this function, we are receiving the claim form input parameters and constructing the HTTP adapter call to invoke the adapter that was created in “Create the HTTP adapter” on page 148. Note that two additional functions are required, mapped as `onSuccess` and `onFailure` in the adapter call, and shown in Example 5-23.

Example 5-23 Adapter invocation post-processing

```
function dataSentOK(result) {
   if ("OK"== result.invocationResult.MobileAppResponse.ResponseCode) {
      $("#resultMessage").html("Your claim has been received.<br>Tracking #:
"+result.invocationResult.MobileAppResponse.TrackingNumber);
   }
   else {
      $("#resultMessage").html("Your claim has been rejected.<br>Please contact
Customer Service.");
   }
}

function dataSentError(result) {
   $("#resultMessage").html("We are experiencing technical difficulties.<br>Please
retry your operation later.");
}
```

4. In a normal execution of the adapter, a response code will be retrieved, along with the tracking number if the operation was successful. The corresponding messages will be displayed in the results window.
5. If there is an execution error (such as connectivity problems), a different message will be delivered to the user.
6. **Save** and close the file.

Deploy and test

Now, development is complete, and the full UI and adapter integration can be tested.

Assuming that the adapter is deployed and tested as indicated in section “Create the HTTP adapter” on page 148, we are ready to build the application environment for preview and unit testing.

Build the application environment

Build the application environment by following these steps:

1. Right-click `apps/ClaimApp` and select **Run As** → **Build all Environments**. This starts the build process.
2. Monitor the build process in the Console view. If there are no errors, the following message in Example 5-24 displays.

Example 5-24 The build process completes without errors

```
[yyyy-mm-dd hh:mm:ss] Application 'ClaimApp' with all environments build finished.
```

Test the application

Now, test the application by following these steps:

1. Right-click `apps/ClaimApp` and select **Run As** → **Preview**. The mobile browser simulator will be opened in your default browser, showing the application in a simulated mobile device.

   Keep in mind that the mobile browser simulator uses a Java applet, so the default browser must have a configured Java Runtime Environment. Also, some security warnings might display the first time that the tool is executed. Just accept and allow the execution of the presented content. Otherwise, the mobile application might not work properly.

Figure 5-26 on page 158 shows the application running in the mobile browser simulator.

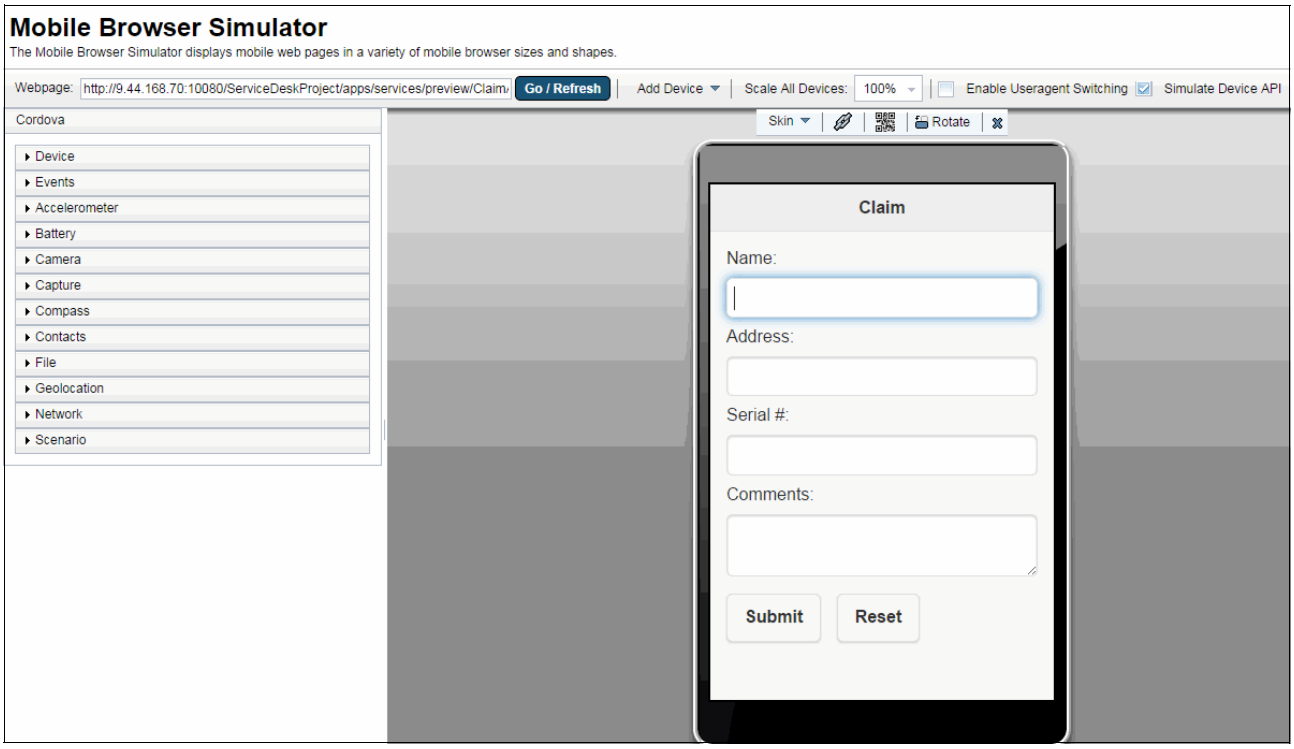


Figure 5-26 Mobile browser simulator

Generating the application for different platforms

One of the main capabilities of Worklight is the ability to generate multiple-platform applications. Specific requirements might be needed to generate a native application that can be installed on a real device. To check for these requirements, see the IBM Knowledge Center:

http://www-01.ibm.com/support/knowledgecenter/SSZH4A_6.2.0

For our scenario, we generate both an Android application and a mobile website.

Generate an Android and a mobile web application

To create a new environment, follow these steps:

1. In Worklight Studio, select **File** → **New** → **Worklight Environment**.
2. Select the project and application and check the **Android phones and tablets** option and the **Mobile web app** option, as shown in Figure 5-27 on page 159.

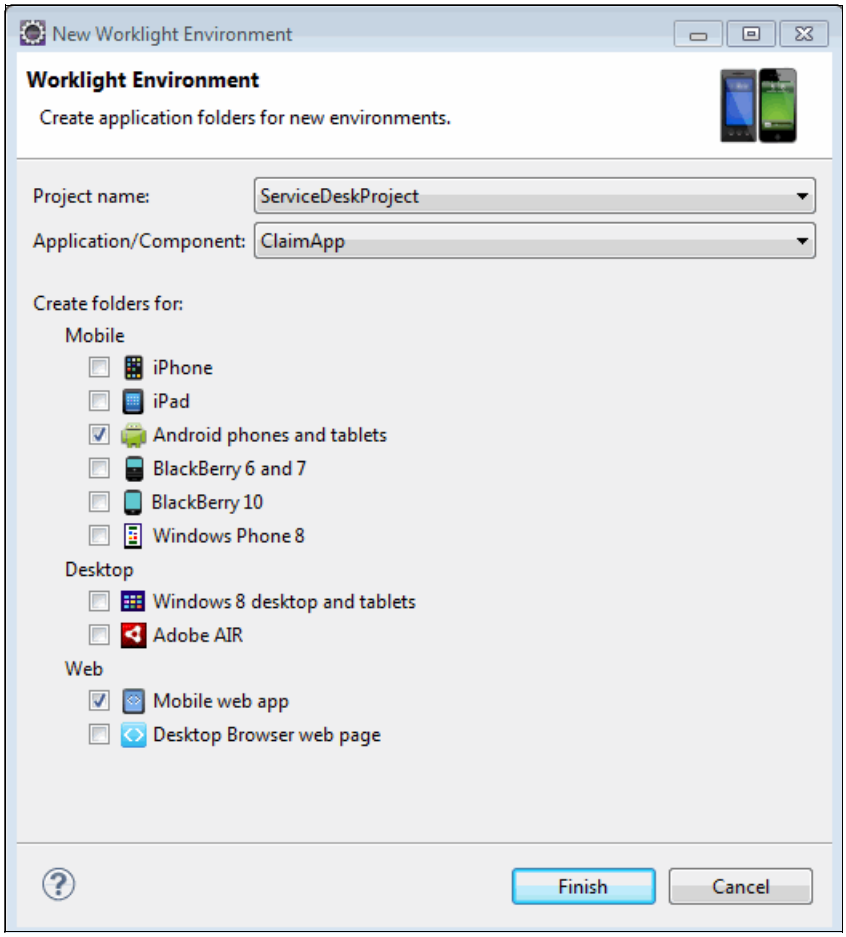


Figure 5-27 New Worklight environment

3. Click **Finish**, and the two folders are created under `ClaimApps`: an `android` folder and a `mobilewebapp` folder.
4. Now, build the new environment. Right-click **ClaimApps** and select **Run As** → **Build All Environments**.
5. When the building process ends, a new Android project is created. It is named using the pattern `<protect_name><app_name>Android`. In our scenario, the name is `ServiceDeskProjectClaimAppAndroid`.
6. This is a normal Android project, so it can be tested as such. Right-click the `ServiceDeskProjectClaimAppAndroid` project and select **Run As** → **Android Application**. This launches the Android emulator and runs the application inside it.

5.5.3 Install the application on the server

To install the mobile application on the Worklight server *for the first time*, follow these steps:

- "Generate the installable assets" on page 160
- "Install the base software" on page 160
- "Configure the server" on page 162
- "Create a Worklight runtime environment" on page 169
- "Add the mobile application" on page 172

Generate the installable assets

To deploy the application to an external Worklight server, a few configuration steps are needed.

The application is exported in the form of a `.war` file, which is generated by you, to the target Worklight server. After the export and installation of the `.war` file, the Worklight console is enabled and is used to install the remaining components: the Worklight application and the HTTP adapter.

1. To generate the `.war` file, in Worklight Studio, right-click the `ClaimApp` application and select **Run As** → **Build Settings and Deploy Target**. Figure 5-28 shows the configuration window for this file.

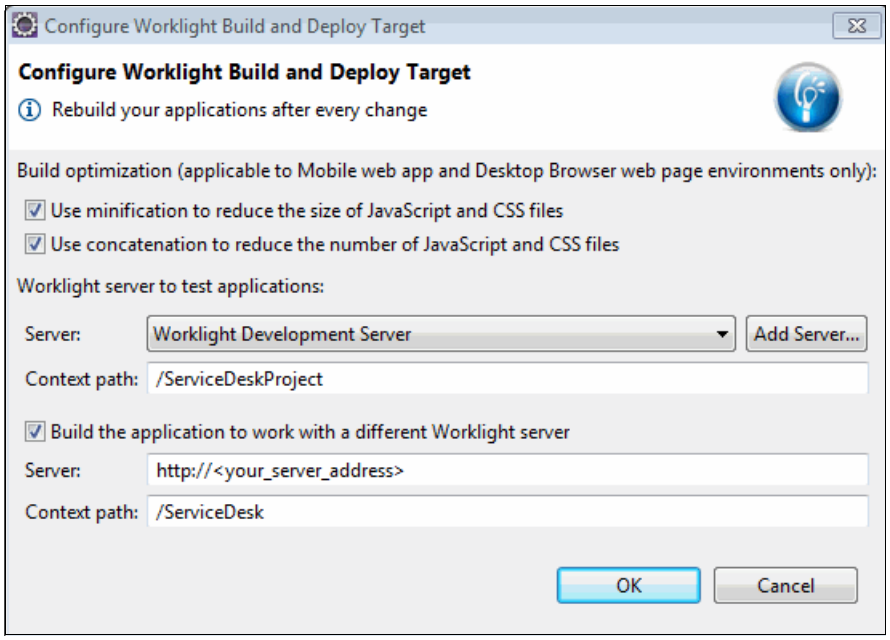


Figure 5-28 Configure your Worklight build and deploy the target

2. In the window that displays (Figure 5-28), check the first two check boxes if they are not selected. This optimizes the size of the deployed application.
3. Check the option to **Build the applications to work with a different Worklight server**. This updates the calls and references to the new server installation. Include the destination server name or IP address and a context path.

   The context path is part of the invocation when accessing the application from the mobile-web environment. In our scenario, we chose **/ServiceDesk**.
4. Click **Ok.**
5. Build the application by right-clicking **ClaimApp** and selecting **Run as** → **Build All Environments**.
6. Look in the bin folder for the `ServiceDeskProject` and the newly generated files. Copy the following files, as they will be used later for installation:
   - *Backend.apapter:* The adapter installer
   - *ClaimApp-all.wlapp:* The application installer
   - *ServiceDeskProject.war:* The war file containing the application runtime environment

Install the base software

There are several approaches to installing a mobile application on the Worklight server.

For our scenario, we chose to use the Server Configuration Tool for the initial application deployment. For a complete reference of installation choices, see the product documentation at the IBM Knowledge Center:

https://www-01.ibm.com/support/knowledgecenter/SSZH4A_6.2.0/com.ibm.worklight.deploy.doc/devref/t_transporting_apps_and_adapters.html

To do the installation using our selected method, follow these steps:

1. Install the supporting database. In our scenario, we chose an IBM DB2 compact installation, without modifying the default configuration values. Other database management systems are supported. For a list of these, see the product documentation at the IBM Knowledge Center:

   https://www-01.ibm.com/support/knowledgecenter/SSZH4A_6.2.0/com.ibm.worklight.deploy.doc/devref/t_transporting_apps_and_adapters.html

2. Install the application server to run Worklight. In our scenario, we chose to use IBM WebSphere Application Server Liberty Core.
3. After installing the application server, but before starting the Worklight installation, it is necessary to create an empty server instance on the application server. Navigate into the bin folder of the Liberty installation, and execute the following command:

   ```
   ./server create myServer
   ```

4. Using the IBM Installation Manager, start the installation of the Worklight Server similarly as you did in the previous step. After selecting the location and features, a window provides configuration information, as shown in Figure 5-29.

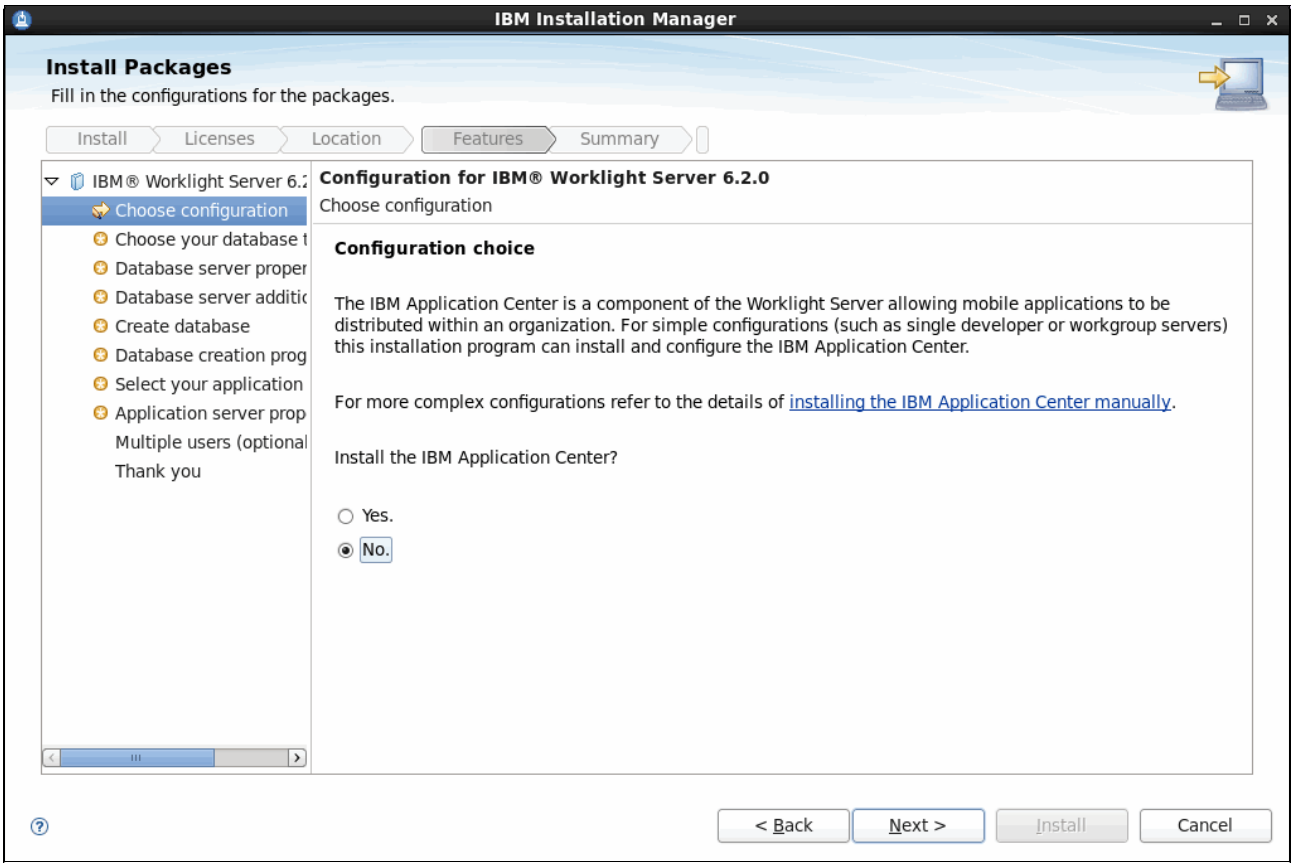


Figure 5-29 IBM Worklight server configuration

5. For our scenario, we did not need to install the Worklight Application Center. Therefore, when prompted to install the Application Center, choose **No** and click **Next**.
6. No further configuration is needed, and so the next windows can be bypassed by clicking **Next**, according to the on-screen instructions. At the last window, the server installation starts.
7. At the end of the installation, click **Finish**.

You are ready to use the Worklight Server Configuration Tool.

Configure the server

To configure the server using the Server Configuration Tool, follow these steps. It is expected that the instance owner is performing these configuration steps:

1. Navigate to: `<worklight_base_dir>/WorklightServer/ConfigurationTool/<your_platform>` and locate the corresponding executable `ServerConfigurationTool` file.
2. Start the Server Configuration Tool by executing the file found in the previous step.
3. Select **Create a Worklight Server configuration**, as shown in Figure 5-30.

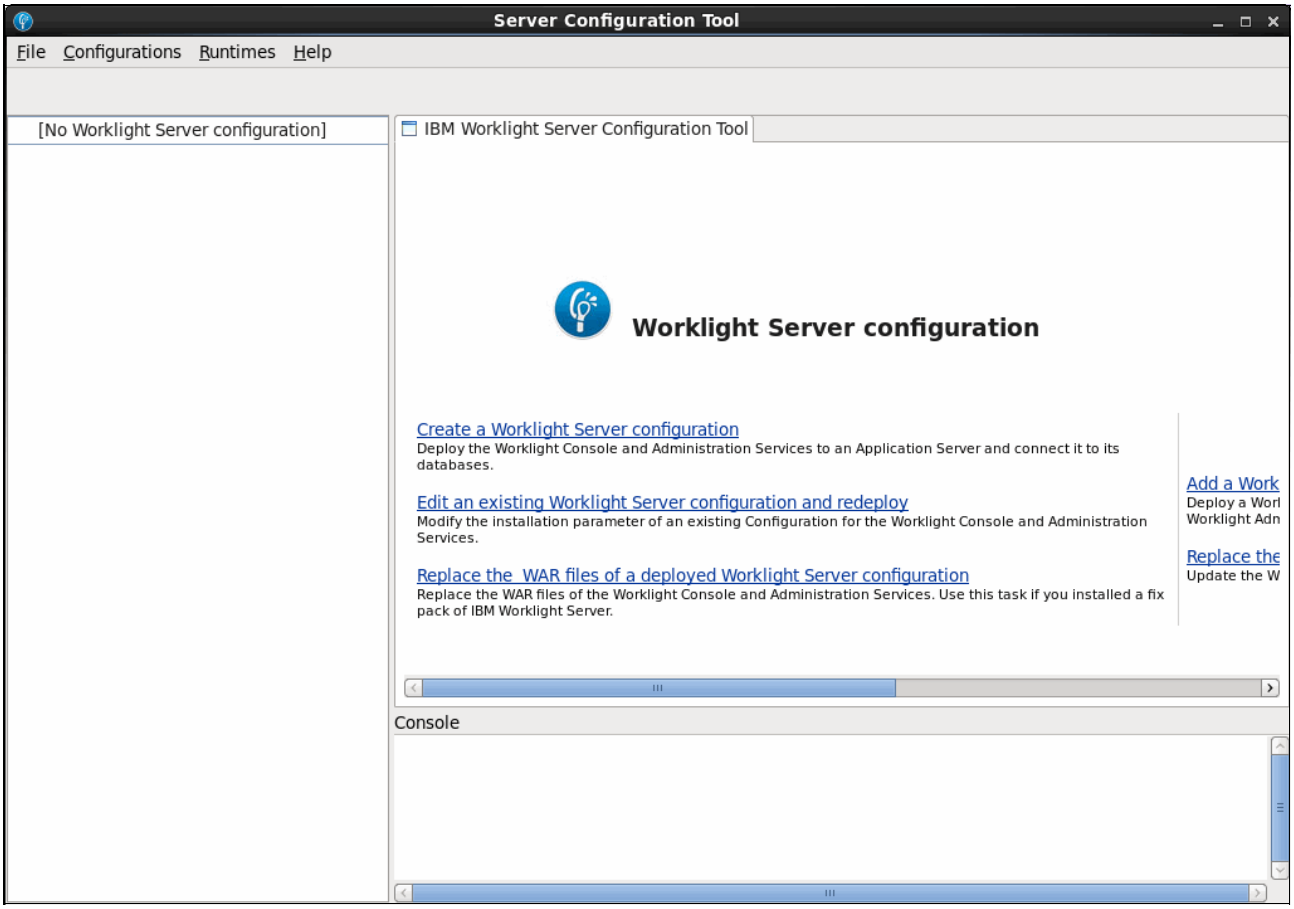


Figure 5-30 Server Configuration Tool

4. Enter a configuration name, such as **base**, and click **OK.** The Configuration Details window displays, as shown in Figure 5-31 on page 163.

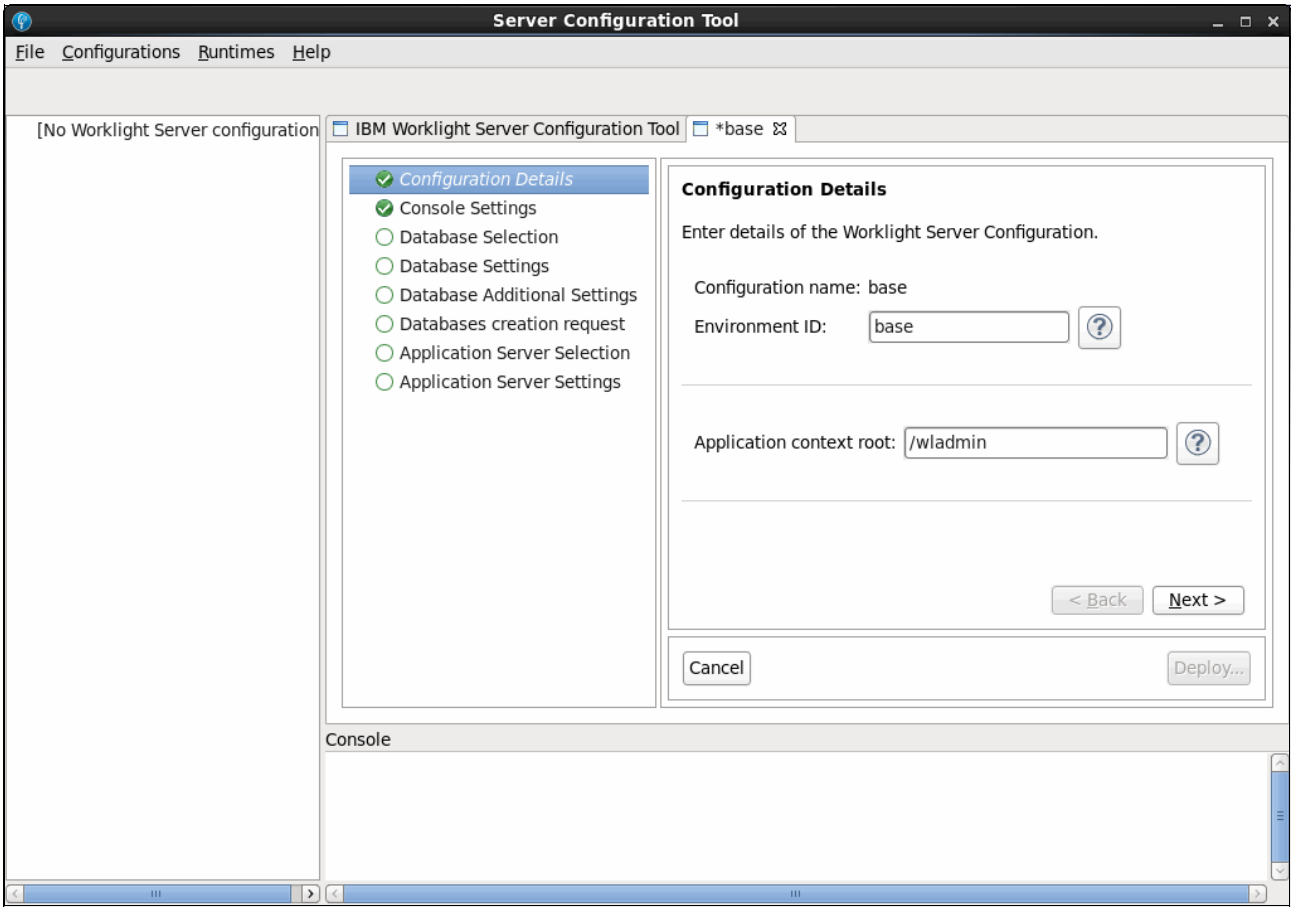


Figure 5-31 The Configuration Details window

5. Leave the default values as they are, and click **Next** to advance to the Console Settings window, as shown in Figure 5-32 on page 164.

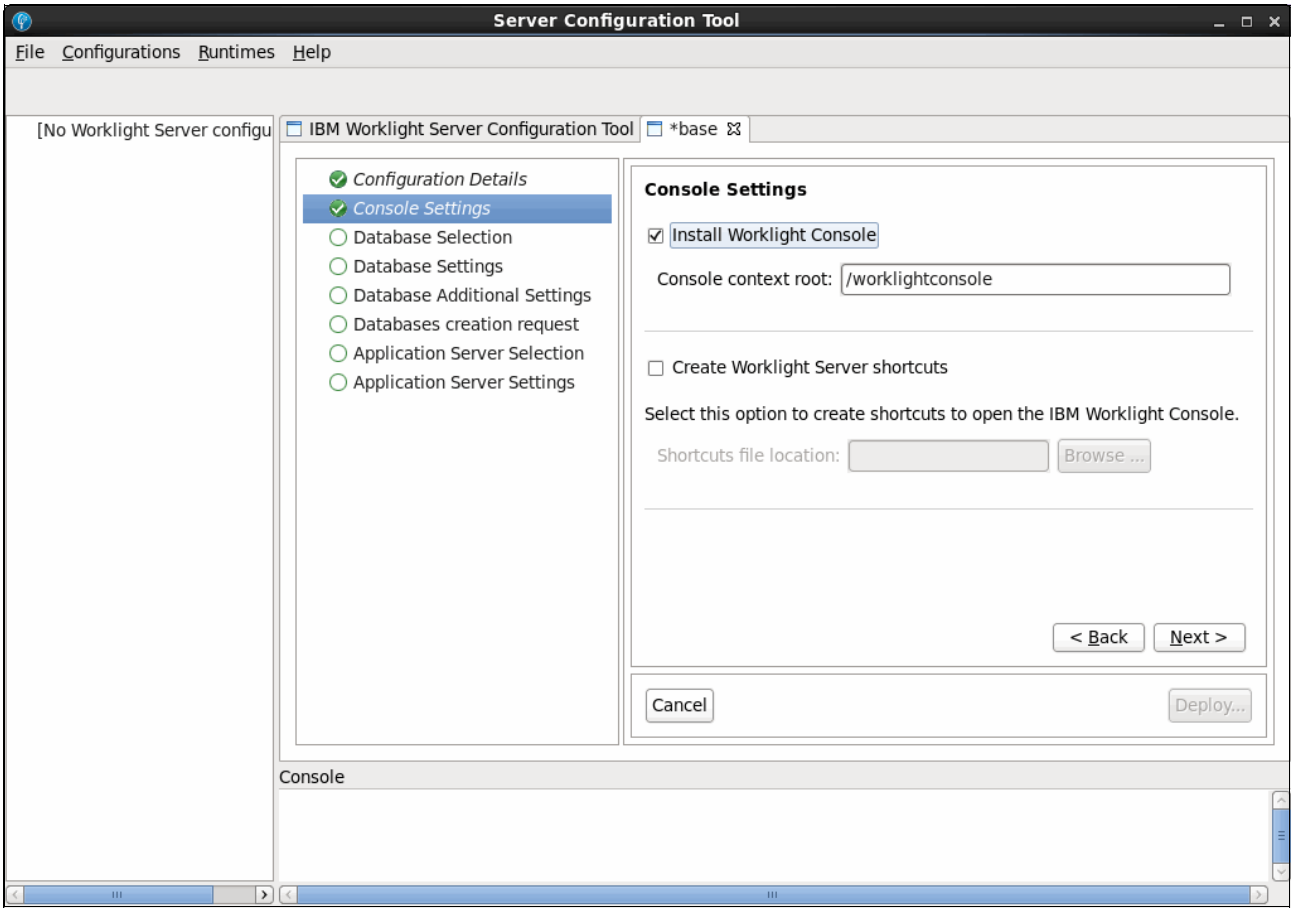


Figure 5-32 The Console Settings window

6. Leave the default values as they are, and click **Next** to advance to the Database choice window, as shown in Figure 5-33 on page 165.

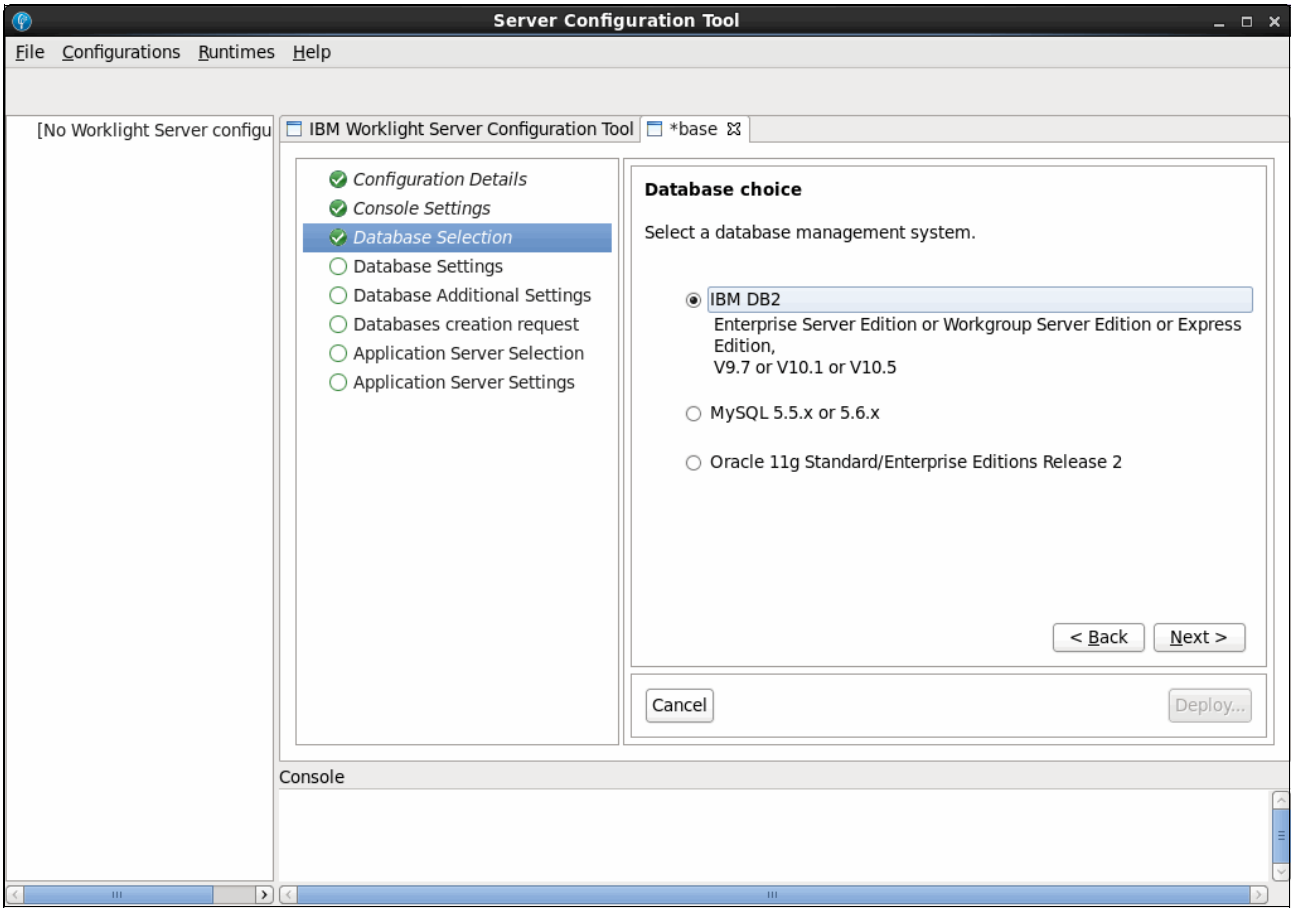


Figure 5-33 Database choice window

7. In the Database choice window, select the supporting database installation, which, for our scenario, is **IBM DB2**. Click **Next** to advance to the DB2 Database Settings window, as shown in Figure 5-34 on page 166.

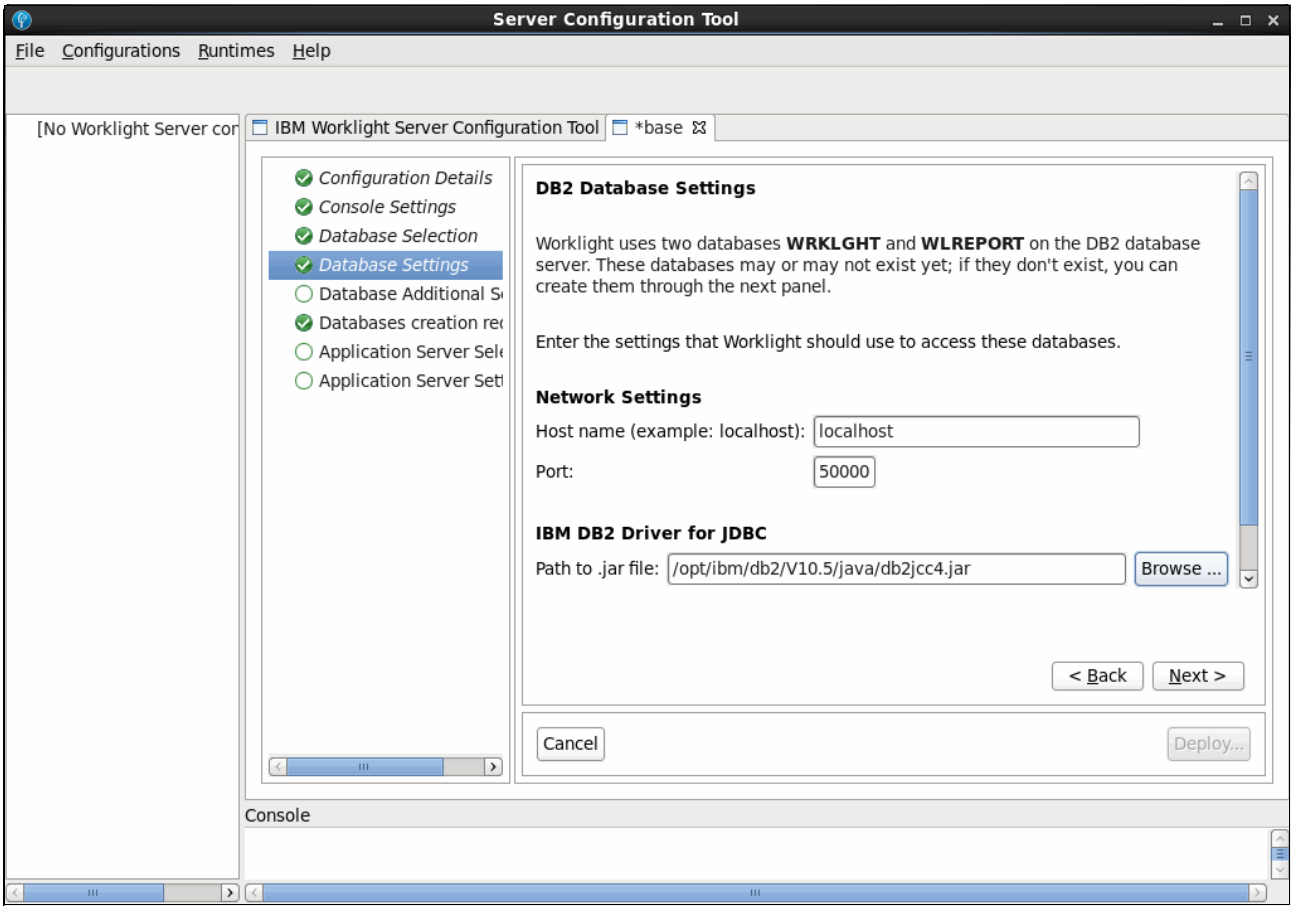


Figure 5-34 The DB2 Database Settings window

8. In the DB2 Database Settings window, enter the database server location, port, and driver. For our scenario, we used:
   - *Default port:* `localhost, 5000`: Modify this for your system.
   - *Driver location:* `/opt/SA-W401/ibm/db2/V10.5/java/db2jcc4.jar`: The link for downloading the DB2 driver is in the bottom of the window. (This link is not visible in Figure 5-34.)
9. Click **Next** to advance to the DB2 Database Additional Settings window, as shown in Figure 5-35 on page 167.

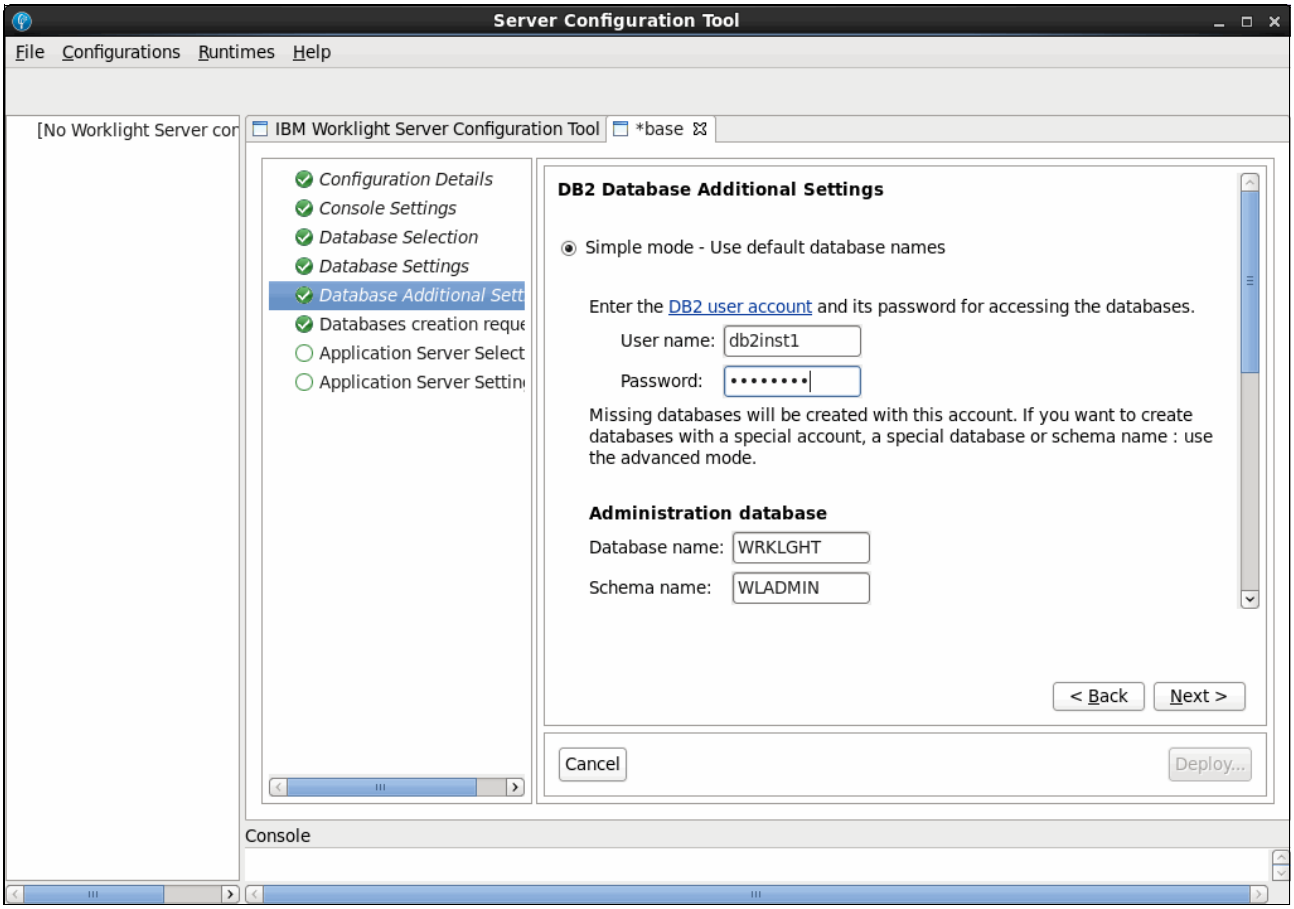


Figure 5-35 DB2 Database Additional Settings window

10. In the DB2 Database Additional Settings window, we chose the simple mode, as this allows the tool to create the databases automatically.
11. For the user and password credentials, we entered `db2inst1`, which refers to the DB2 instance owner in our system. You can also use any user with the privilege to create databases in your system. Click **Next** to advance to the Create DB2 Databases window, as shown in Figure 5-36 on page 168.

> **Note:** If for any reason the tool is unable to create the database in simple mode, you can switch to advanced mode and create the WRKLGHT database from outside the tool. In the next step, it will be recognized as an existing database and the process will continue.

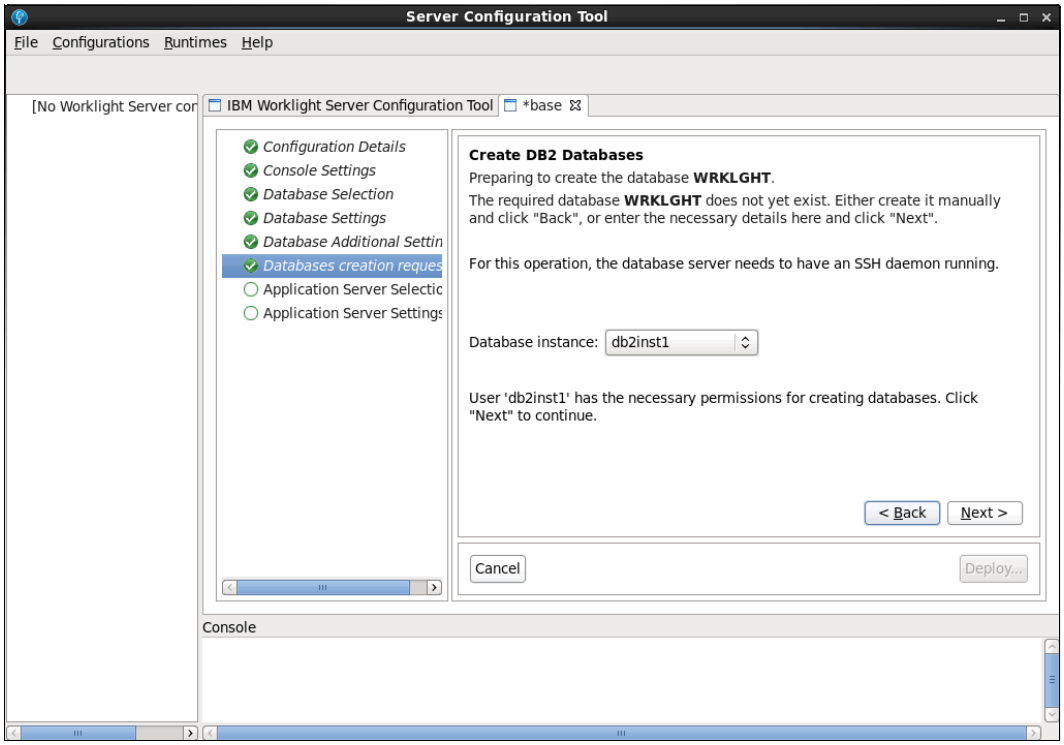


Figure 5-36 The Create DB2 Databases window

12. The next window confirms that the user has permission to create the database. Because we are using the instance owner, this is usually not a problem. Click **Next** to advance to the Application Server Selection window, as shown in Figure 5-37.

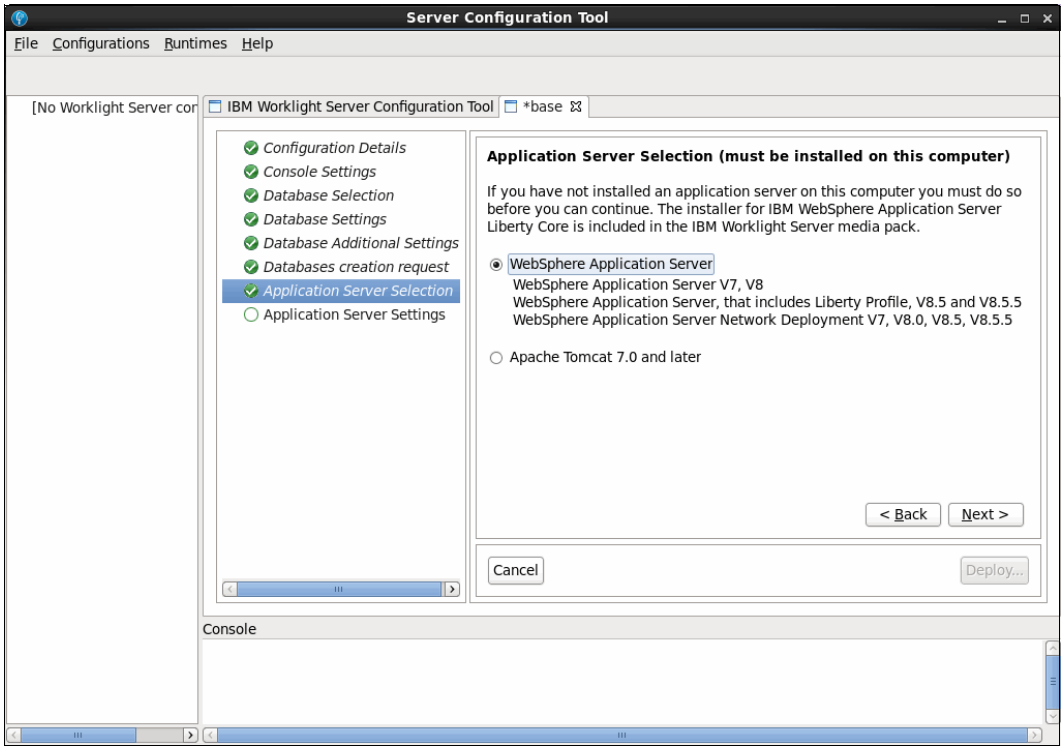


Figure 5-37 Application Server selection

13. In the Application Server Selection window, choose the application server that was installed to support IBM Worklight. For our scenario, we used the WebSphere Application Server. Click **Next** to advance to the Application Server Settings window, as shown in Figure 5-38.

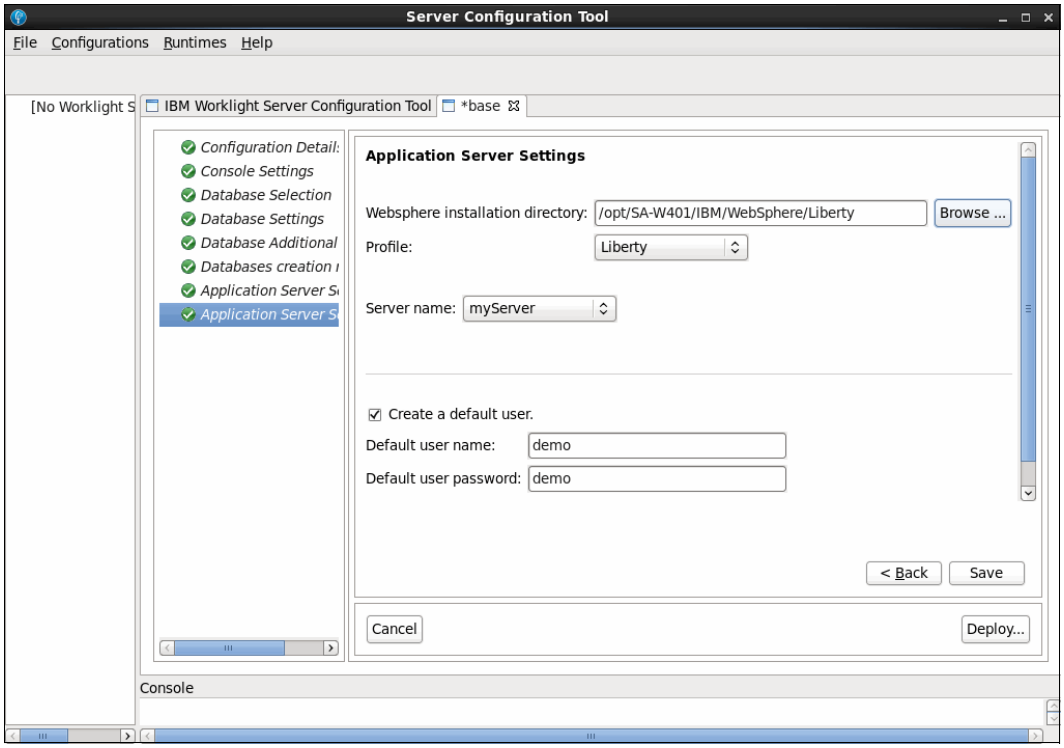


Figure 5-38 The Application Server Settings window

14. In the Application Server Settings window, choose the installation path for the application server. For our scenario, we used `/opt/SA-W401/IBM/WebSphere/Liberty`. If the path is correct, the profile and server that were created in Step 3 on page 161 will be available for selection.

15. Ensure that the **Create default user** option remains checked. This user will be able to access the Worklight console and can change this setting if needed. For simplicity, we retained the default value.

16. Click **Save** to save all of the changes. Then, click **Deploy** to begin the server deployment.

17. Check the console as the operation progresses.

Create a Worklight runtime environment

The next step is to create a Worklight environment and import the server component of the mobile application:

1. In the ServerConfigurationTool main window select the option to **Add a Worklight Runtime Environment to a Configuration**.

2. When prompted, select a configuration and a name for the environment. For our scenario, there is only one configuration named `base`. For the environment name, we chose `base Environment`.

3. Click **OK** to start the configuration, as shown in Figure 5-39 on page 170.

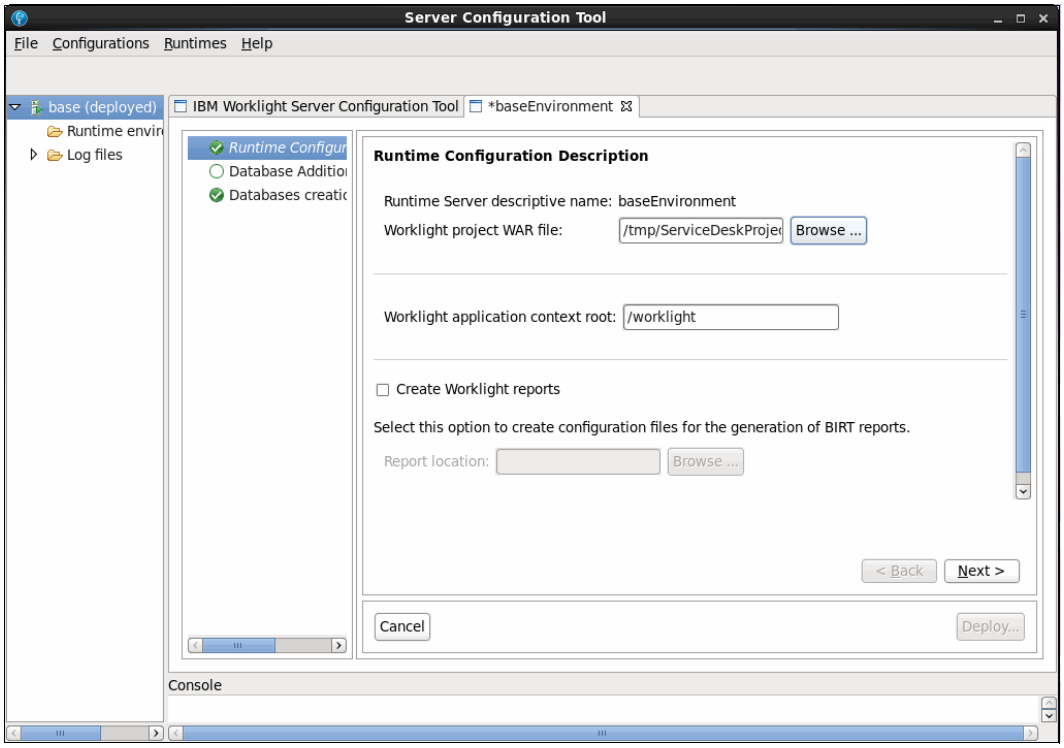


Figure 5-39 The Runtime Configuration Description window

4. Import the project `.war` file that was generated in “Generate the installable assets” on page 160. Click the **Browse** button and select the file.
5. For **Worklight application context root**, use `/worklight` value, and click **Next** to advance to the DB2 Database Additional Settings window, as shown in Figure 5-40 on page 171.

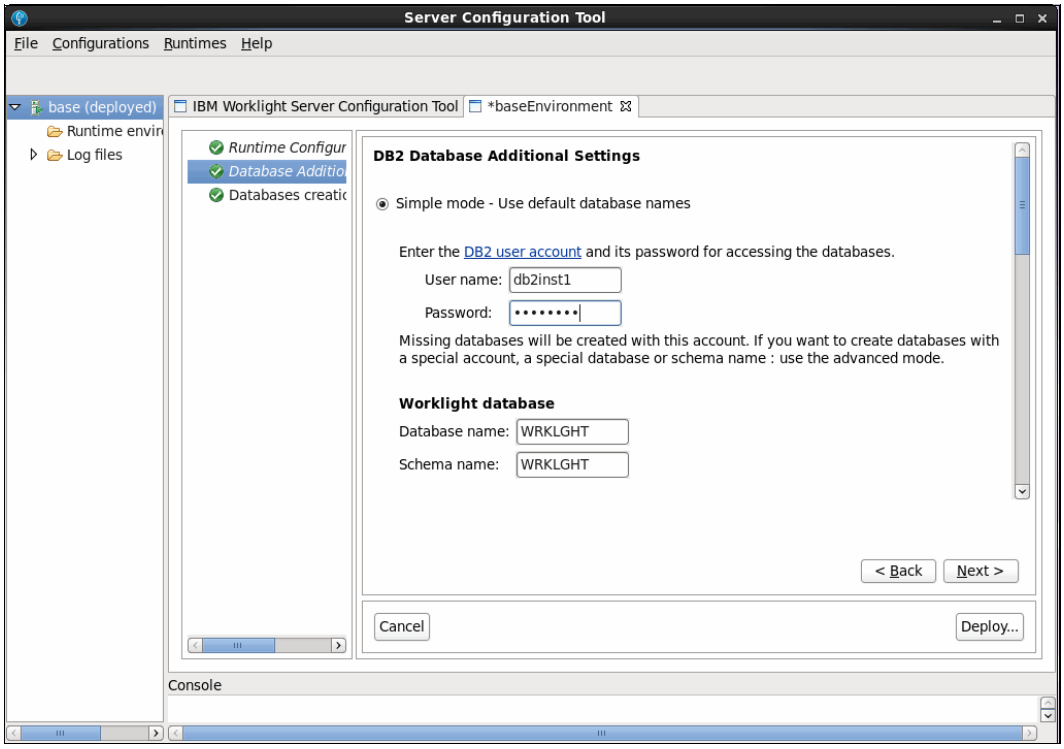


Figure 5-40 The DB2 Database Additional Settings window

6. Select Simple mode: Use default database names, and enter the user name and password for database access. For our scenario, we chose `db2inst1` for both of these values. Click **Next** to advance to the Create DB2 Databases window, as shown in Figure 5-41.

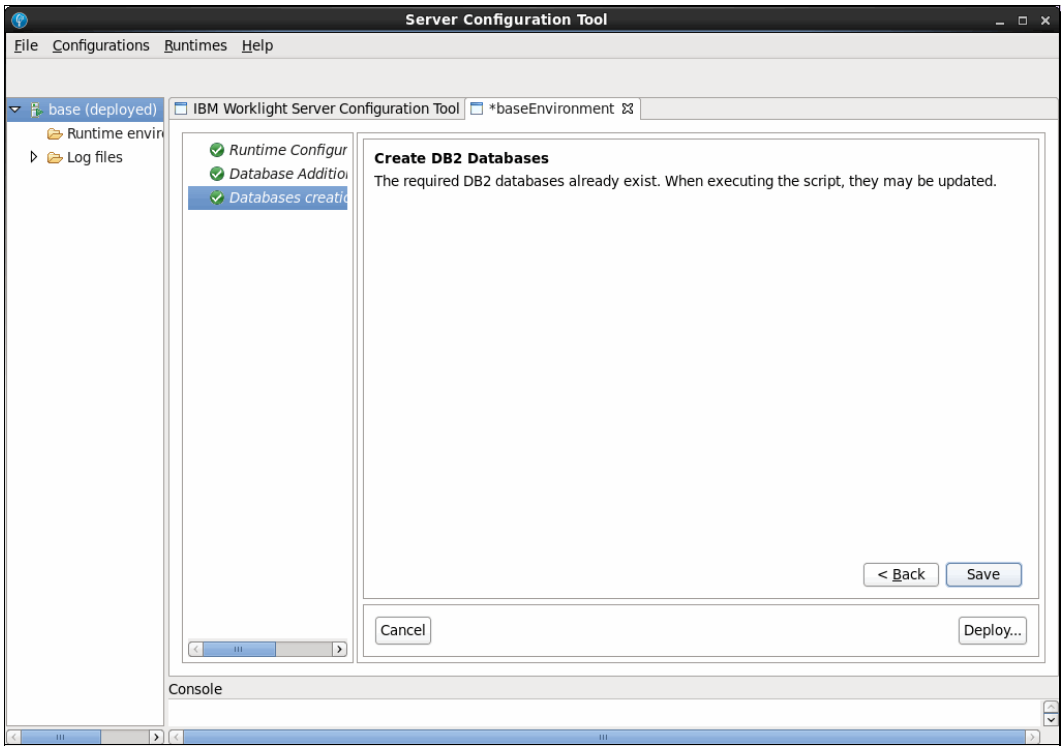


Figure 5-41 The Create DB2 Databases window

7. We are using the database that was created in Step 10 on page 167, and so no additional configuration is needed. Click **Save**. Then, click **Deploy** to deploy the environment.
8. Monitor the console messages to verify the deployment. If there are no errors, close the Server Configuration Tool.

Add the mobile application

Using the Worklight console, add the mobile application to the server and enable it to be used by remote clients.

To add the application to the server, follow these steps:

1. Start the server that was created in Step 3 on page 161 by executing the following command at the bin folder of WebSphere Application Server Liberty Core installation:

   ```
   ./server start myServer
   ```

2. Open a web browser to access the Worklight console at `http://<server_ip>:<server_port>/worklightconsole`, using your server IP and application server port. For our scenario, we use port **9080** as it is the default WebSphere Liberty port.
3. When prompted, enter the user name and password that were defined in step 14 on page 169. For our scenario, we left the default user name and password as `demo`. Figure 5-42 shows the console login. Click **Log in** to advance into the console workspace.

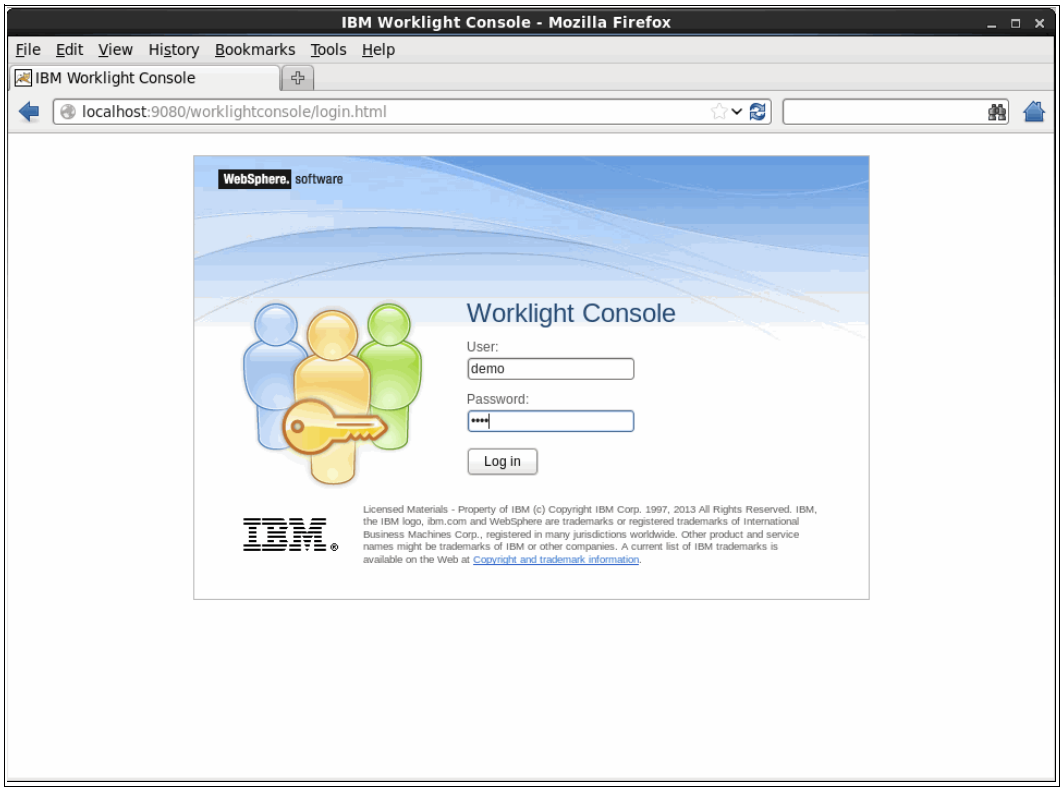


Figure 5-42 The Worklight Console login window

4. An empty workspace displays, as shown in Figure 5-43 on page 173. We need to import the remaining application files that were created in “Generate the installable assets” on page 160.

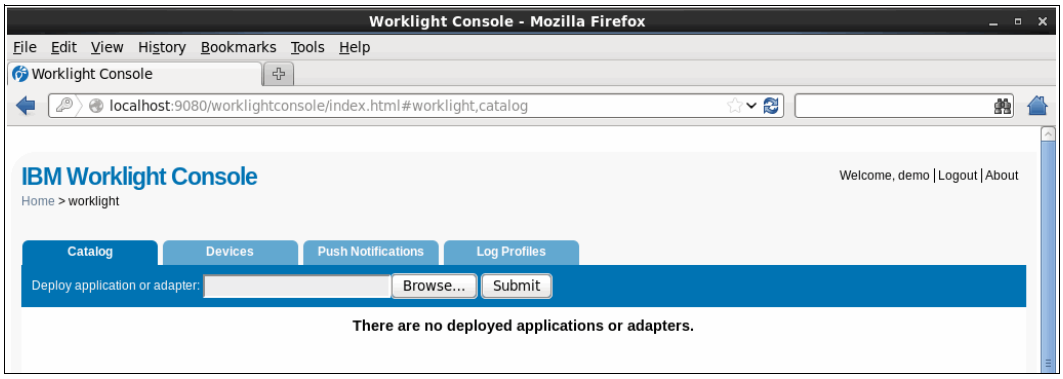


Figure 5-43 Installing application using the Worklight Console

- Click the **Browse** button and select the application file named `ClaimApp-all.wlapp` saved in step 6 on page 160 from the file system and click **Submit**. The application will be installed as shown in Figure 5-44.

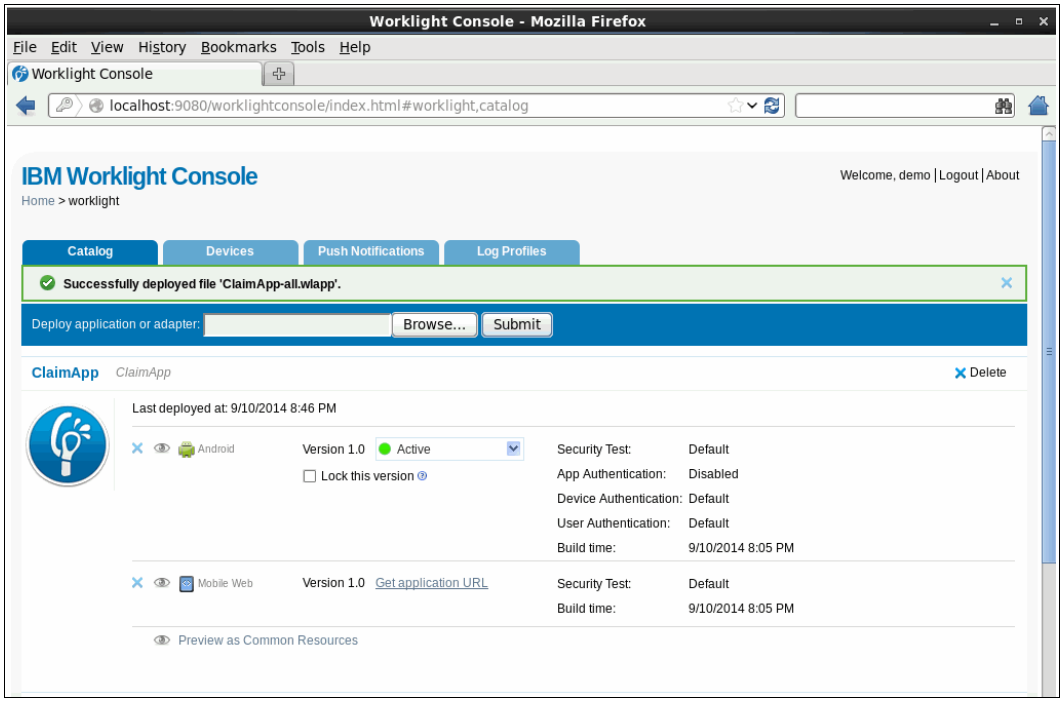


Figure 5-44 Application deployed

5. Repeat the sequence, selecting the `Backend.adapter` file saved in step 6 on page 160 and clicking **Submit**. The HTTP adapter of the application will be deployed and displays as shown in Figure 5-45 on page 174.

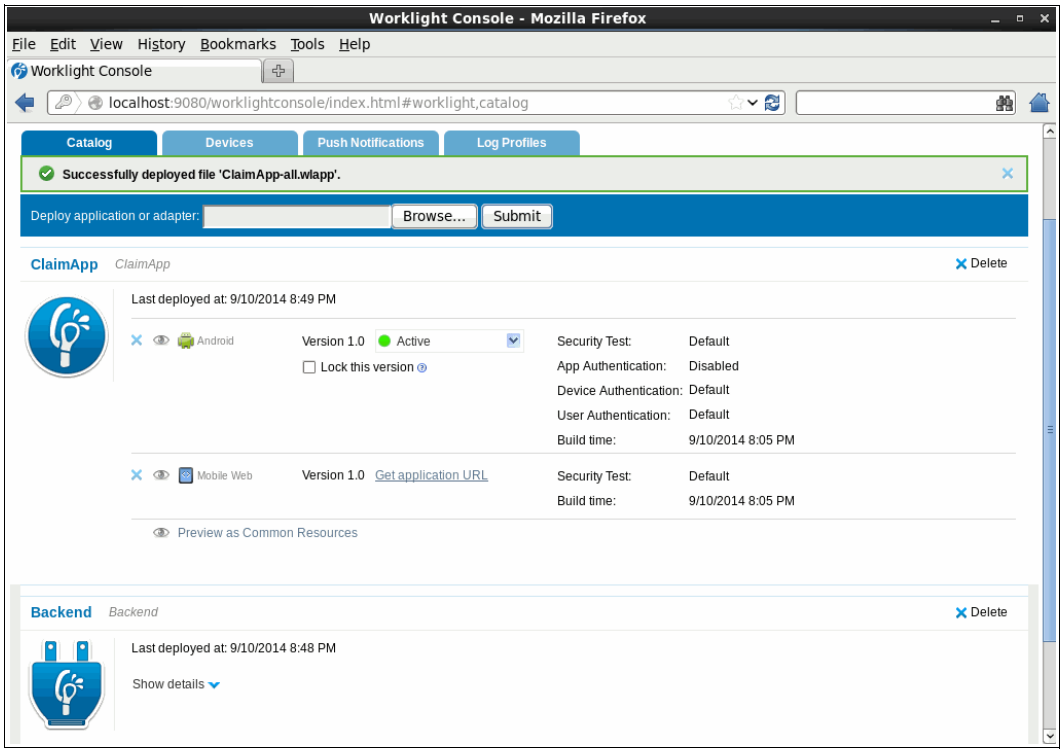


Figure 5-45 Application and adapter deployed

The application is now installed and ready to use. It can be accessed either using the **Get application URL** link for the mobile web environment, or from the Android installed application.

5.5.4 Updating the application

After the application is installed for the first time, it might require updates.

To update an installed application, get the latest .wlapp and .adapter files generated in “Generate the installable assets” on page 160 and deploy them using the Worklight console as described in “Add the mobile application” on page 172.

6

Rapid application deployment using IBM UrbanCode Deploy

This chapter describes how to use UrbanCode Deploy to prepare, build, and execute your deployment process, as it relates to the business scenario described in 1.2, “The business scenario” on page 14.

In this chapter, we discuss the following topics:

- 6.1, “IBM UrbanCode Deploy” on page 176
- 6.2, “Prepare the environment” on page 179
- 6.3, “Build and execute the deployment process” on page 190
- 6.4, “Deploy the application on Liberty” on page 198

© Copyright IBM Corp. 2014. All rights reserved.

6.1 IBM UrbanCode Deploy

Most software development projects encounter delays during packaging and deployment. Usually, software deployment is a concept that is sometimes obscured by jargon. The definition of deployment is the process of moving software through the various staging areas, to final production. The complexity of deployment also lends itself to human error, thus causing delays to the project.

IBM UrbanCode Deploy relieves you from the challenge of manual deployment by providing tools that improve the speed of the development process by the use of automation and improved reliability. The release automation tools in IBM UrbanCode Deploy provide visualization tools for n-tiered deployments, thus enabling you to model processes for large-scale deployments. There are also features for deploying to a specific environment before production. The drag-and-drop design tools help you to create processes by making it easy to visualize the entire scenario, end-to-end.

Architecture

The UrbanCode Deploy architecture consists of a service tier and a data tier (see Figure 6-1).

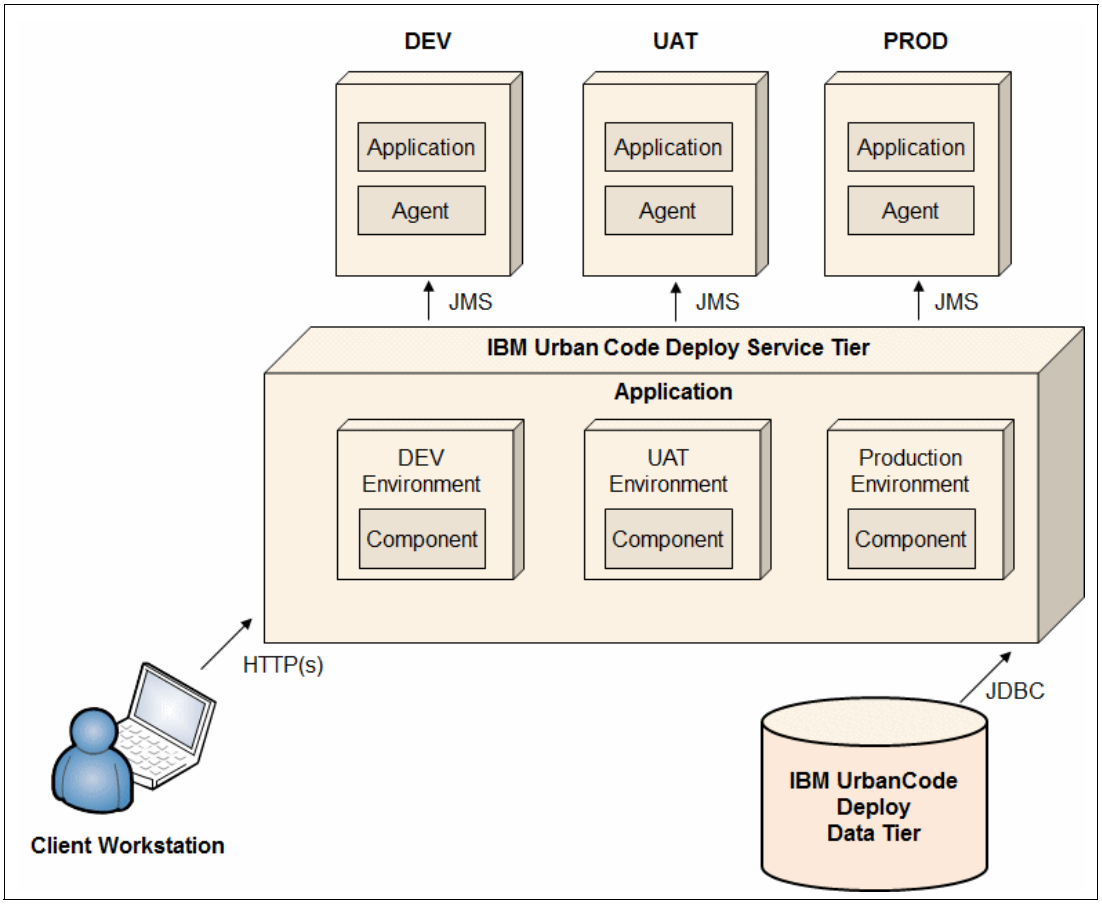


Figure 6-1 Deployment topology of UrbanCode Deploy

The service tier has a central server that provides a web server front-end and core services, such as workflow, agent management, deployment, inventory, security, and others. A service can be thought of as a self-contained mechanism for hosting a piece of business logic. Services can be consumed by clients, agents, or other services. Deployments are

orchestrated by the server and run by agents that are distributed throughout the network. Most clients use browsers to communicate with the web server using HTTP or HTTPS. Most server-agent communication is done with Java Message Service (JMS), but HTTP and HTTPS are also used, as required.

IBM UrbanCode Deploy uses stateless communications for server-agent communications (JMS-based) and client-web service communications. *Stateless*, as used here, means that the server retains little session information between requests, and each request contains all the information that is required to handle it. The server sets up listening sockets and listens for agents, relays, and users (clients). For added security, agents do not listen on ports. Agents send requests when they are ready to make the transition to a new state.

Server-agent communication is built around transferring, or deploying, components. Components can contain any business-meaningful content, such as environment information, configuration data, source, static files, or anything else that is associated with a software project. Because JMS connections are persistent and not based on a request-response protocol, IBM UrbanCode Deploy does not continually open and close ports. These persistent connections enable the server to communicate with agents at any time while the agents remain secure and scalable.

Many IBM UrbanCode Deploy services are representational state transfer (REST)-style. REST-style services are web services that focus on transferring resources over HTTP. A resource can be any business-meaningful piece of data. Resources are transferred by a self-describing format, such as XML or JavaScript Object Notation (JSON). The XML and JSON representations typically model resource states at the time of the agent or client requests. REST-style services achieve statelessness by ensuring that requests include all the data that is needed by the server to make a coherent response.

The data tier relational database stores configuration and runtime data, the data tier file store, CodeStation, contains log files, artifacts, and other non-structured data objects. Reporting tools can connect directly to the relational database.

6.1.1 Components

The deployable content in the system is defined as components in UrbanCode Deploy. Components can be binary files, static content, property files, or anything else that is associated with the software that UrbanCode Deploy delivers to the target destinations.

The deployable items in UrbanCode Deploy are called *artifacts*. These artifacts have versions that are then used to ensure that the appropriate component instances have been deployed.

Artifacts are derived from a number of systems in UrbanCode Deploy.

Components are deployed to resources (see 6.1.3, “Resources” on page 178) by agents (which are physical processes).

6.1.2 Applications

An application is defined as a mechanism that is initiated to deploy components. These applications organize the components in an orderly manner.

Application process

When you create an application, you identify the included components and define an application process.

Application processes, like component processes, are created with the process editor. UrbanCode Deploy provides several common process steps. Otherwise, application processes are assembled from processes that are defined for their associated components.

Application processes can run manually, automatically on some trigger condition, or on a user-defined schedule. When a component has several processes, the application determines which ones are run and in which order. For instance, an n-tiered application might have a web tier, a middleware tier, and a database tier. And, continuing the example, the database tier must be updated before the other two, which are then deployed concurrently. An application can orchestrate the entire process, including putting servers online and offline for load-balancing, as required.

When an application process runs, it interacts with a specific environment. An environment is a collection of one or more resources. At least one environment must be associated with the application before the process can run. Application processes are independent of environments. Processes can be designed independently of any particular environment. These application processes enable a single application to interact with separate environments, such as QA, or production. To use the same application process with multiple environments (a typical scenario), associate each environment with the application and run the process separately for each one.

In addition to deployments, several other common processes are available, such as rolling-back deployments. UrbanCode Deploy tracks the history of each component version, which enables application processes to restore environments to any point.

Environments

An environment is a user-defined collection of resources that host an application. Environments are modeled on some stage of the software project lifecycle.

There can be different topologies for environments. For example, an environment can consist of a single server, or it can be spread over several servers, or spread over clusters of servers. Environments are scoped to applications. Although multi-tenant servers can be the target of multiple applications, experience shows that most IT organizations use application-specific environments.

UrbanCode Deploy maintains an inventory of every artifact that is deployed to each environment and tracks the differences between them.

6.1.3 Resources

A resource is a user-defined construct that is based on the architectural model of UrbanCode Deploy.

Resources aid bookkeeping. Inventory is tracked for resources. Resources are created and managed through the user interface.

A resource represents a deployment target, such as a physical server, virtual machine, database, or Java Platform, Enterprise Edition container. Components are deployed to resources by agents. Resources generally are hosted on the same host where its managing agent runs. A host can have more than one resource. If an agent is configured to handle multiple resources, a separate agent process runs for each one.

In addition, a resource can represent a process that is distributed over several physical or virtual machines. Finally, environments consist of resources.

Agents

An agent is a process that runs on the target host and communicates with the UrbanCode Deploy server.

Agents are integral to the client/server architecture of UrbanCode Deploy. Agents do the actual work of deploying components and so relieve the server from the task, making large deployments that involve thousands of targets possible.

Typically, an agent runs on the same host where the resources it handles are located. A single agent can handle all resources on its host. If a host has several resources, an agent process runs separately for each resource.

Depending on the number of hosts in an environment, a deployment might require many agents. Agents are unobtrusive and secure. Agent communications use Secure Shell (SSL) encryption and mutual key-based authentication. For added security, agents do not listen to ports, but open direct connections to the server instead.

6.2 Prepare the environment

In this section, we discuss how an Enterprise Application Resource (EAR) file is deployed in WebSphere Application Server.

To prepare the deployment environment for UrbanCode Deploy, install the UrbanCode Deploy server and at least one agent. To learn more about installing UrbanCode Deploy, visit the IBM Knowledge Center:

http://www-01.ibm.com/support/knowledgecenter/SS4GSP_6.0.0/com.ibm.udeploy.install.doc/topics/install_ch.html

6.2.1 Create a deployment process

To create your deployment process, you need a component to run, as described in 6.1.1, “Components” on page 177.

Components have component processes. See Figure 6-2 on page 180. A component process is a series of user-defined steps that operate on the component or its artifacts. Each component has at least one process and can have several. A component process can be as simple as a single step or contain numerous relationships, branches, and process switches. Component processes are created with the process editor. The process editor is a visual drag-and-drop editor that lets you drag process steps onto the design space and configure them as you go. As more steps are placed, you visually define their relationships with one another. Process steps are selected from a menu of standardized steps. UrbanCode Deploy provides steps for several utility processes, such as inventory management and workflow control. More process steps are provided by plug-ins. A component process can have steps from more than one plug-in.

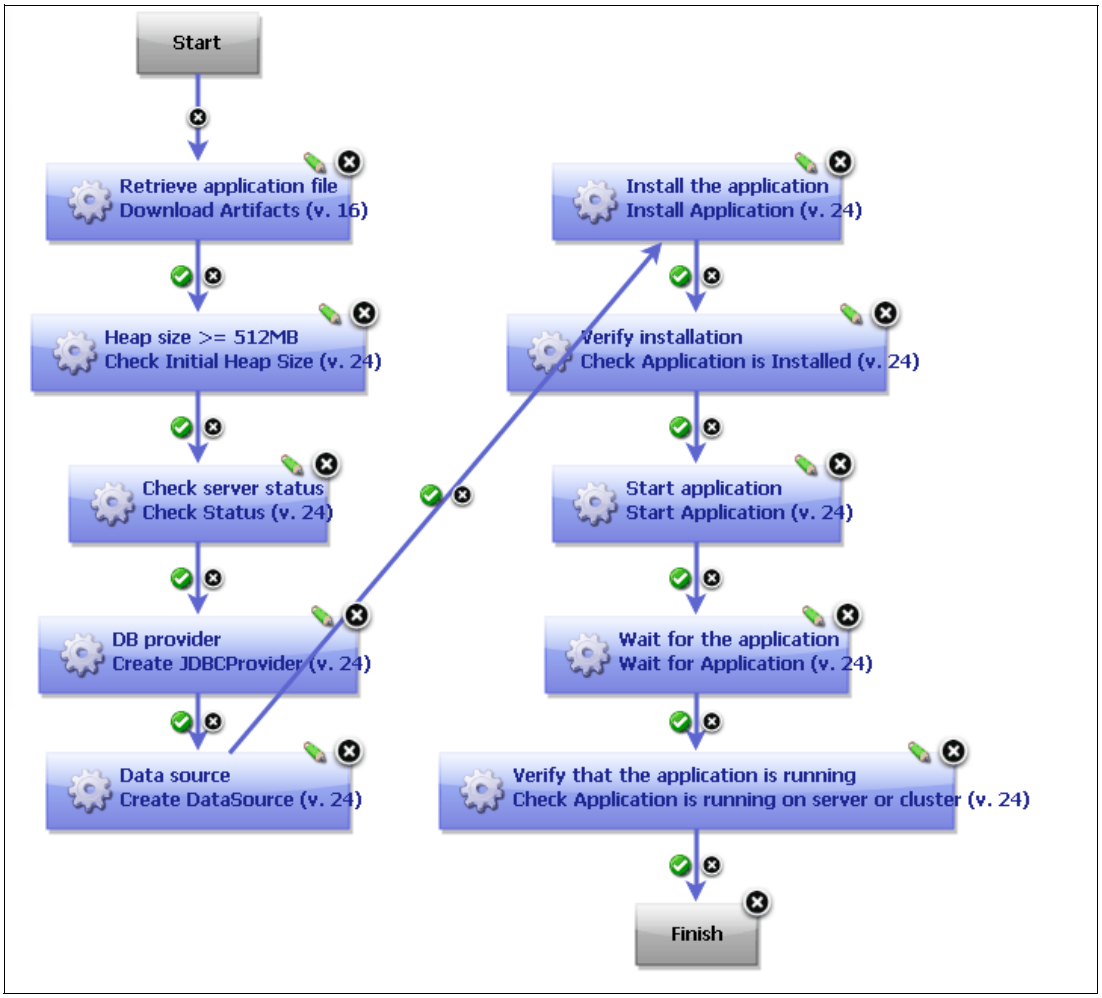


Figure 6-2 Sample of an UrbanCode Deploy deployment process

Note: In this chapter, we use the EAR file derived from the solution in Chapter 5, “Creating solution artifacts” on page 113.

Install the agent

To install an agent:

1. Navigate to **Resources** → **Agents**.
2. Click **Install New Agent**, and complete the required fields (see Figure 6-3 on page 181).

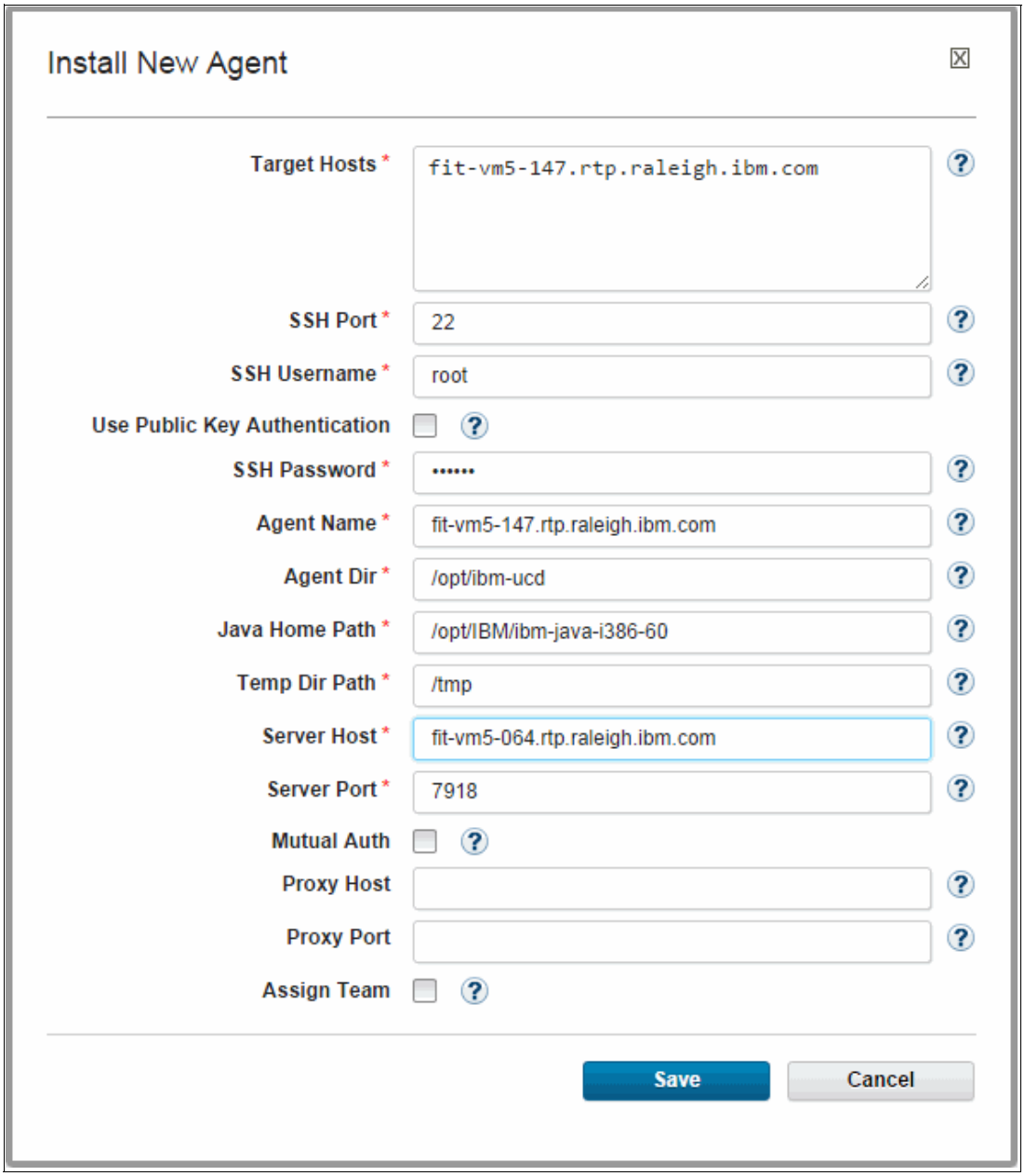


Figure 6-3 Installing the UrbanCode Deploy agent

Table 6-1 defines the required fields for those listed in Figure 6-3.

Table 6-1 Install agent fields

Field name	Description
Target Hosts	Host name or IP address of the agent machine where UrbanCode Deploy will connect
SSH port	SSH port of the agent machine
SSH username	SSH username of the agent machine

Field name	Description
SSH password	SSH password of the agent machine
Agent name	Name of the agent
Agent Dir	Target directory in the agent machine where the agent binaries will be installed
Java Home Path	JAVA_HOME directory of the agent machine
Temp Dir Path	Temporary directory path where the installation binaries will be placed
Server host	Host name or IP address of the UrbanCode Deploy server
Server port	Port where UrbanCode Deploy listens

3. Start the agent manually. The binaries are in the bin folder of the directory specified in the Agent Dir field (see in Table 6-1 on page 181).

   Enter the code in Example 6-1 to start the UrbanCode Deploy agent.

   Example 6-1 Start the UrbanCode Deploy agent

```
[root@fit-vm5-147 bin]# ./ibm-ucdagent start
```

   The agent displays on the agents page of UrbanCode Deploy (see Figure 6-4).

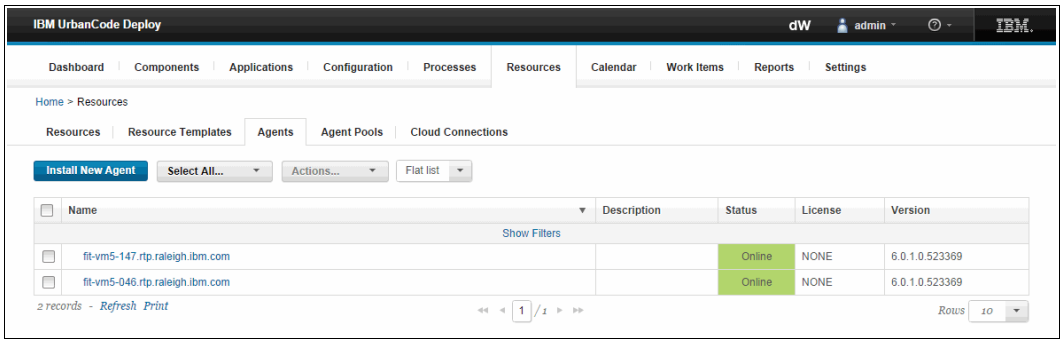


Figure 6-4 UrbanCode Deploy installed agents

Create a component for deployment configuration

After creating an agent, create a component that uses the Middleware Configuration for WebSphere Template. This procedure gathers the configuration of your environment and generates a template that can be used in multiple environments:

1. On the **Components** tab, click **Create New Component**. In this example, we use **MCWASTemplate** as the name for our template.
2. In the template field, select **Middleware Configuration for WebSphere**.
3. Click **Save**.

Figure 6-5 shows the completed window.

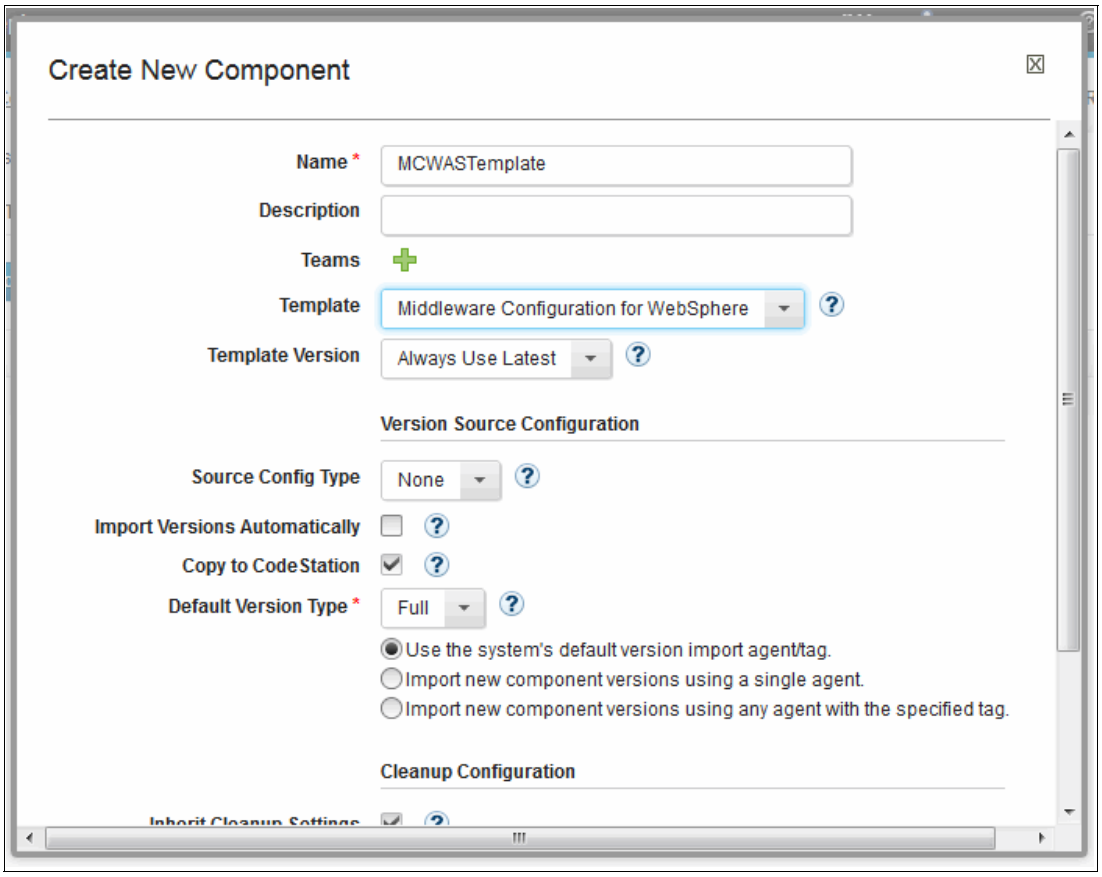


Figure 6-5 The MCWASTemplate has been created

Set a cluster variable token

For clustered deployments, follow these steps to set a cluster variable token:

1. Navigate to **Configuration → Environment Property Definitions**.
2. Add an environment property named `websphere.ClusterVariableToken`.
3. (*Optional*) Set a default value.

The result is shown in Figure 6-6 on page 184.

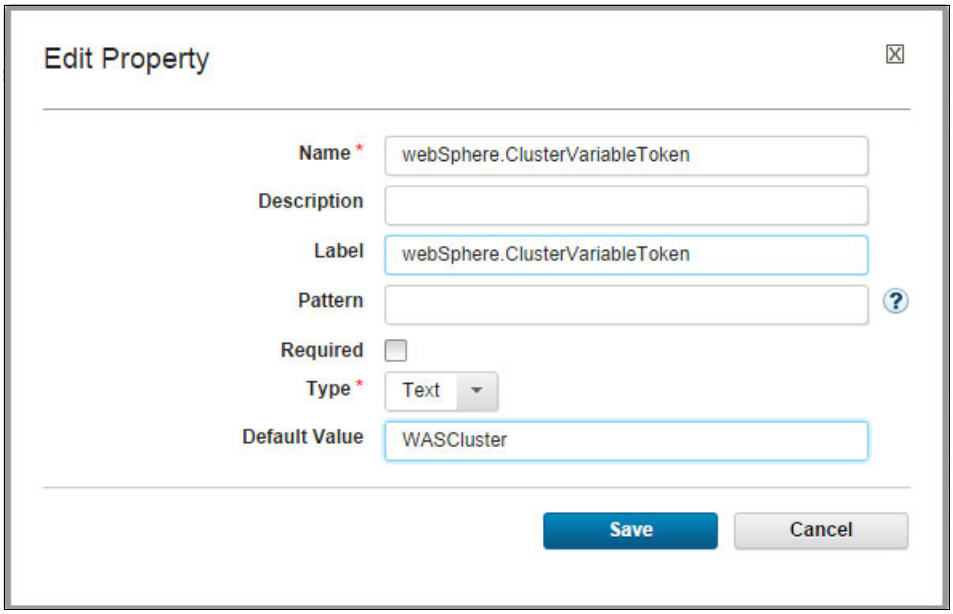


Figure 6-6 Setting the Cluster Variable Token

Create a top-level group

With the template created, map the component to the resource:

1. On the Resources tab, click **Create Top-Level Group**.
2. Enter the required fields, then click **Save** (see Figure 6-7).

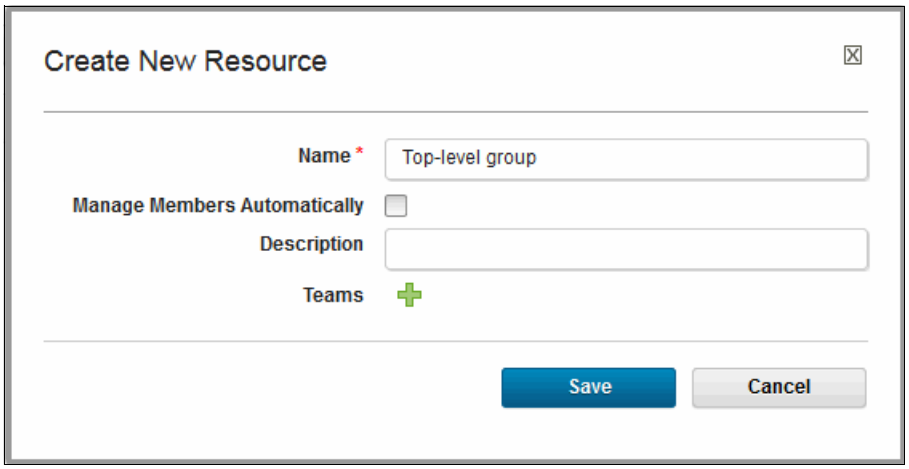


Figure 6-7 Creating a top-level group resource

3. An agent needs to be mapped to the top-level group. Hover on the top-level group, and click **Actions**.
4. Click **Add Agent** to add the agent to the group (see Figure 6-8 on page 185).

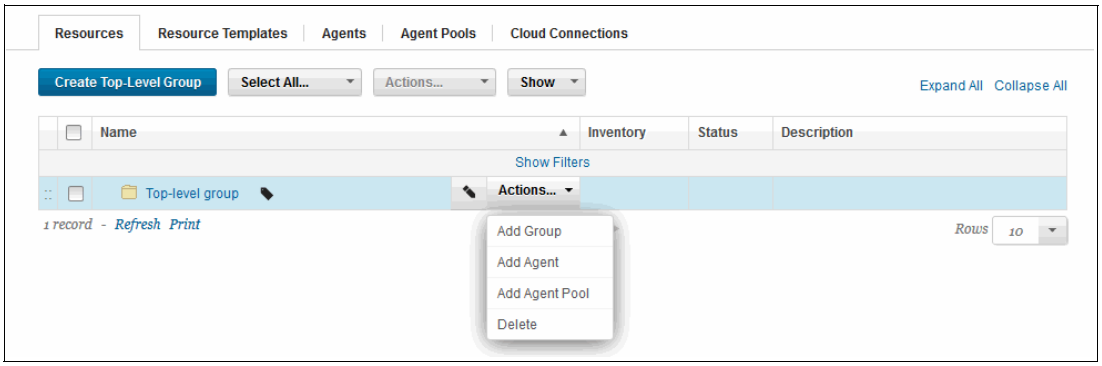


Figure 6-8 Adding an agent to the top-level group

5. Select the agent that you installed in “Install the agent” on page 180, then click **Save** (see Figure 6-9).

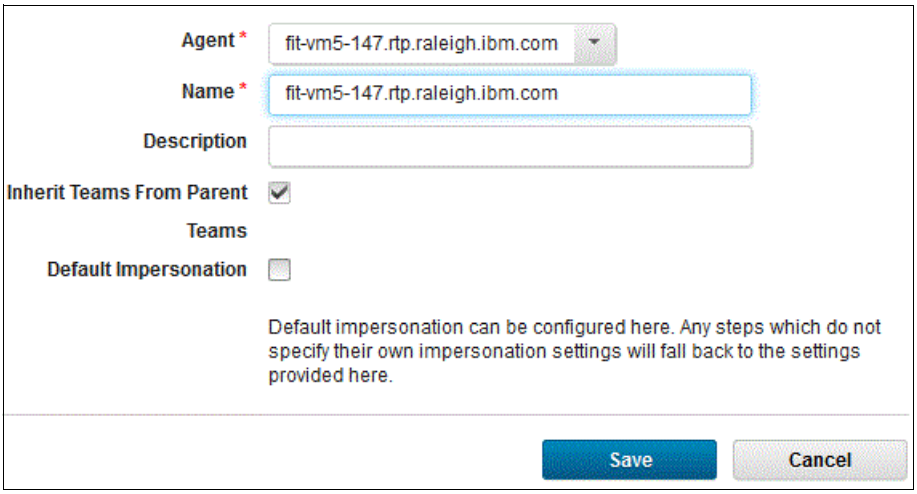


Figure 6-9 An agent has been added to the top-level group

6. Click **Save**, and the window displays that the agent has been added to the top-level group. The cell will be added automatically (see Figure 6-10).

> **Note:** After you added the agent, the agent will automatically discover the cell name of your WebSphere environment. The cell discovery takes some time, so click **Refresh** from time to time. Wait for the cell to be discovered before proceeding to the next step.

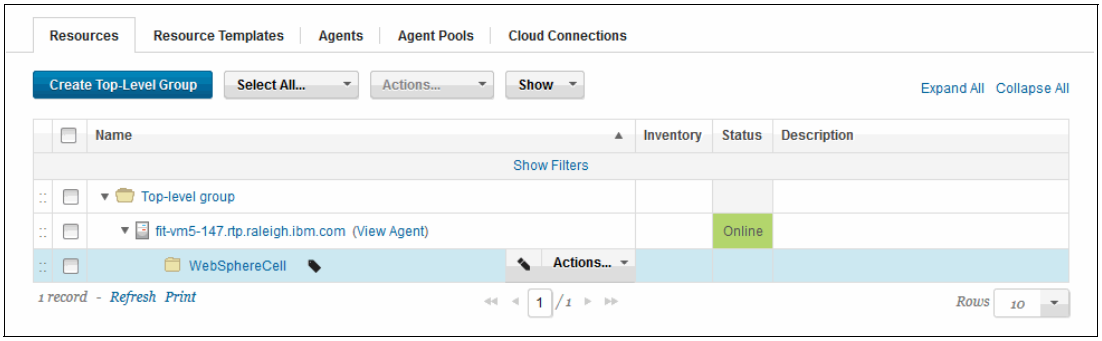


Figure 6-10 The agent discovers the cell

7. Navigate to the cell by clicking the cell name, then click the **Configuration** tab. Verify that the settings are correct. Figure 6-11 shows our example.

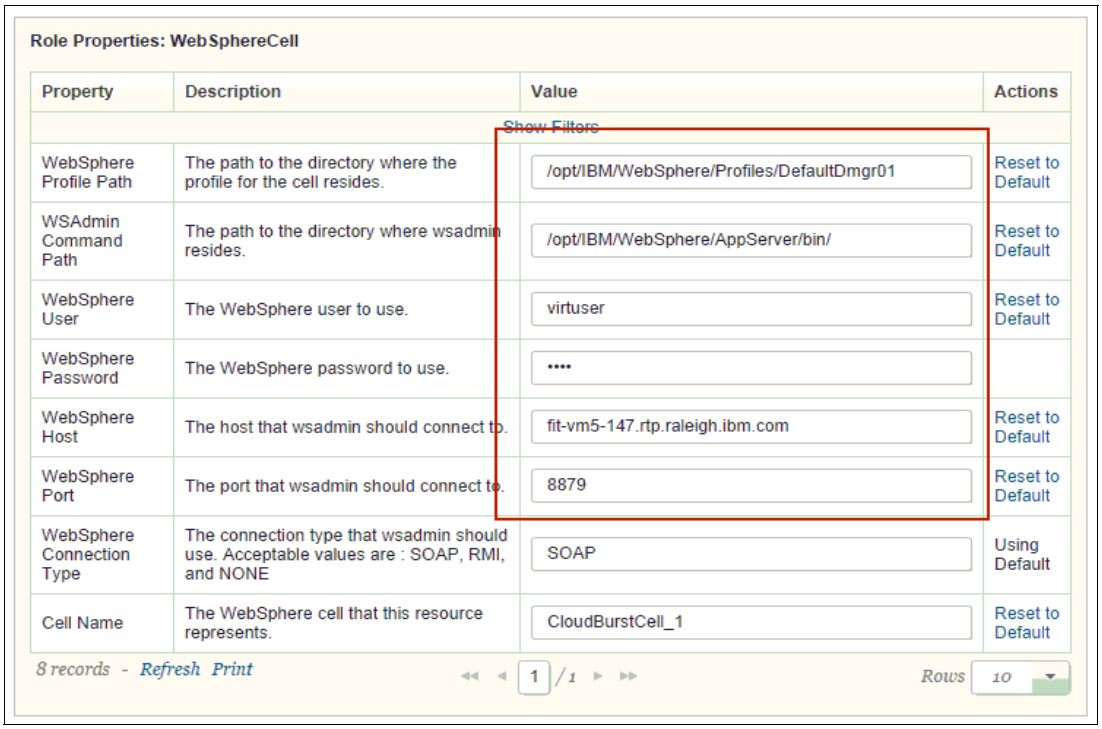


Figure 6-11 Basic configuration of a WebSphere cell in UrbanCode Deploy

8. Once the cell has been discovered, hover over the cell, and an Action button displays. Click **Actions** → **Auto Configure**. Select **WebSphere Topology Discovery** (see Figure 6-12).

> **Note:** UrbanCode Deploy automatically discovers the WebSphere node, cell, and cluster configuration using Auto Configure. The discovery takes some time, so click **Refresh** from time to time. Wait for the cell to be discovered before proceeding to the next step.

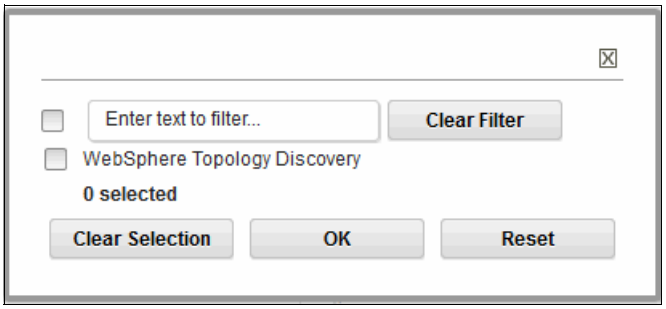


Figure 6-12 Configure the WebSphere Topology Discovery cell

After discovery, you will notice that the cell is populated.

9. Hover over the cluster, then click **Add Component**. Use the component name that was added in “Create a component for deployment configuration” on page 182. Figure 6-13 on page 187 shows an example of how the component is mapped inside a cluster.

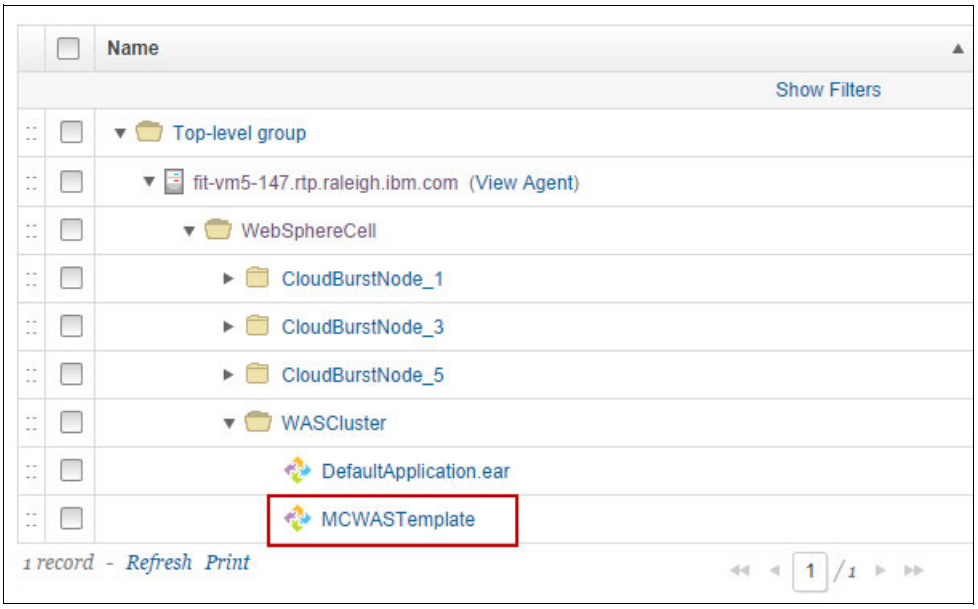


Figure 6-13 Mapped component inside a WebSphere Application Server cluster

Create an application

We now create an application to generate the deployment configuration. To do so, navigate to the Applications tab, then click **Create New Application**.

Map the component to the application

To map the component to the application:

1. Navigate to the Components tab, then click **Add Component**.
2. Select the component name that was created in “Create a component for deployment configuration” on page 182 (see Figure 6-14).

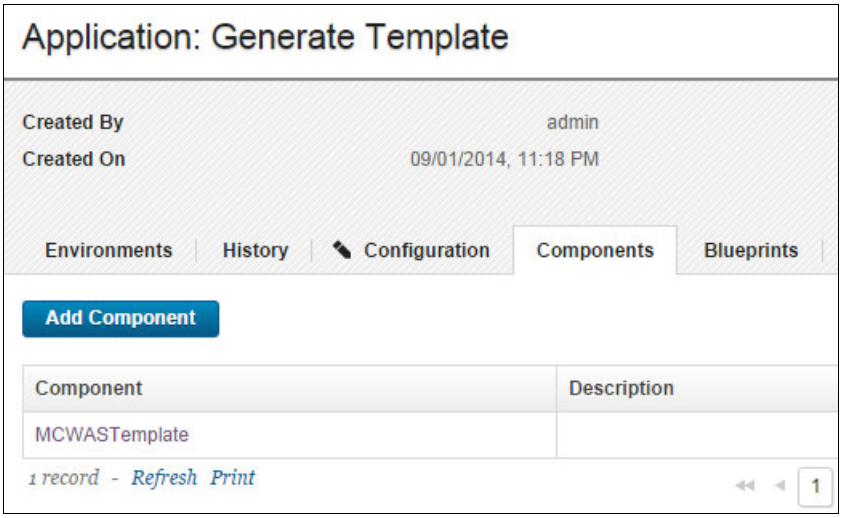


Figure 6-14 The newly added component has been mapped to the application

Create the environment

Create the environment for deploying the application:

1. On the **Environments** tab, click **Create New Environment**. This might represent your production, test, or development environment.
2. Enter a name for the environment (see Figure 6-15).

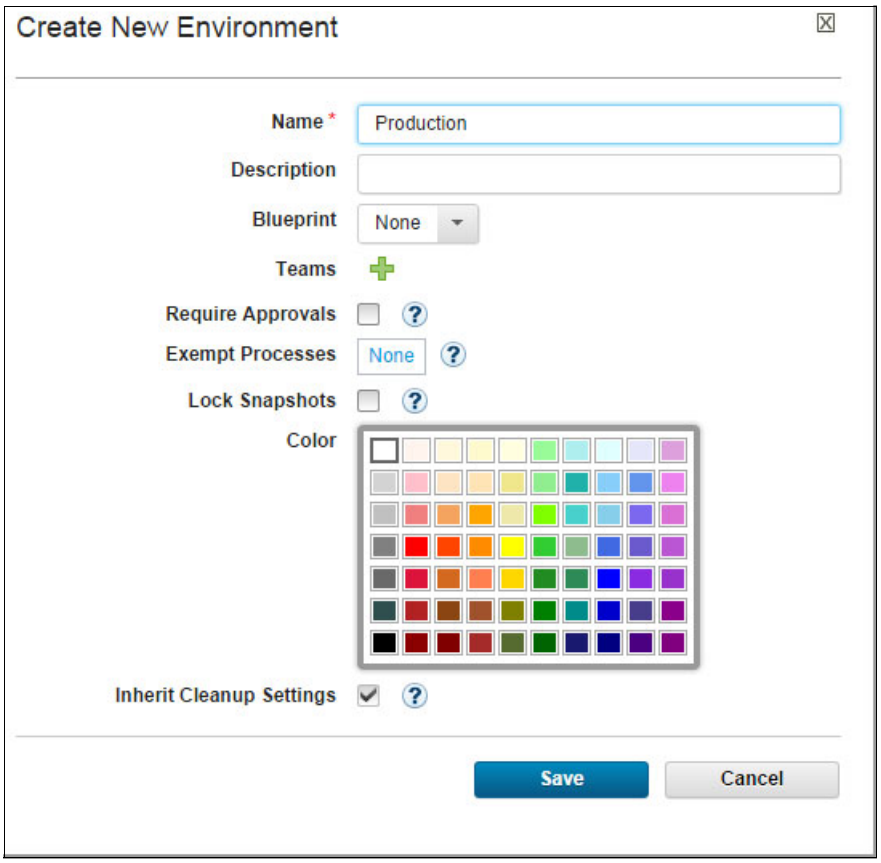


Figure 6-15 The Create New Environment panel

3. Click **Save**.

Map the resource to the environment

To map the resource, follow these steps:

1. Select the environment created in “Create the environment” on page 188.
2. On the **Resources** tab, click **Add Base Resources**.
3. Select the name of the component that was created in “Create a component for deployment configuration” on page 182 (see Figure 6-16 on page 189).

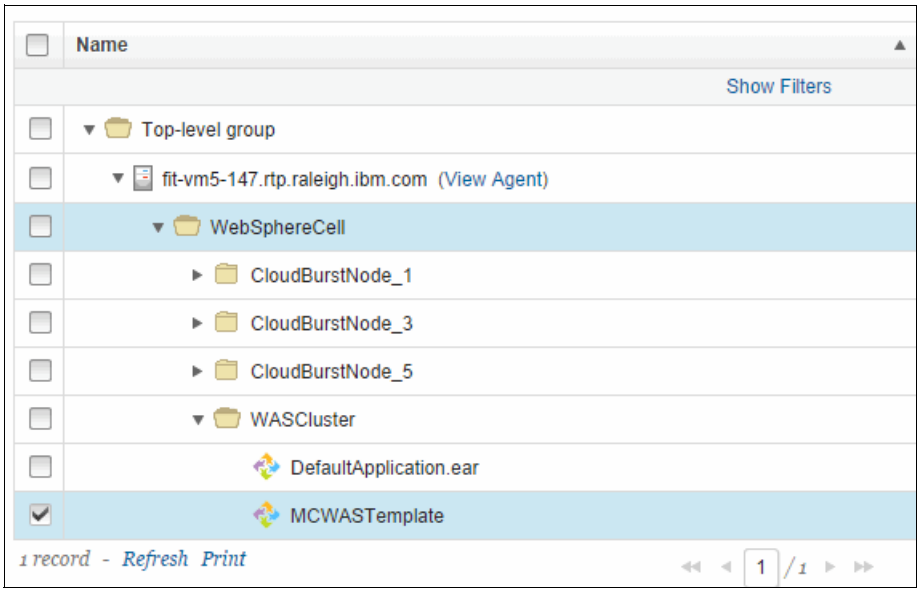


Figure 6-16 Mapping the component to the environment

Run the application to gather deployment configuration information

Deploy the application and gather information about the deployment configuration:

1. Navigate to **Applications** → **Environments** tab.
2. Request the process by clicking the **play** button (see Figure 6-17).

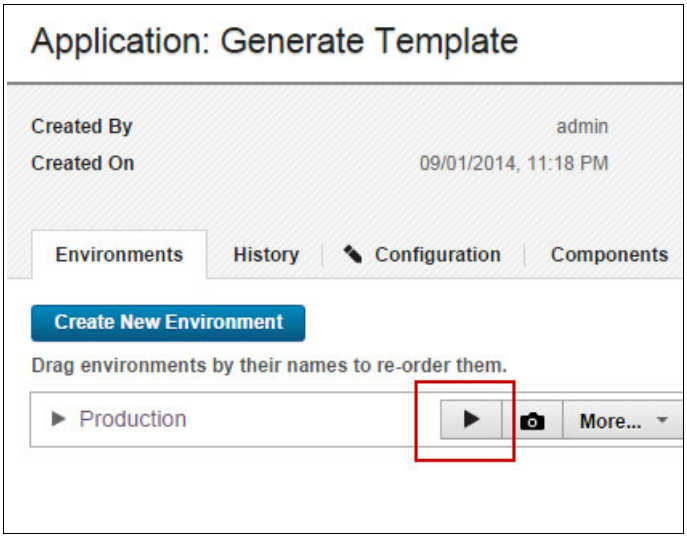


Figure 6-17 Gathering deployment configuration information

3. Accept the default values, then click **Submit** (see Figure 6-18 on page 190).

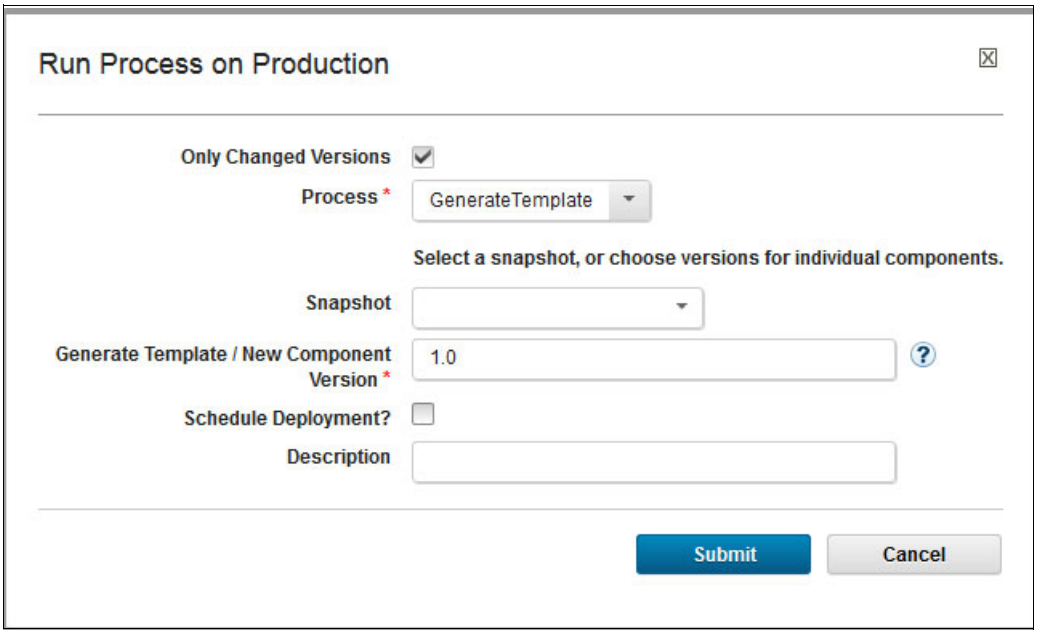


Figure 6-18 Running the process in your environment

6.3 Build and execute the deployment process

We now create the process to deploy our application. We use the generated template from 6.2.1, “Create a deployment process” on page 179 for our deployment environment.

6.3.1 Create a new component

Create a new component that contains your EAR or Web Application Resource (WAR) file. The Base Path field (see Figure 6-19 on page 191) represents the directory where the artifacts are located.

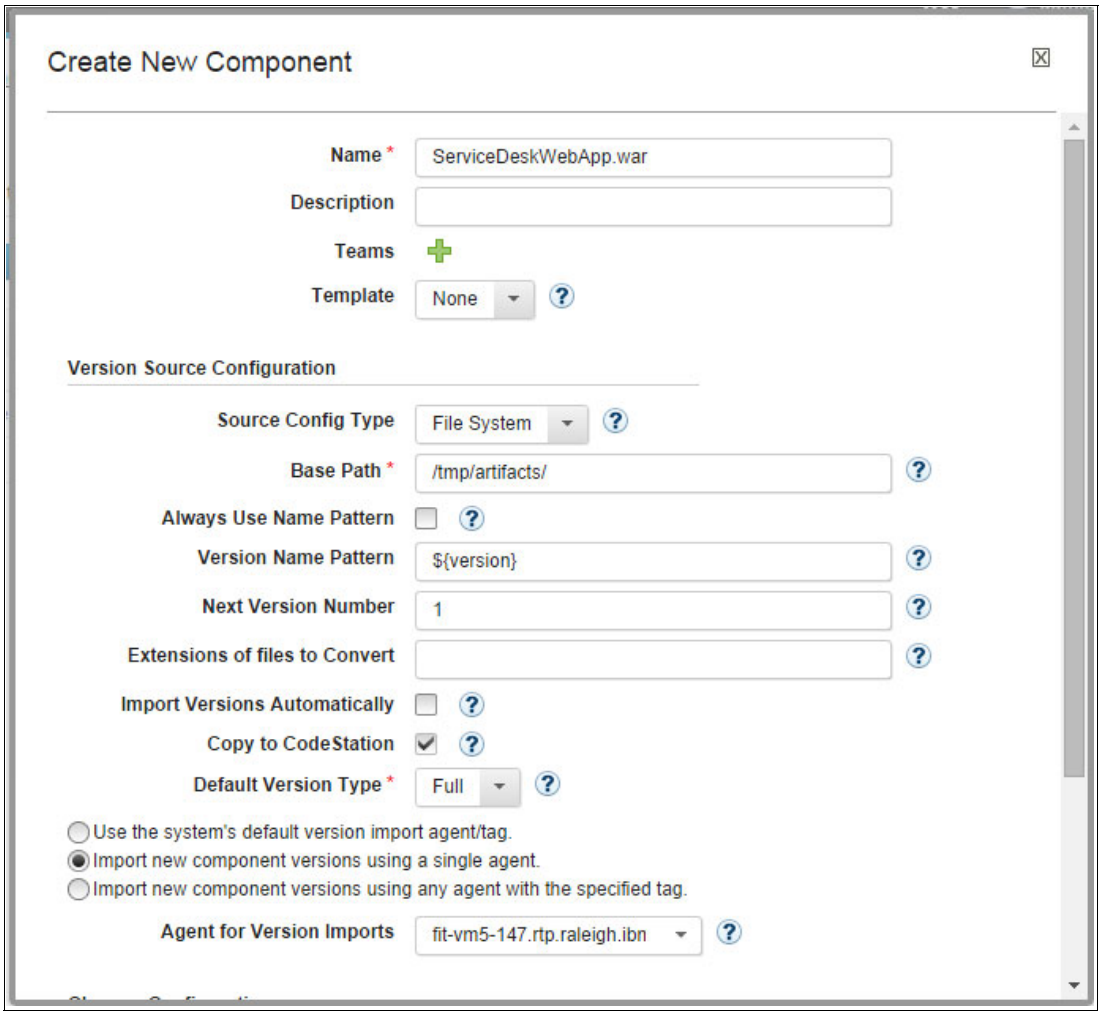


Figure 6-19 Creating a new component for deployment

Create the new component:

1. Select the **Import new component** versions using a single agent option, then select your agent that was created in “Install the agent” on page 180.
2. Import a new version by navigating to the **Versions** tab, then clicking **Import New Versions**.
3. Enter the **Version Name**, then click **Save** (see Figure 6-20).

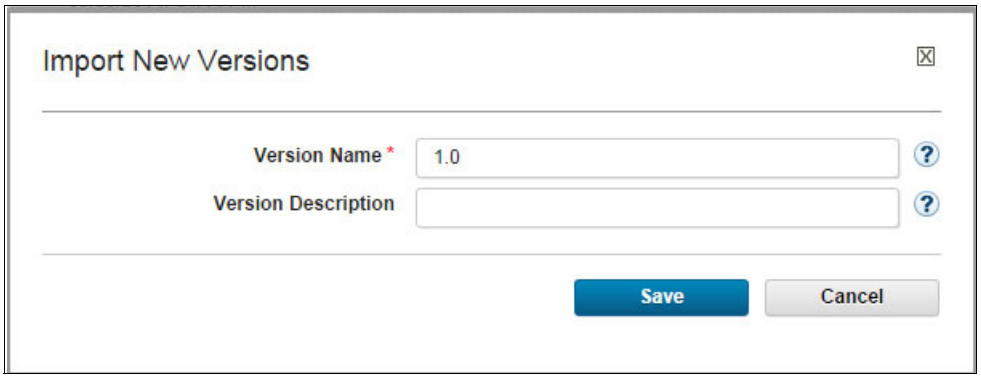


Figure 6-20 Importing a new version

4. With the import of the version complete, create a new process by navigating to the **Processes** tab, then click **Create New Process.**
5. Accept the default values, then click **Save**.
6. Click the newly created process to start orchestrating your process:
   a. Download the artifacts from the repository. In the plug-ins panel, expand **Repositories** → **Artifact** → **IBM UrbanCode Deploy**.
   b. Drag **Download Artifacts** to your canvas.
   c. Accept the default values, then click **Finish** (see Figure 6-21).

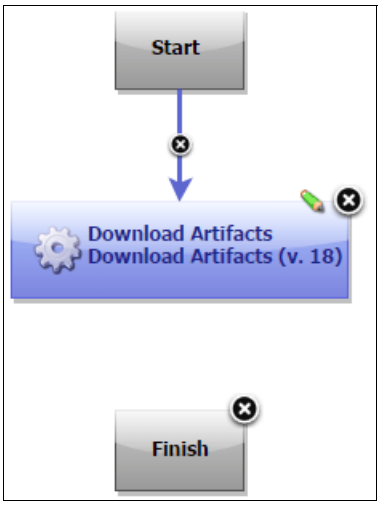


Figure 6-21 The UrbanCode Deploy process canvas with download artifacts as the first step

   d. In the plug-ins panel, expand **Application Server** → **Java** → **WebSphere** → **Deployment**.
   e. Drag **Install or Update Application** to your canvas.
   f. Complete the required fields (see Figure 6-22 on page 193). Table 6-2 on page 193 describes the values for installing your application to WebSphere.

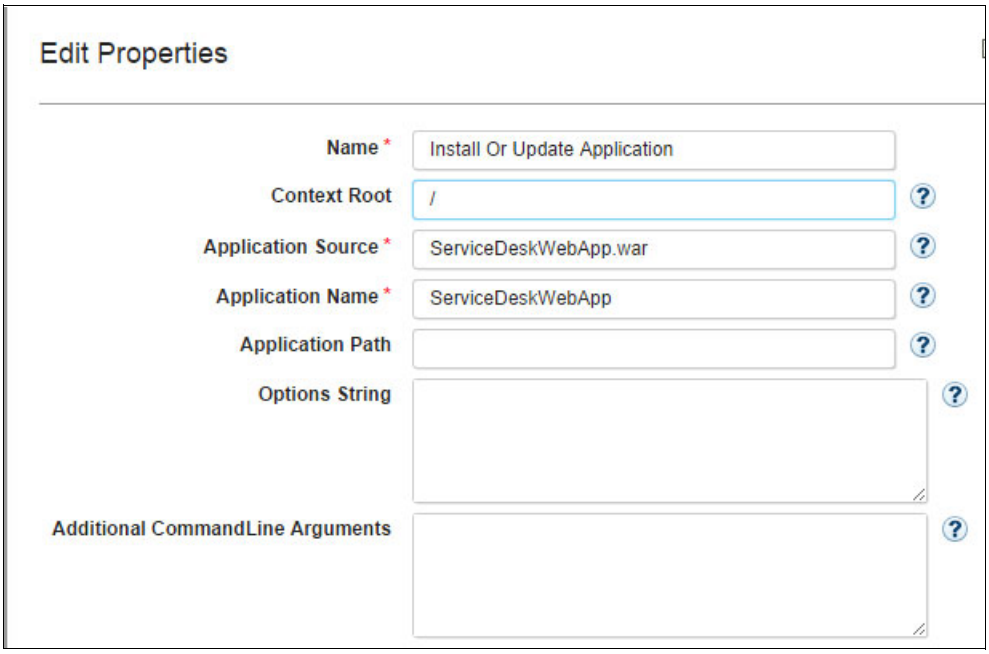


Figure 6-22 Properties of the install or update application node

Table 6-2 Required fields for installing or updating the application

Name	Description	Value
Name	The name of the node (provides a default value)	Install or Update Application
Application source	The location of the application to install	`ServiceDeskWebApp.war`
Application name	The name of the application being installed	`ServiceDeskWebApp`

g. Click **Save**.

h. Connect all nodes (see Figure 6-23 on page 194).

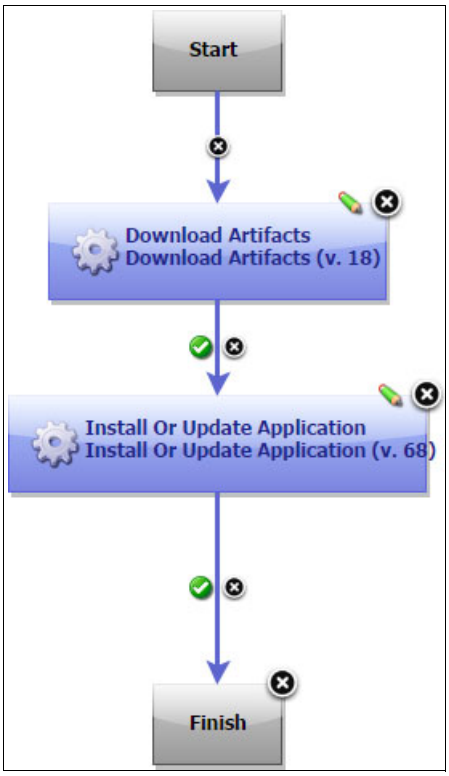


Figure 6-23 Connecting the nodes

i. Click **Save**.

j. Navigate to the Resources tab, then expand the top-level group that was specified in “Create a top-level group” on page 184.

k. Add the newly created component to the cluster (see Figure 6-24).

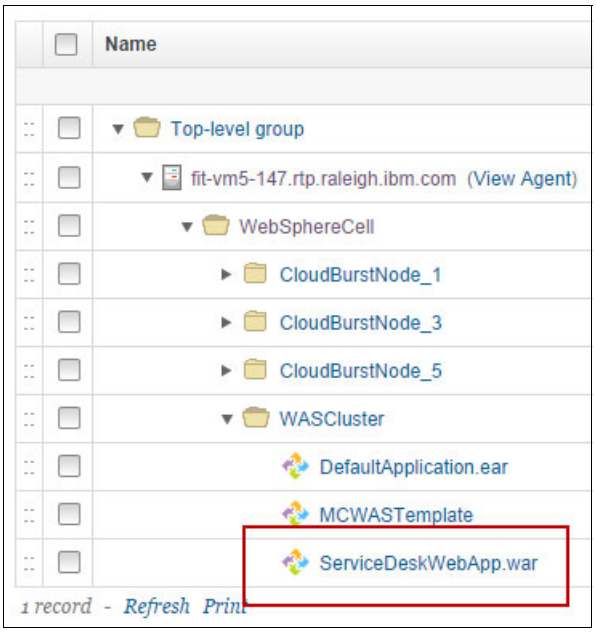


Figure 6-24 Resources containing the newly created application

We will use this resource for our application, as described in 6.3.2, “Create an application for deployment” on page 195.

6.3.2 Create an application for deployment

To create an application for deployment:

1. Create a separate application, based on the steps in “Create an application” on page 187.
2. Create an environment for deployment, which is based from the steps on “Create the environment” on page 188:
   a. In the components section, map the following according to the steps from “Map the component to the application” on page 187:
      - MCWASTemplate
      - ServiceDeskWebApp.war
3. In the environments section, map the following resources, based from the steps in “Map the resource to the environment” on page 188:
   – MCWASTemplate
   – ServiceDeskWebApp.war

   The completed window is shown in Figure 6-25.

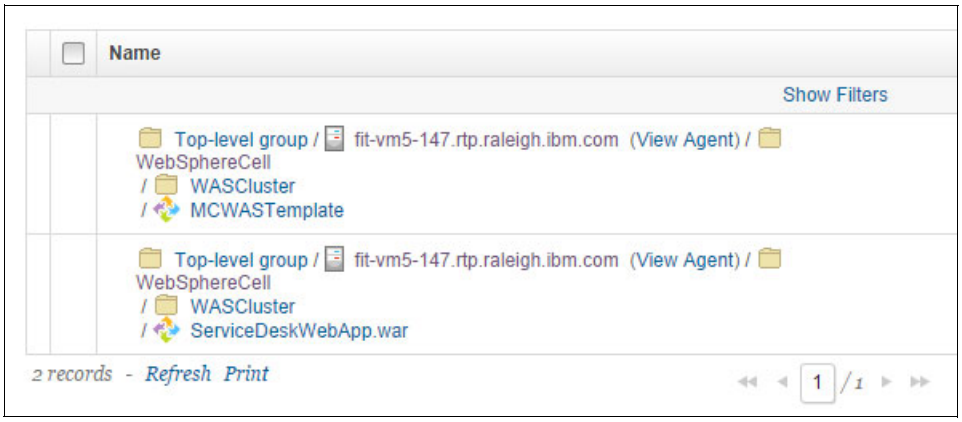


Figure 6-25 Mapped base resources in an environment

4. Navigate to the **Applications** → **Processes** tab, then click **Create New Process**.
5. Enter a name for your process.
6. Click **Save** and you can begin working on your process.
7. Drag two of the install components (in our example, `Install MCWASTemplate` and `InstallServiceDeskWebApp.war`) to the canvas.
8. Populate the fields according to Table 6-3 and Table 6-4 on page 196. Figure 6-26 on page 196 shows the application deployment process.

Note: For any values that are not mentioned in section 6.3.2, “Create an application for deployment” on page 195, use the default values.

Table 6-3 Install the component for the MCWASTemplate

Name	Description	Value
Component	Name of the component	MCWASTemplate

Name	Description	Value
Component Process	The component process used to run	Configure WebSphere Application Server (Template)

Table 6-4 Install the component for the ServiceDeskWebApp.ear file

Name	Description	Value
Component	Name of the component	ServiceDeskWebApp.ear

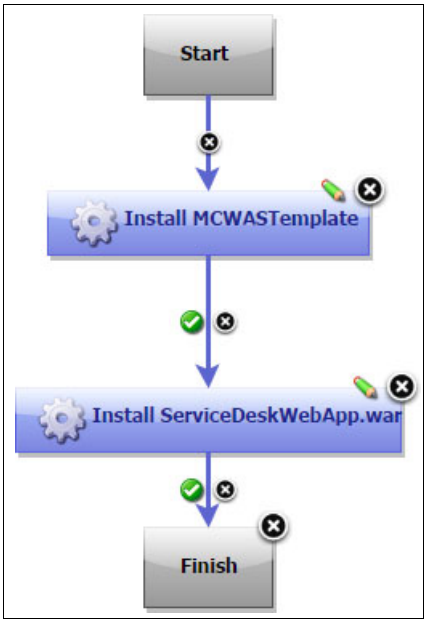


Figure 6-26 Application deployment process

9. Save the configuration.
10. Navigate back to your application from the Applications tab, then click **Request Process** on your environment, as shown in Figure 6-17 on page 189.
11. Click the **Choose Versions** link to select the version to deploy.
12. Click **Add** to add the version of your template and deployment component.
13. Click **OK**. Then click **Submit** to start the deployment process (see Figure 6-27 on page 197).

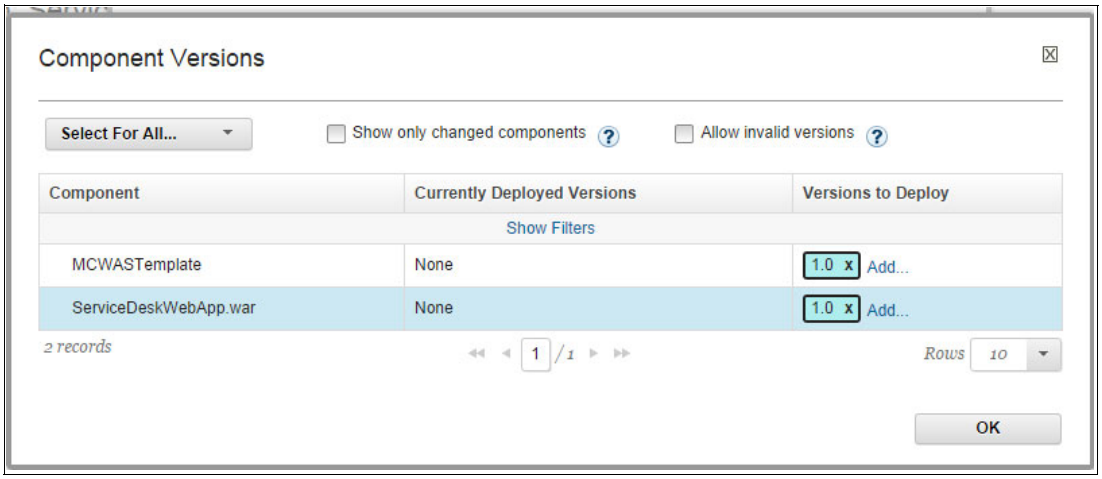


Figure 6-27 Selecting component versions

Tip: For future deployments, you need to increment the version of the deployment component. Otherwise, UrbanCode Deploy recognizes the same version and ignores the deployment.

Figure 6-28 on page 198 shows an example of a successful deployment.

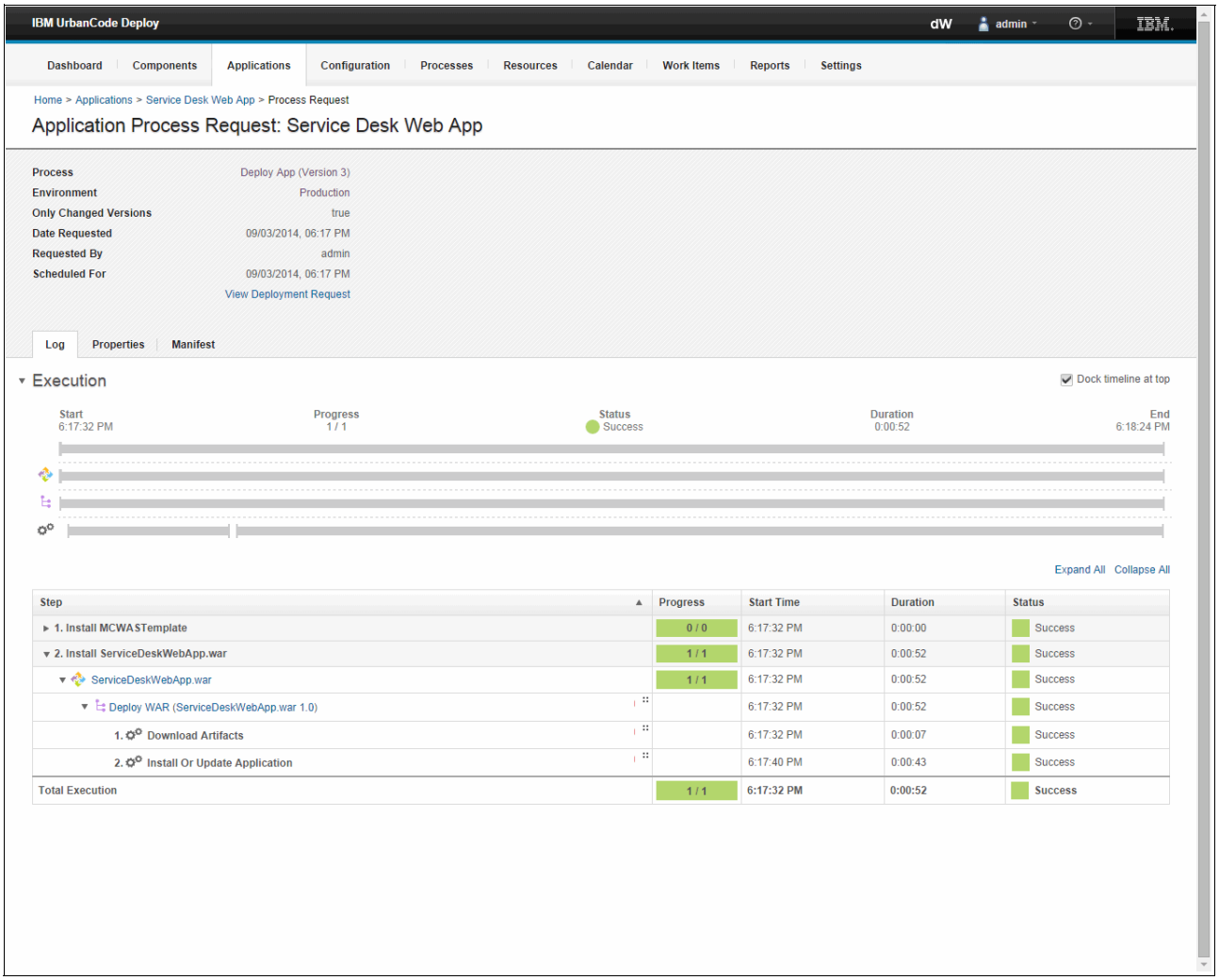


Figure 6-28 Successful deployment of a WAR file in UrbanCode Deploy

6.4 Deploy the application on Liberty

In this section, we are creating the instructions to deploy the WAR file from the WebApplication using Liberty. We begin by creating a component that defines the artifacts that you want to deploy. A component represents the smallest functional unit of the application, for example, WAR, static files, and pictures. For more information about components, see 6.1.1, “Components” on page 177.

6.4.1 Create a component

For component creation, follow these steps:

1. Click the **Components** tab, then **Create New Component**.
2. In the **Name** field, enter **Liberty**,
3. In the **Source Config Type** field, select **File System**.
4. In the **Base Path** field, enter **/tmp/artifact**.

5. Choose the option to **Import new component version using a single agent**. This enables the **Agent for Version imports** field, in which you need to select the server where the files will be stored.
6. Click **Save**.

Figure 6-29 shows the result.

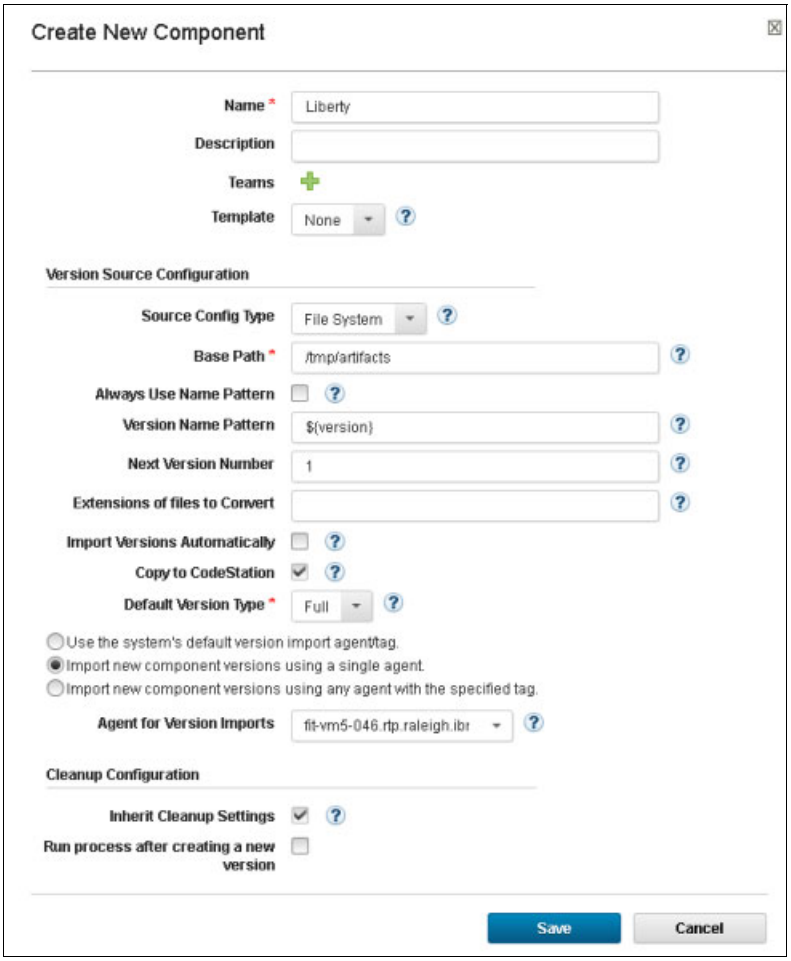


Figure 6-29 Creating a component

Create a component version

A component version contains the files that we want to deploy, consisting of artifacts, such as a WAR file and the server.xml files. When we create or update an artifact, such as a war file or a server.xml file, we need to import these artifacts into UrbanCode. To do so, we need to define a new version name, which is arbitrary and can be defined by you, as shown in Figure 6-30 on page 200.

> **Note:** For this component, we are using the file system as the Source Config type. In this case, we must copy all of the artifacts for the path specified in the field Base Path. Here we are using the base path, `/tmp/artifacts`.

To create the component version, follow these steps:

1. Click the **Components** tab.

2. Choose the **Liberty Component** that was created in 6.4.1, “Create a component” on page 198.
3. Click the **Versions** tab.

Figure 6-30 shows the window.

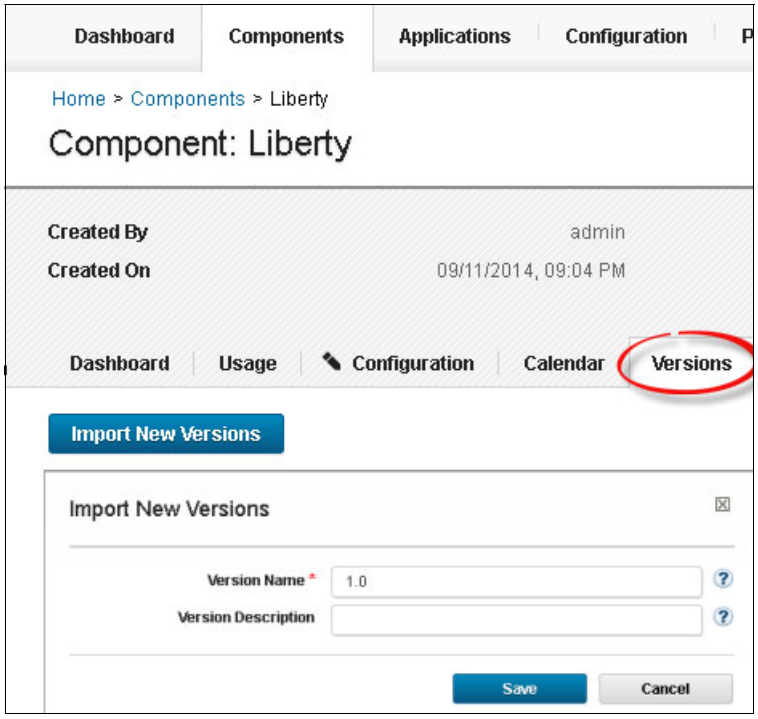


Figure 6-30 Creating the component version

4. Click **Import New Versions**.
5. Enter **1.0** (or the name of your choice) for that version in the **Version Name** field.
6. Click **Save**.

The result is displayed in Figure 6-31 on page 201, where the new version is imported and contains the artifacts (a WAR file and the server.xml files).

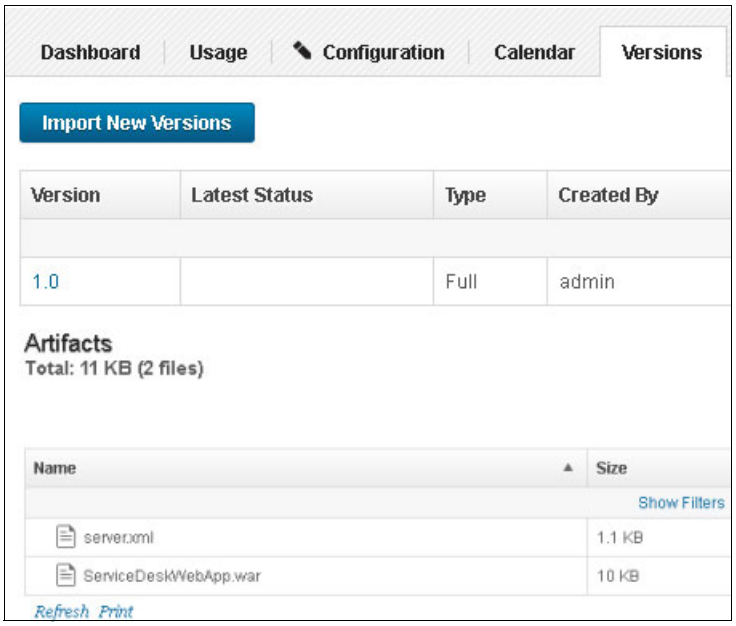


Figure 6-31 The new version is imported

6.4.2 Create the new application

For clarity, components are the individual units that you are deploying. The components are used to create scripts, read the repository, and execute these on the server. The resource unit can represent a file, a server, and so on. The application unit is the combination of the component and resource units that are being deployed. So, application represents a way to organize the pieces of your deployment into one deployable package.

To create the application, follow these steps:

1. Click the **Applications** tab.
2. Click **Create New Application**.
3. In the Name field, enter **Liberty Application**.
4. Click **Save**.

Figure 6-32 on page 202 shows the results.

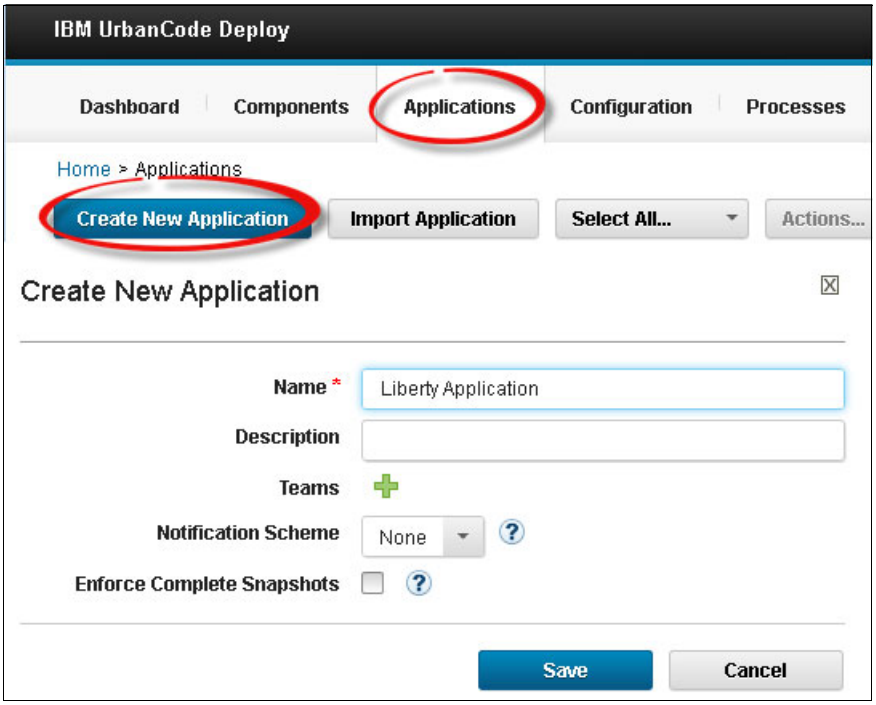


Figure 6-32 Creating a new application

Create the application environment

To create the application environment, follow these steps:

1. Click the **Applications tab**.
2. Select the **Application** that was created in Step 6.4.2, “Create the new application” on page 201.
3. Click the **Create New Environment** button.
4. In the Name field, enter **Legacy**.
5. Click **Save**.

Figure 6-33 on page 203 shows the result.

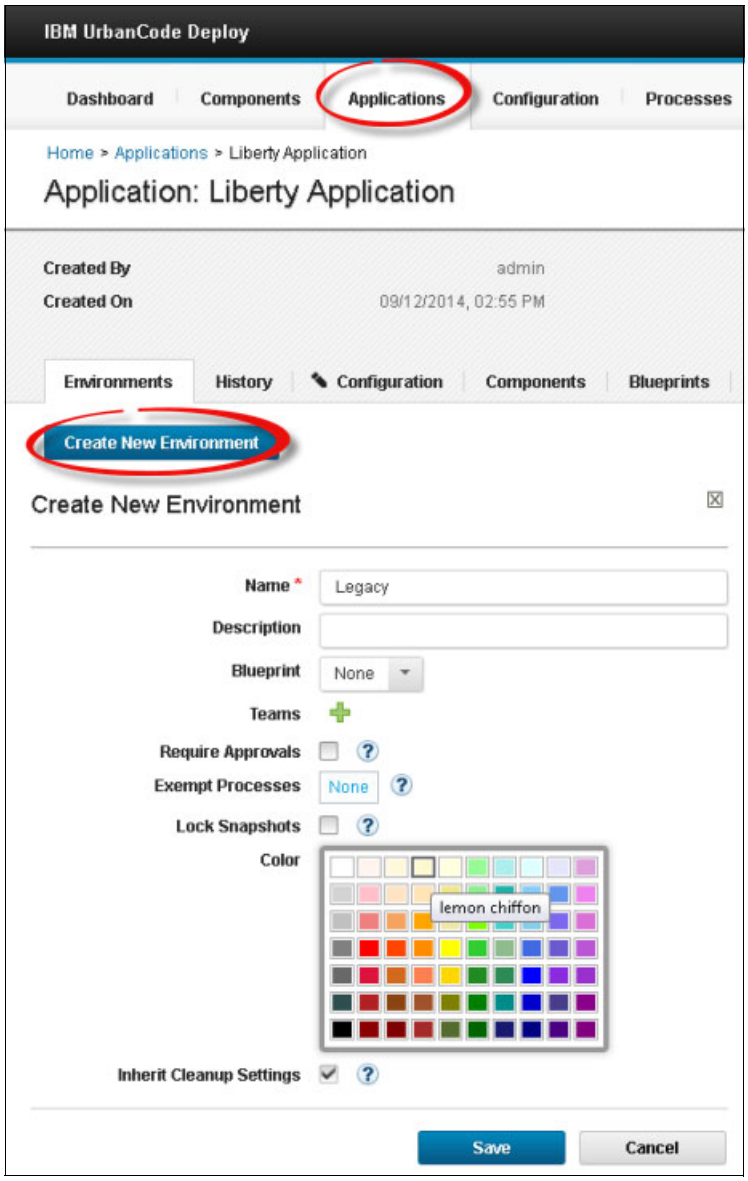


Figure 6-33 Creating the application environment

6.4.3 Create the resource

A *resource* represents what is going to be deployed to the environment. An environment can point to multiple resources, or it can be a single resource. The resource can be, for example, a database, MQ, the location of your Apache server, or any location you are deploying to. Before we can fully configure this environment, we must first create a resource.

Figure 6-34 on page 204 shows that we have already installed the agent on those servers (see “Install the agent” on page 180). An agent lives on a resource and performs deployment operations. An agent downloads the artifacts from your code station repository (or whatever repository you are using) and executes the action commands to deploy your component to the applications.

Figure 6-34 on page 204 shows the results.

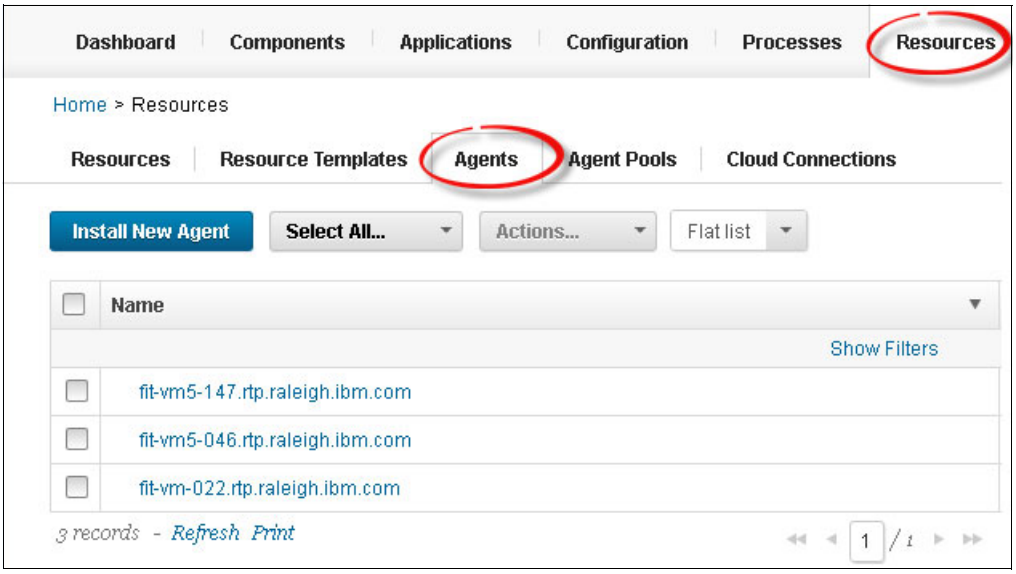


Figure 6-34 Checking the agents

To create the resource, follow these steps:

1. Click the **Resources** tab.
2. Click the **Create Top-Level Group** button.
3. Enter **Liberty-Group** in the Name field.
4. Click **Save**.

Figure 6-35 shows the result.

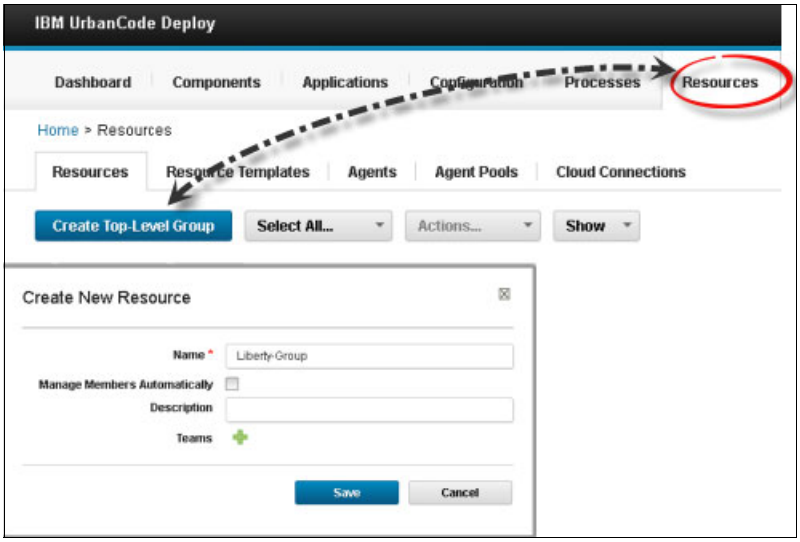


Figure 6-35 Creating the resource

Add an agent to the resource

With the resource *Liberty-Group* created, we need to add our agent to this resource:

1. Select the **Liberty-Group** resource.
2. Click the down-arrow by the **Actions** button, and select **Add Agent**.

3. In the Agent field, choose the **agent** to add.
4. Click **Save**.

Figure 6-36 shows the result.

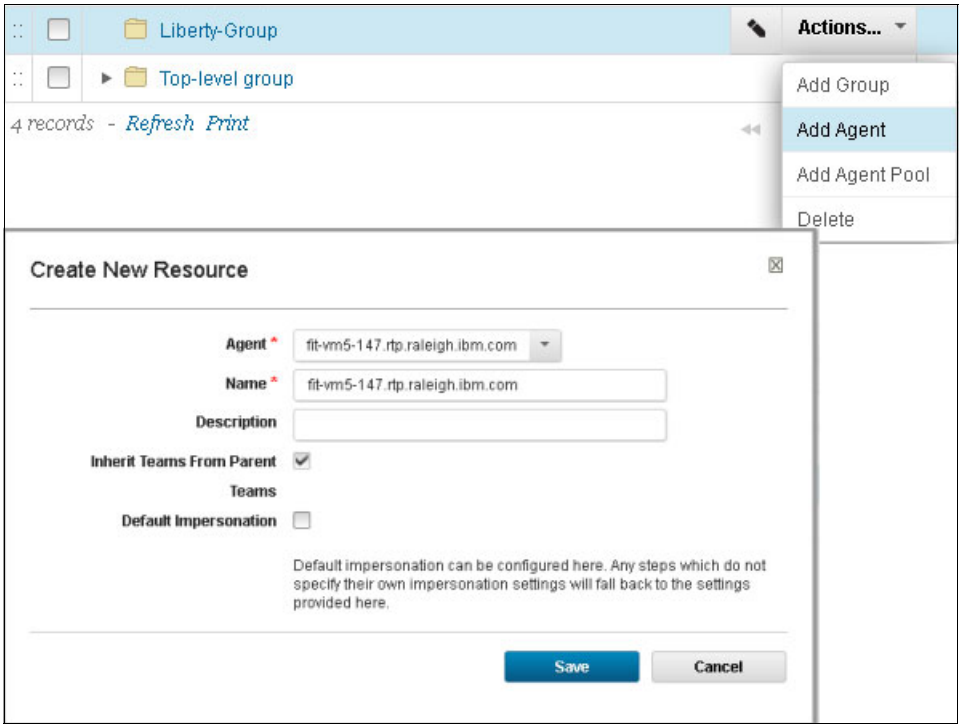


Figure 6-36 Adding an agent to a resource

Figure 6-37 on page 206 shows that the agent is included in the resource, so we have identified the agent to deploy to. Next, we need to add the *Liberty-Group* resource to the *Legacy* environment, which is in the *Liberty Application* application.

To add the resource to the environment, follow these steps:

1. Click the Applications tab.
2. Choose **Liberty Application**.
3. Choose the **Legacy** environment.
4. Click the **Add Base Resources** button.
5. In the list that displays, expand the **Liberty-Group** Item.
6. Select the **agent**.
7. Click **OK**.

Figure 6-37 on page 206 shows the result.

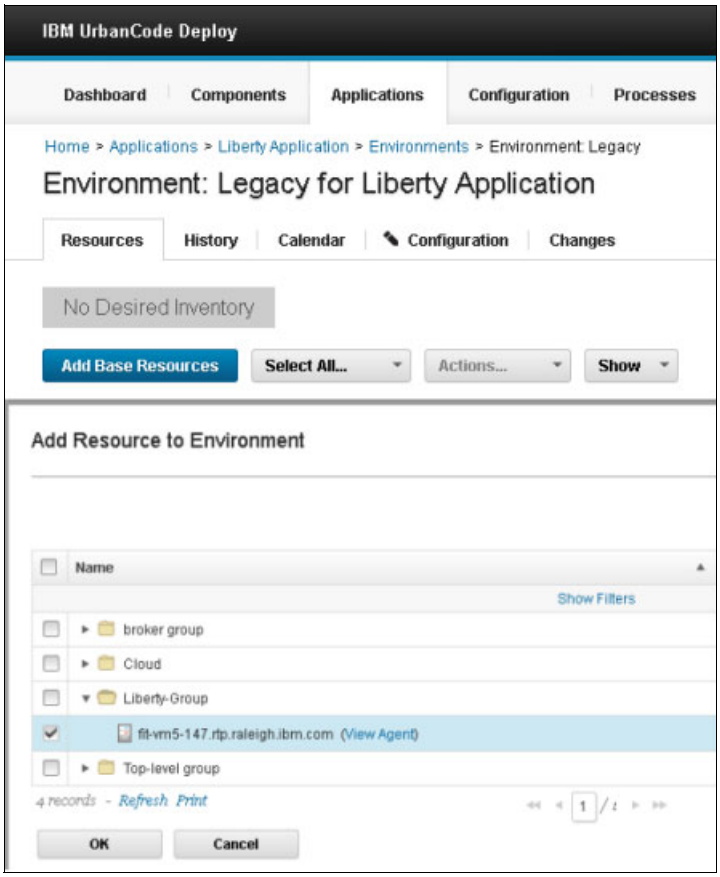


Figure 6-37 Adding a resource to the application environment

6.4.4 Add a component to the application

When we are looking for the Environment “Legacy”, whatever this agent has been installed, right now we are going to be deploying this web application (war and server.xml files) for this agent. We first need to hook the Component with the Application. This is accomplished by navigating to the *Legacy Application*, then associating the *Liberty* component with the resource. This is shown in Figure 6-38 on page 207.

To add the component to the application, follow these steps:

1. Click the **Applications** tab.
2. Choose the **Liberty** application.
3. Click **Application scope** on the Components tab.
4. Click **Add Component**.
5. Choose the **Liberty** component that was created in Step 6.4.1, “Create a component” on page 198.
6. Click **Save**.

Figure 6-38 on page 207 shows the result.

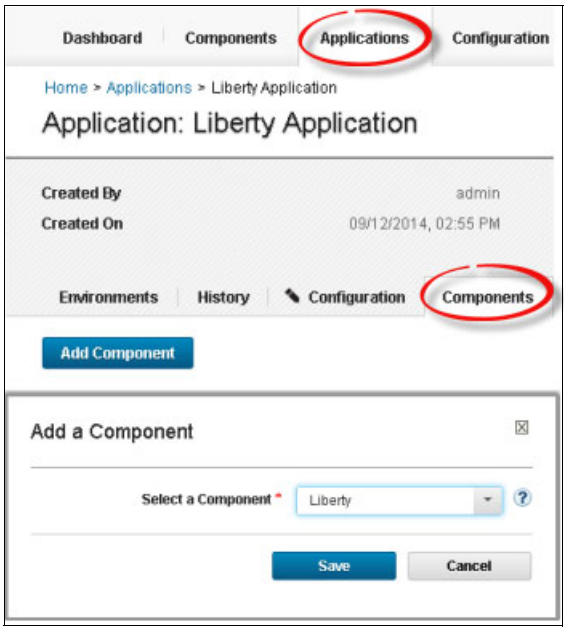


Figure 6-38 Adding a component

6.4.5 Create the component process

The infrastructure is now defined for deploying to the environment. Next, we deploy, using a two-step process: First, by deploying the component; and second, by deploying the application. Recall that the component is a member of the application and that it (the component) has a separate deployment process.

Note: Because we are deploying a WAR file, we need to have a Liberty plug-in installed. To do this:

1. Download the Liberty plug-in available at the following site:

   `https://developer.ibm.com/urbancode/plugins`

2. Navigate to the **Settings tab** → **Automation Plugins**, and click **Load Plugin** to install the `.zip` file that was downloaded in Step 1.

To create the component process, follow these steps:

1. Click the **Components** tab.
2. Choose the **Liberty** component that was created in Step 6.4.1, “Create a component” on page 198.
3. Click the **Processes** tab.
4. Click **Create New Process**.
5. Enter **Deploy War** in the Name field.
6. Click **Save**.
7. Click the **Deploy WAR file** link in the table to open the Design Editor.

Figure 6-39 on page 208 shows the result.

Figure 6-39 Creating the component process

8. Download the artifacts from the repository. In the plug-ins window, expand **Repositories** → **Artifact** → **IBM UrbanCode Deploy**.
9. Drag **Download Artifacts** to your canvas.
10. Accept the default values, then click **Finish**.

Figure 6-40 on page 209 is annotated to show how to add the action to install the WAR file in Liberty before downloading the artifacts.

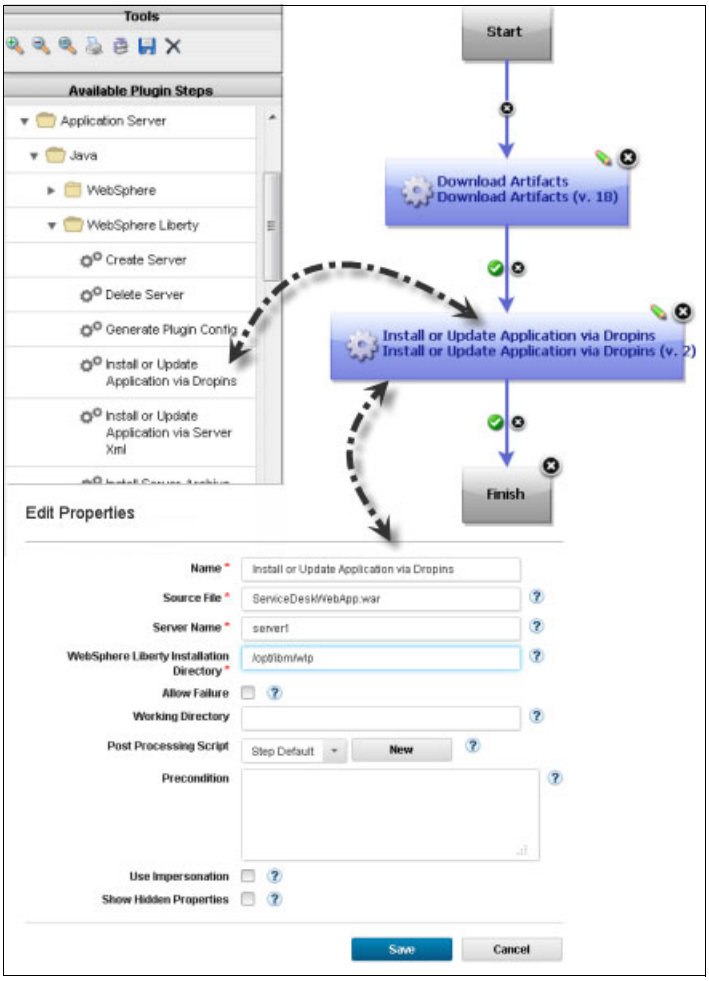


Figure 6-40 Design Editor

After creating the component process, we need to create the application process by following the same procedure listed in Step 4 on page 195 (part of 6.3.2, “Create an application for deployment” on page 195). To do this:

1. Drag the item **Install Component** to the canvas
2. Choose the process that was created for that component. For our scenario, we selected **Deploy WAR**.

To Run the application process:

1. Navigate to the **Applications** → **Environments** tab.
2. Request the process by clicking **Play**.
3. Choose the **Deploy War** process.
4. Click **Submit**.

Figure 6-41 on page 210 shows the result.

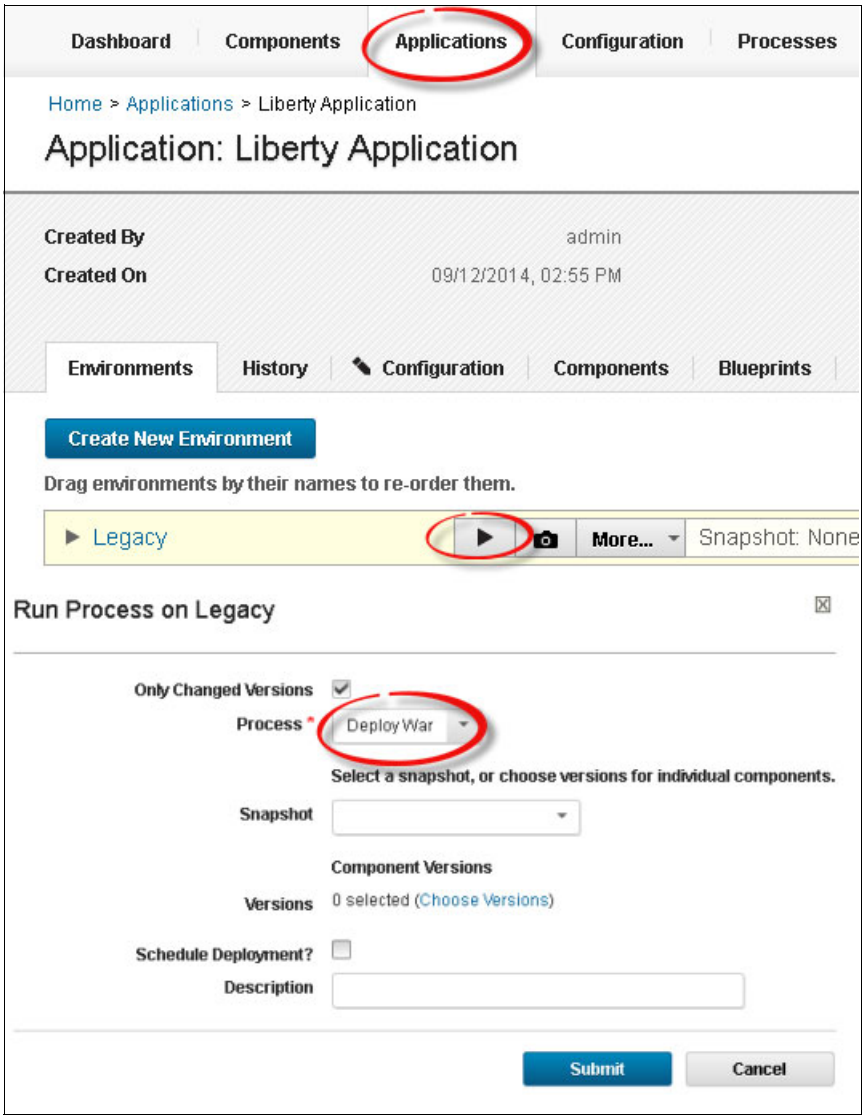


Figure 6-41 Running the application process

7

Quality management and the software development project

This chapter describes how to use IBM Rational Quality Manager for the Collaborative Lifecycle Management project that is described in 1.2, “The business scenario” on page 14.

Rational Quality Manager is designed to help teams collaborate by synchronizing teamwork throughout the lifecycle and by automating labor-intensive activities.

Rational Quality Manager is built on the Jazz platform, a collaborative, role-based, business-driven environment that provides tools for workflow control, tracking, and metrics reporting. It is a collaborative, web-based, quality management solution that offers comprehensive test planning, manual testing, and integration with automated test tools.

This chapter covers the following topics:

- 7.1, “Quality Management” on page 212
- 7.2, “Test Execution” on page 218
- 7.3, “Defect and change management” on page 219
- 7.4, “Test effort management” on page 222

© Copyright IBM Corp. 2014. All rights reserved.

7.1 Quality Management

With Rational Quality Manager, teams can better govern their projects by providing reliable and timely metrics, as shown in Figure 7-1.

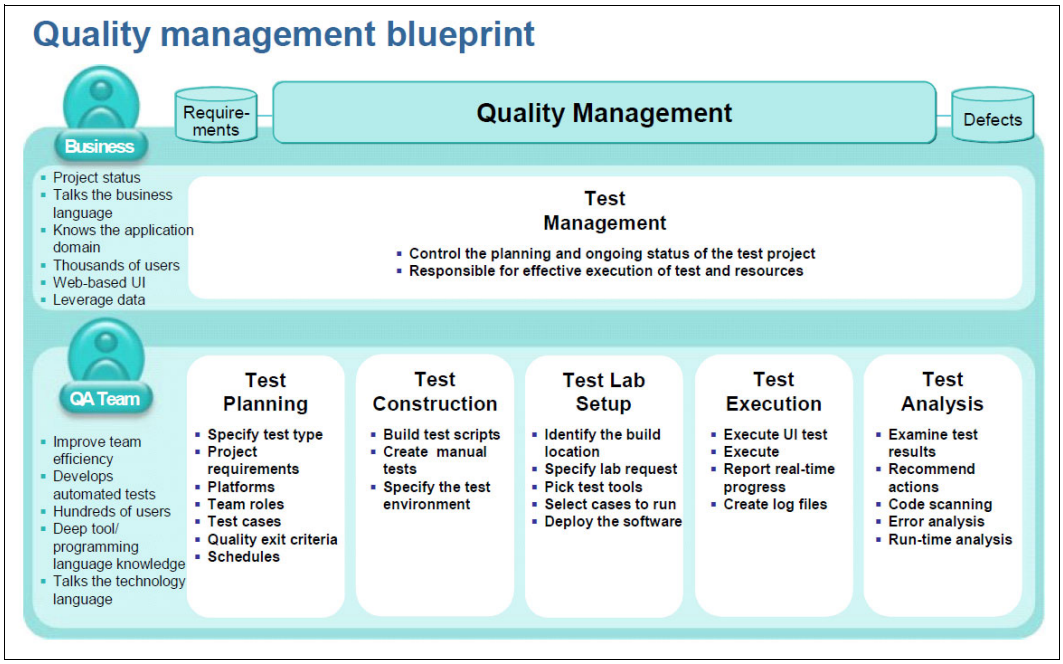


Figure 7-1 IBM Rational software: Quality management blueprint

During the test phase of a software development project, the following questions are frequently asked:

- Are we testing that all of the approved requirements have been met?
- Are we peer-reviewing all tests?
- Is the software working as required and expected?
- Are the testers collaborating test results with the developers?
- Where are the test results to be located?
- What was tested for a specific release?
- Are we ready to deploy?

In a real-world scenario, these are not easy questions. You need to collect massive amounts of data for tracking, including test requirements, test results, and a defects list. This is a challenge, given that the data is often spread across the organization, stored in various tools on multiple servers, and in multiple domains that require access to collect the data.

Collecting all of this data and reducing it down to answer the question, *Are we ready to deploy?* quickly becomes an overwhelming, time-consuming effort.

What is needed is:

- A test plan that captures and tracks different types of data.
- A way to execute manual and automated tests to fulfill the test plan.
- Conversion of the data into information that is usable by project management.
- A way to monitor all tests that are executing.

With quality management, every team member contributes to the quality of the release throughout the development lifecycle (see Figure 7-2 on page 213).

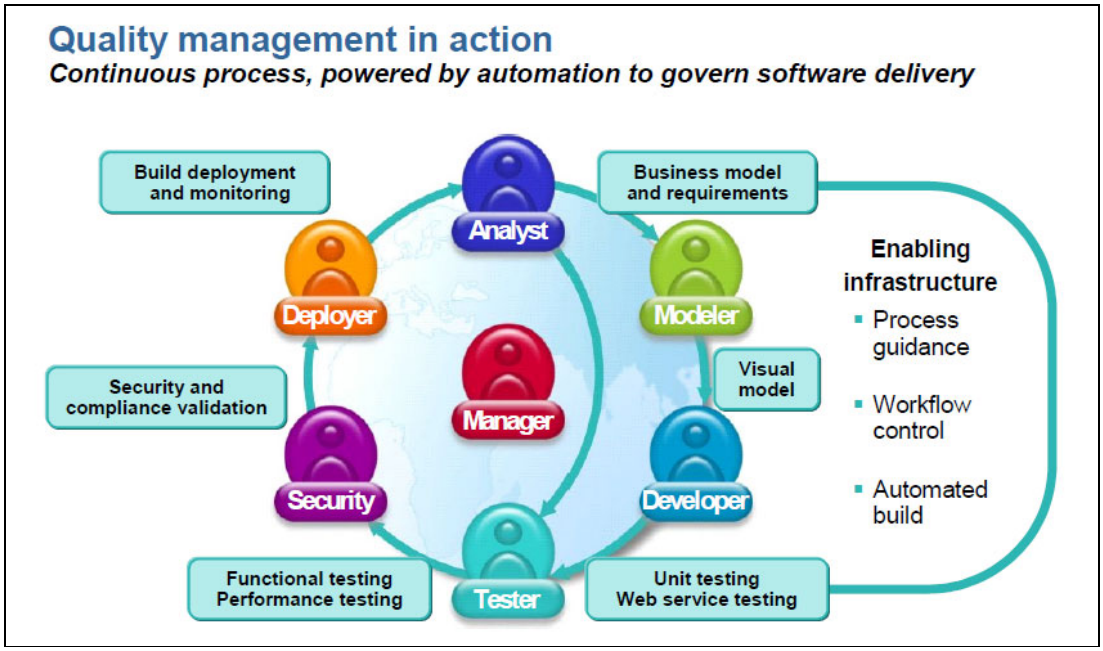


Figure 7-2 Quality management in action

7.1.1 Create a test plan

To create a test plan using Rational Quality Manager, follow these steps:

1. Access your Rational Quality Manager web-page:

   `https://<clm_server_url>:9443/ccm/web/projects/`

2. Select your Project Area from the Quality Management section (see Figure 7-3 on page 214).

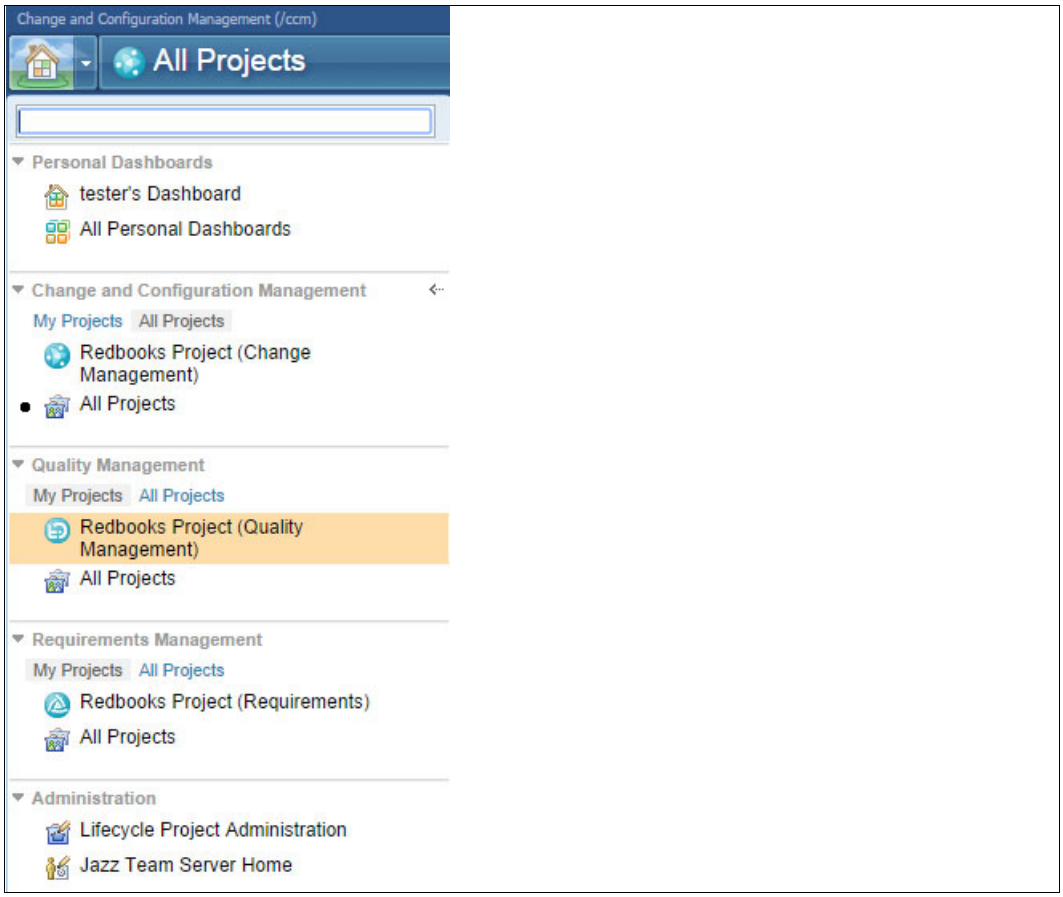


Figure 7-3 Selecting the Rational Quality Manager project area

3. Select **Planning** → **Create Test Plan** (see Figure 7-4).

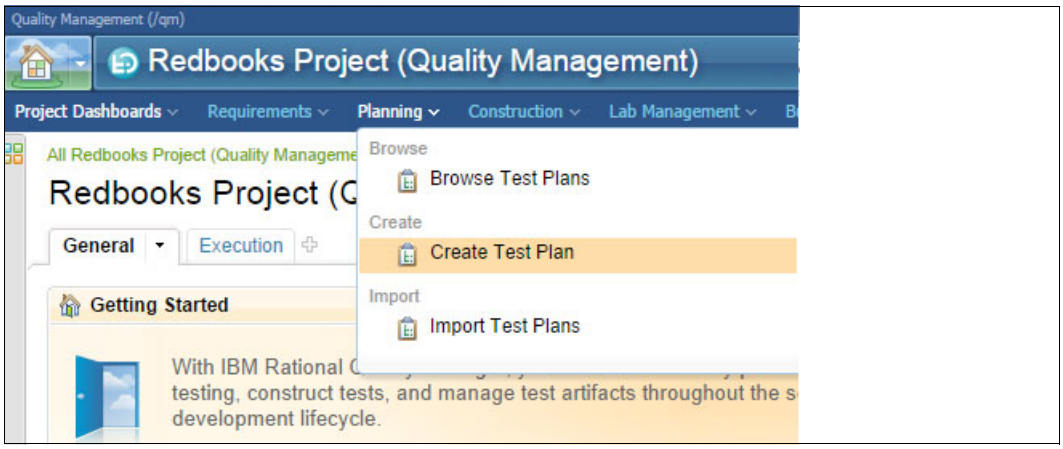


Figure 7-4 Creating a test plan in Rational Quality Manager

4. Select the **Default Template**.
5. **Name** the test.
6. Select an **Owner**.
7. Assign a **Priority**.

8. In the **Summary** section, define the plan categories that you will use, and update each to document your test plan.
9. Click **Save**.

7.1.2 Associate the requirements

After the test plan is created, associate it to the approved requirements using IBM Rational Requirements Composer:

1. Open the test plan.
2. Click **Requirements Collection Links**.
3. Click the **Add new links** button.
4. Select an approved **requirements collection** from Rational Requirements Composer.
5. Click **OK**.
6. Click **Save** (see Figure 7-5).

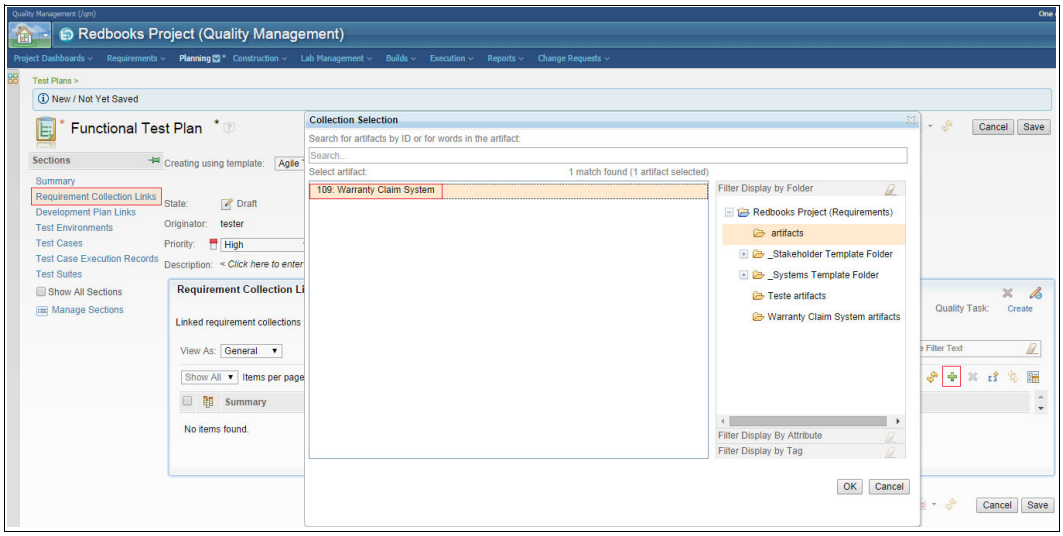


Figure 7-5 Associating a requirements collection with the test plan for traceability

7.1.3 Create a test case

A test case is essential to maintaining a high level of quality throughout the testing process. The test case defines what you need to validate to ensure that the system being tested is functioning correctly.

Each requirement from the associated requirements collection must be associated to a minimum of one test case. Rational Quality Manager can automatically create one test case for each requirement in the associated requirements collection.

To create a test case:

1. Click **Requirements Collection Links**.
2. Check the associated Requirements Collection check box.
3. Click the **Reconcile Requirements in Collections** icon.
4. Select all of the Requirements for which you want to create test cases.

5. Click the **Generate # Test Cases** icon (see Figure 7-6).

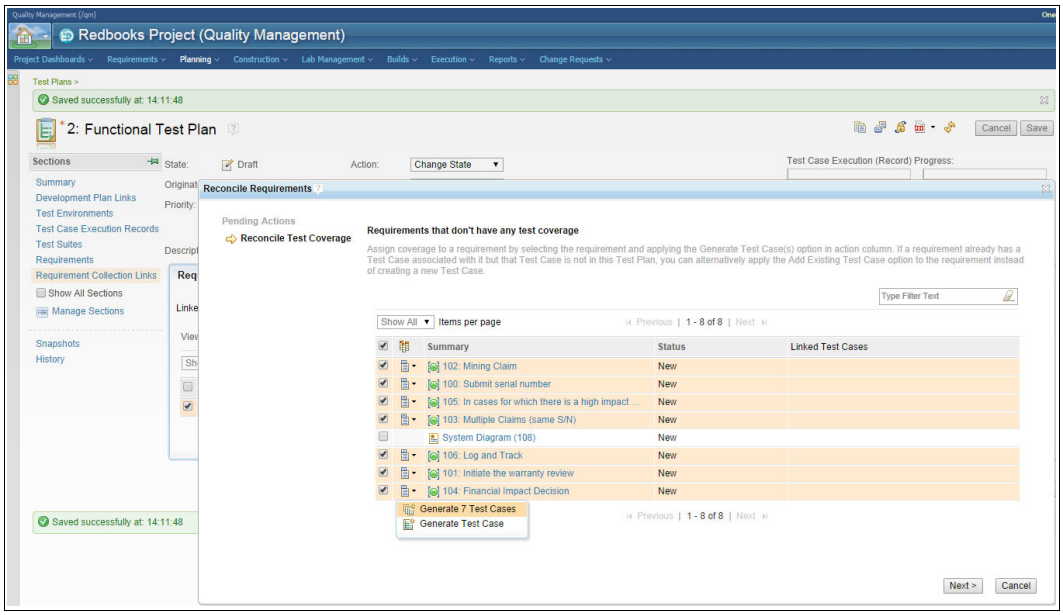


Figure 7-6 Creating test cases from the associated requirements list

6. Assign an **Owner**, and click **OK**.
7. Click **Test Case Design** to enter a definition of the overall design for the current test case. This can include any background setup information or notes about topologies.

To access all of the test cases that were created for this project, click the **Test Case** link in the left panel. In the window that displays, you can enter information about this test case.

7.1.4 Define the platforms and test environments for the test plan

Define the supported environments for this test plan so that we can validate the system:

1. Open the test plan.
2. Click **Test Environments**.
3. Choose all of the platforms that will be covered by the test Plan. To do this, click **Platform Coverage**, then click the pencil icon to select the platforms.
4. Click **Save**.

After the platforms are selected, define all of the test environments that will be used during the tests:

1. Click the **Test Environment Tab**, and select all of the test environments that apply (see Figure 7-7 on page 217).

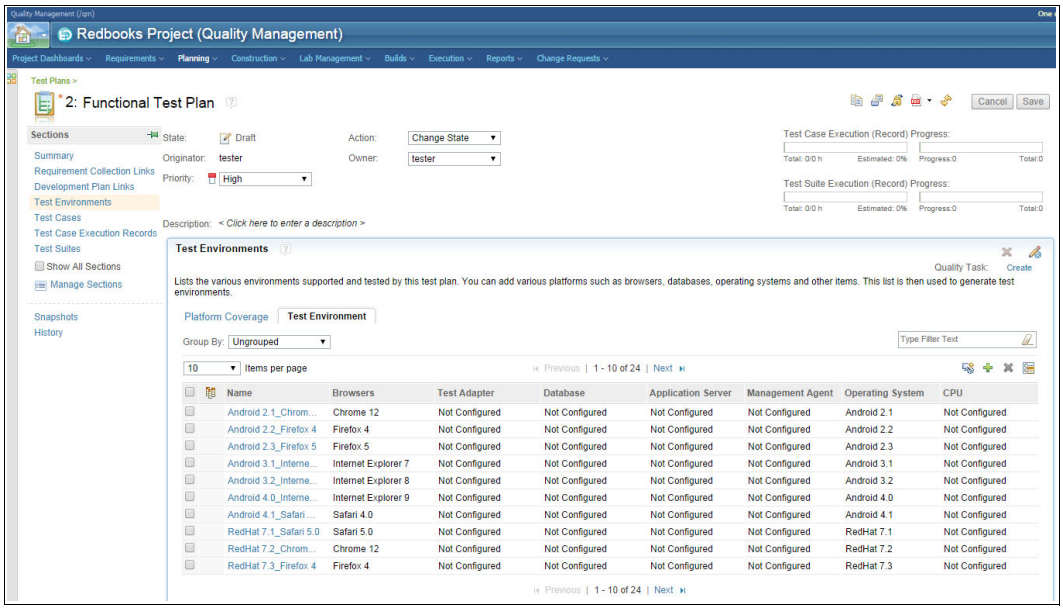


Figure 7-7 Selecting the applicable test environments

2. Click **OK**.
3. Click **Save**.

7.1.5 Create a test script

A *test script* is a manual or automated script that contains detailed instructions for implementing a test case. You can write manual test scripts to be run by a human tester, or you can automate some or all of the instructions in the test script. You can also associate automated functional test scripts, performance test scripts, and security test scripts with a test case. Rational Quality Manager collects the results and reports test progress no matter where the testing was run from.

> **Note:** Rational Quality Manager provides options for automating test scripts using different tools. To integrate Rational Functional Tester with Rational Quality Manager, you need to configure the Rational Functional Tester adapter. This adapter needs to be running on the computer where the test cases will be executed.
>
> For more information, see the following website:
>
> `http://jazz.net`

To create a manual Test Script:

1. Open the selected Test Case where you want to create the Test Script.
2. Click the **Test Scripts** link on the left Panel.
3. Click the "**Create new Test Script**" icon.
4. Specify the **Name** of the test script. Assign the **Owner**. Set **Manual** as the type.
5. Click **OK**. Click **Save**.
6. Click the Test Script to access it and configure it.

7.2 Test Execution

Once all test plans, test cases, and test scripts were created and configured, the testers can start the execution of the test script.

1. Access Rational Quality Manager.
2. Click **Execution** → **Test Cases**.
3. Select the Test Case and click the **Run Test Case** icon.
4. Click **Finish** (see Figure 7-8).
5. Execution of the test case begins. The tester executes all of the steps listed, and for each one, records the **Result**.

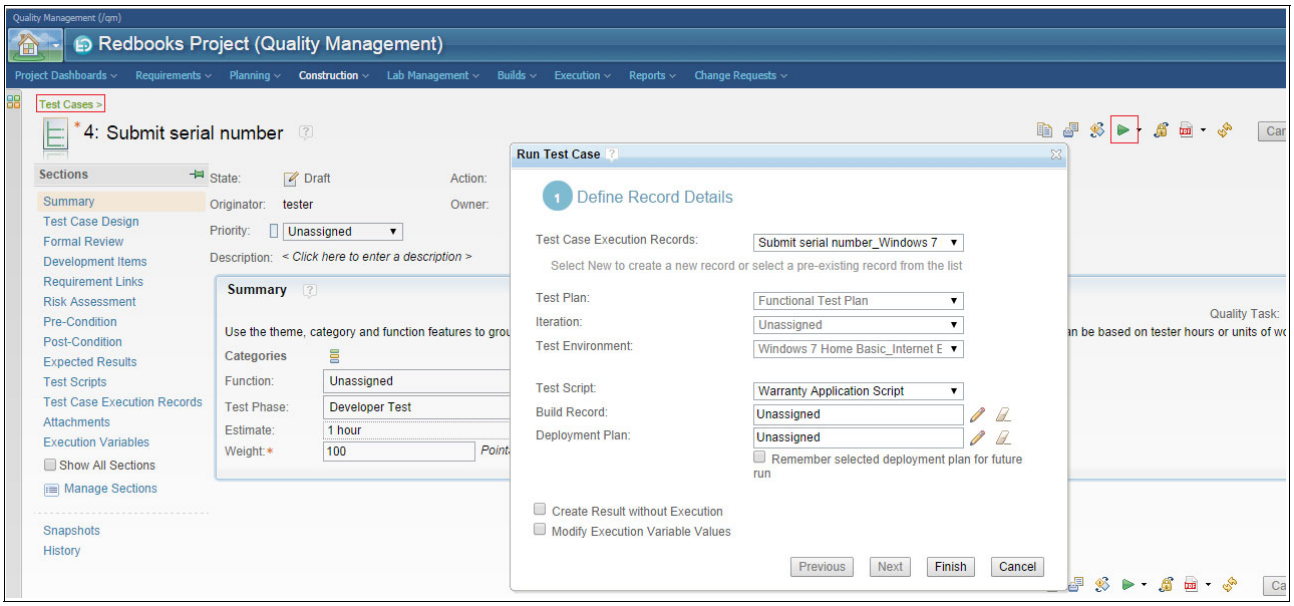


Figure 7-8 Running a Test Case

If an error occurs during the test, the tester can immediately open a Defect by clicking the **Create new defect** icon, and assigning it to the developer for correction (see Figure 7-9 on page 219).

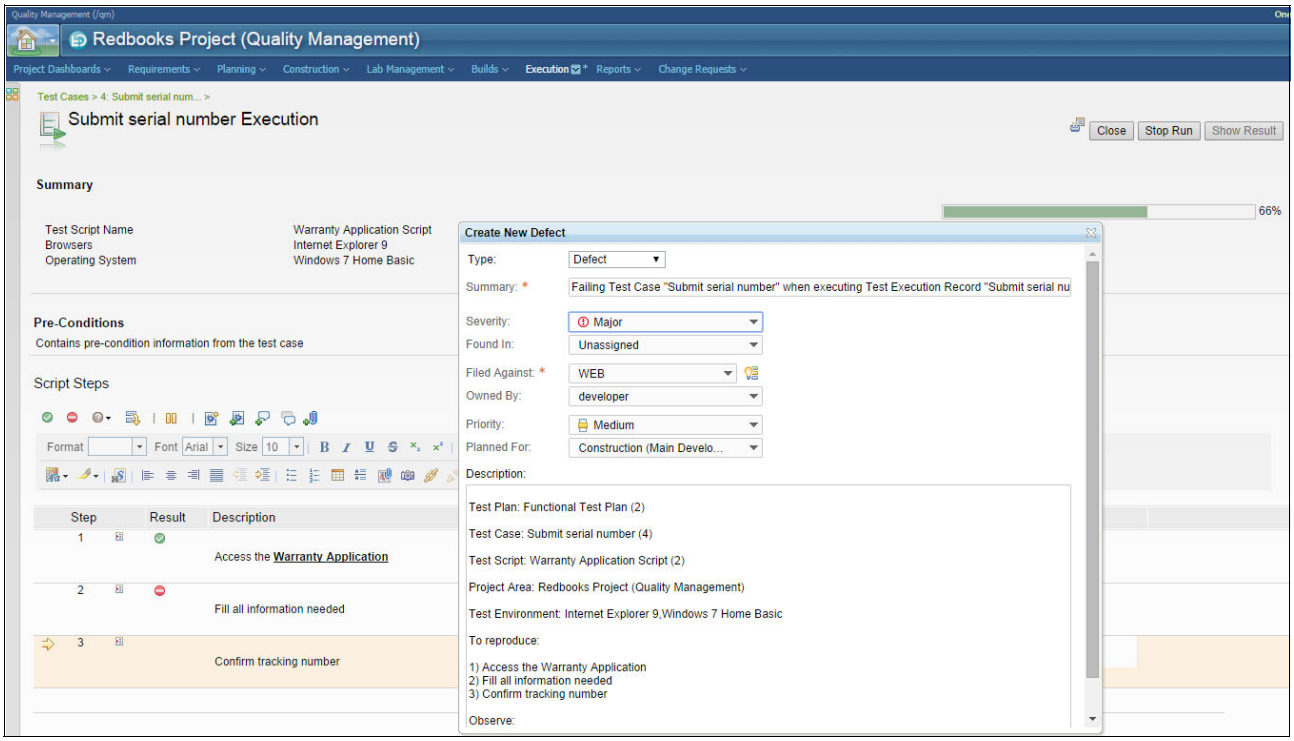


Figure 7-9 Creating a defect during the tests

7.3 Defect and change management

After a defect is opened by the test team and the defect is assigned to the developer by the tester (or project leader), the developer can start solving the problem. The developer uses the Rational Team Concert dashboard or a query to search for new defects that have been assigned. To do so:

1. Open the assigned defect (see Figure 7-10 on page 220).

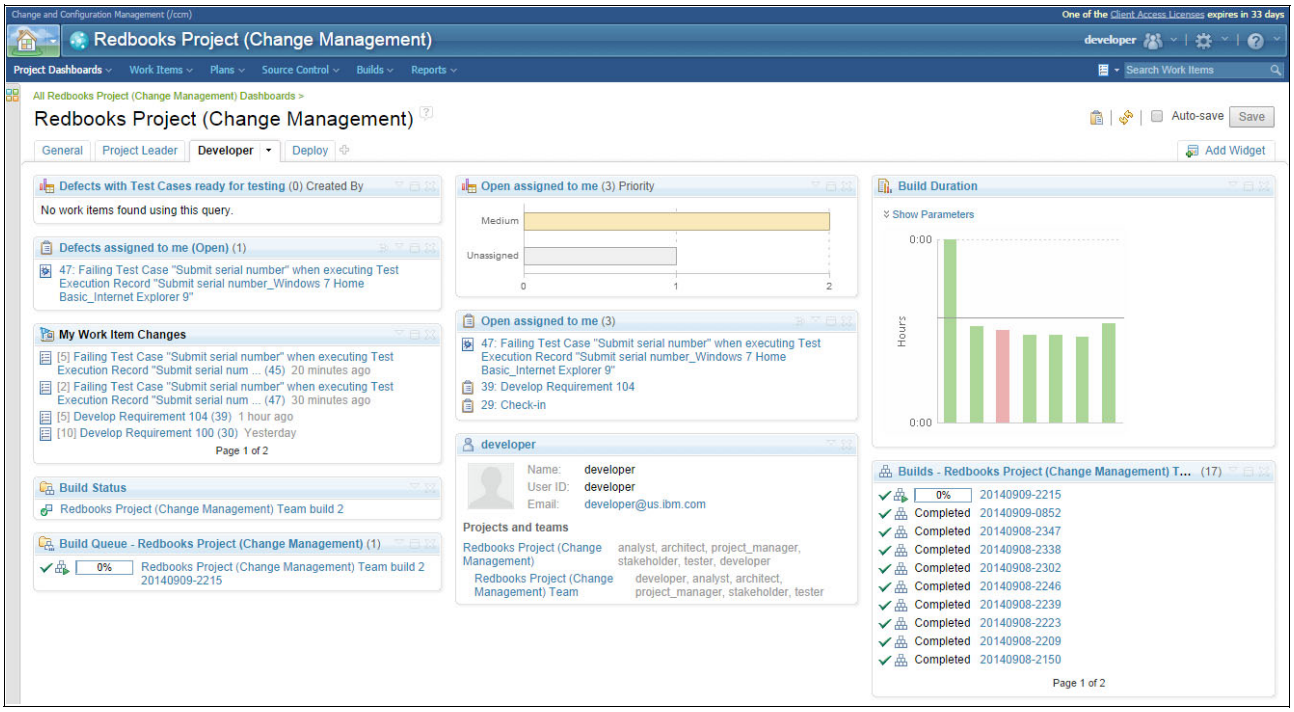


Figure 7-10 The developer uses the Rational Team Concert dashboard to monitor defects

2. Change the status to **in_execution**.
3. Fix the code, then associate the fix to the defect (for traceability).
4. Set the defect status to **Resolved** so that the tester can retest the code (see Figure 7-11 on page 221).

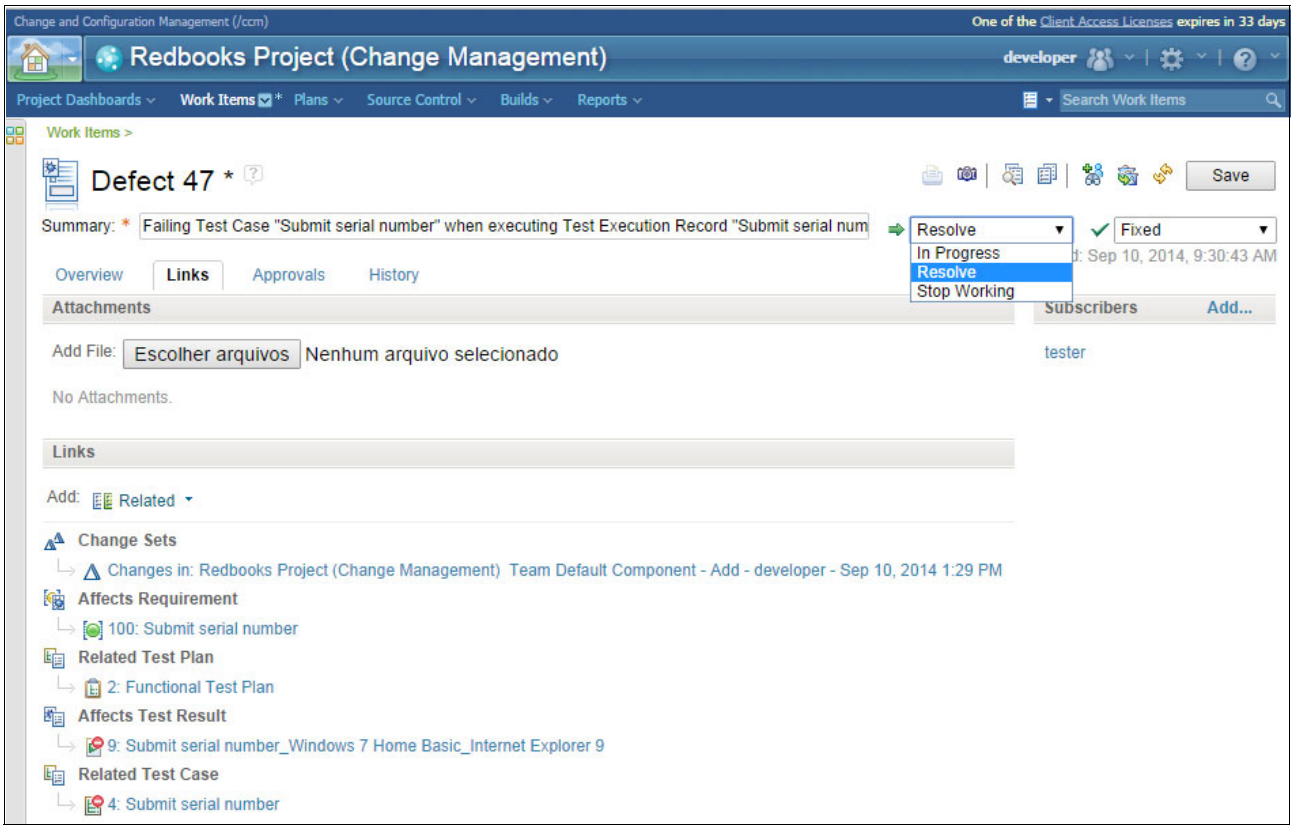


Figure 7-11 The developer resolves the defect that was opened by the test team

Note: The steps in 7.3, “Defect and change management” on page 219 are valid for change management. After a change request is opened in Rational Team Concert, it is available to the analysts, project leader, and project manager to plan the release.

7.4 Test effort management

Rational Quality Manager provides the project manager and project leader all of the information about quality management activities and processes that are involved in a software development project.

Rational Quality Manager contains predefined reports, dashboards, and graphics that promote the continuous quality improvement of a project. The following are examples (Figure 7-12). Using the Rational Quality Manager Dashboards, the users have a real-time vision of all test management activities.

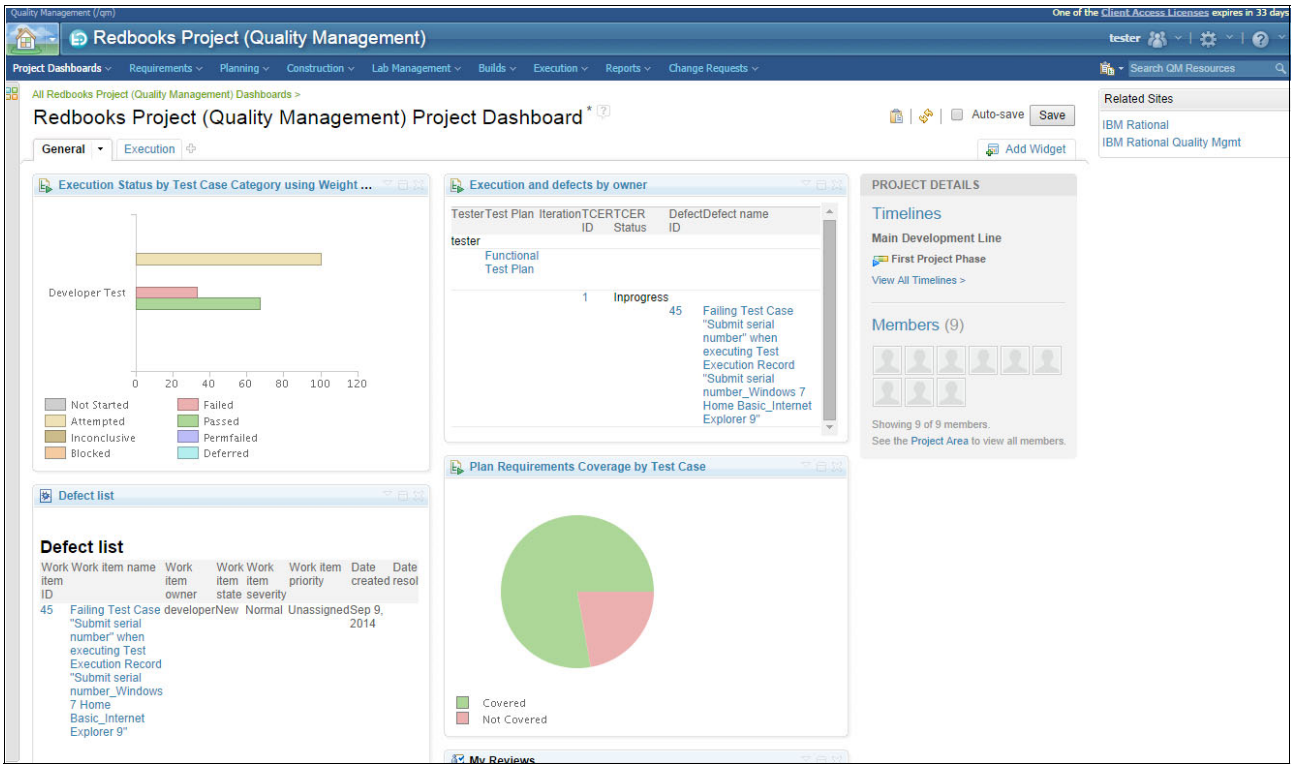


Figure 7-12 An example of a Rational Quality Manager project dashboard

A

Integrating the Eclipse environment with IBM Rational Collaborative Lifecycle Management

This appendix describes the steps to integrate an Eclipse integrated development environment (IDE) with an IBM Rational Collaborative Lifecycle Management project. The integration requires the use of IBM Rational Team Concert Eclipse plug-in.

The following topics are covered:

- “Configure installation between Eclipse IDE environment and Rational Team Concert” on page 224
- “Install the Rational Team Concert Eclipse plug-in” on page 225
- “Rational Team Concert and third-party products” on page 225
- “Integrating an Eclipse Workbench with Rational Team Concert” on page 225
- “Rational Team Concert and UrbanCode Deploy for continuous integration” on page 232

© Copyright IBM Corp. 2014. All rights reserved.

Configure installation between Eclipse IDE environment and Rational Team Concert

This appendix describes the integration of the products that are used in the scenario described in 1.2, “The business scenario” on page 14.

As a developer, you only need to configure the installation between your Eclipse IDE and Rational Team Concert. Rational Team Concert performs the integration with all other Rational Collaborative Lifecycle Management products.

Important: This appendix continues with the processes that apply to the business scenario described in this book. As such, before performing the steps in this appendix, we advise you to perform all of the steps in Chapter 3, “Application Lifecycle Management on a software development project” on page 27.

Note: For details about integrating Rational Collaborative Lifecycle Management products with products outside of the business scenario described in this book, see the following website:

http://jazz.net

The integrated team

Multiple roles are required to integrate on a Rational Collaborative Lifecycle Management project. These roles are:

- *Project manager*: The person in this role needs complete awareness of all project activities, including project health, delivery status, overdue tasks, build status, project metrics, reports, and issues that the team is facing.
- *Project leader*: The person in this role leads all team members and manages their activities. The project leader is also responsible for assigning activities for each one.
- *Developers*: Developers have a private source control space on the server (workspace) where changes can be shared with others and accepted from others. Collaboration between developers is expected, as is flexibility when tasks are reprioritized. Persons in this role need real-time information about team activities. Developers are the users of the powerful build engine that enables the build, test, and deploy processes to be automated. They are also the receivers of automated feedback about the quality of the code delivered to source control.
- *Testers*: Testers require real-time interaction with developers, and therefore use the same collaboration platform that developers do. The process from defect reporting, to code fix, build, test, and release, is tightly integrated, and increases the efficiency of reporting defects, fixing defects, and moving fixes back to testers quickly.
- *Stakeholder* (*customer*): The stakeholder participates during all phases of the project. With efficient and thorough communication, the stakeholder is able to develop an improved predictability of project delivery dates, increases in quality, and reduced time to fix defects.

Install the Rational Team Concert Eclipse plug-in

You can install the Rational Team Concert plug-in on any supported version of Eclipse or Eclipse-based IDE (for example, Integration Designer, IBM Worklight, and Rational Application Developer). To perform the installation, follow these steps:

1. Download the “Rational Team Concert p2 install repository” from the Rational Team Concert download page, located at the following site:

   http://jazz.net/downloads
2. Start the Eclipse environment and select **Help** → **Install New Software**.
3. In the Available Software window, click **Add**.
4. In the Add Repository window, click **Archive** and navigate to the p2 install repository that you downloaded in Step 1.
5. Click **OK**.
6. Ensure that **Group items by category** is selected.
7. Select the Rational Team Concert client (extend an Eclipse installation), and click **Next**.
8. Select **Use already existing package**.
9. Click **Next**, then **Next** again in the Install Details window.
10. Accept the terms of the license agreement, then click **Finish**.
11. A security warning window opens during the installation because the Rational Team Concert client bundles and features are not signed. Click **OK** to complete the installation.
12. Click **Restart Now** when prompted.
13. Follow the steps described in “Integrating an Eclipse Workbench with Rational Team Concert” on page 225 to configure the integration to Rational Team Concert.

Rational Team Concert and third-party products

Rational Collaborative Lifecycle Management integrates with various third-party products and tools.

The integration adapters are based on specifications from Open Services for Lifecycle Collaboration (OSLC), also known as Open Services. The Collaborative Lifecycle Management effort makes extensive use of the OSLC, which defines a common set of resources, formats, and Representational State Transfer (REST)ful services.

Note: For more information about the Rational Collaborative Lifecycle Management integrations that are available, visit the following website:

http://jazz.net

Integrating an Eclipse Workbench with Rational Team Concert

When integrating an Eclipse Workbench with Rational Team Concert, you can associate all source control activities with your assigned project activities. This promotes source control management (on a centralized repository) and collaboration for the project teams.

Note: Before integrating an Eclipse Workbench with Rational Team Concert, a plug-in is required. See “Install the Rational Team Concert Eclipse plug-in” on page 225.

Configure the repository connection

Use the following steps to configure the Repository connection:

1. Open the Eclipse client and click **File** → **Accept Team Invitation**.

   Paste the text that you received in your email when your user was created by the Rational Team Concert administrator (see Example A-1).

   Example A-1 Defining the repository connection

```
teamRepository=https://fit-vm5-221.rtp.raleigh.ibm.com:9443/ccm/
userId=developer
userName=developer
projectAreaName=Redbooks Project (Change Management)
```

2. Click **Finish.**
3. Click **Window** → **Show View** → **Team Artifacts**, and the project area displays (Figure A-1).

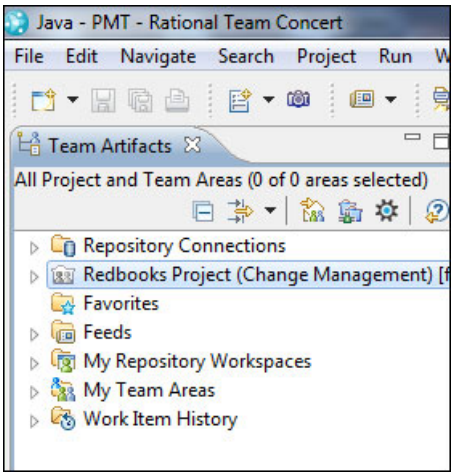


Figure A-1 Listing the configured Rational Team Concert project area

Create a Rational Team Concert Repository Workspace

The Repository Workspace is a replica of the server content on the developer’s workstation (also called a Sandbox). This Repository Workspace contains all of the files and folders that are shared with all team members. After working on the development artifacts, you deliver the changes back to the server.

Use the following steps to create a Rational Team Concert Workspace:

1. Carry out the steps in “Configure the repository connection”.
2. Open the Eclipse client and click **Window** → **Show View** → **Team Artifacts**.
3. On the Team Artifacts tab, right-click **My Repository Workspaces** and select **New Repository Workspace**.

4. Click **Flow with a stream**, and select a Stream to follow. When creating streams or components:
   a. Expand the Project Area where you want to create the stream or component.
   b. Right-click **Source Control**, and select **New** → **Stream**.
   c. Complete the information and click **Save**.

> **Note:** For information about file and folder permissions and security, see the following website:
>
> http://jazz.net

5. Enter a **Name** and **Description** for the new repository workspace, and click **Next**.
6. Select the option to **Use current repository** and click **Next**.
7. Select the security option to apply to the repository workspace, and click **Next**.
8. Select the components to add to the repository workspace.
9. Click **Load repository workspace**.
10. Click **Finish**.
11. Click **Load the root folders**, and click **Next**.
12. Select the components to load.
13. Expand **Advanced Options** and enter the location of the directory to load the server files to.
14. Select **Load and create**, then click **Finish**.
15. Click **Windows** → **Show View** → **Pending Changes** (see Figure A-5 on page 231).

Your workbench is now configured to share files and folders with other team members.

Follow the steps described in “Check in and accept changes” on page 229.

Sharing the Project with Rational Team Concert

This topic must be followed only if you want to share your Project in Rational Team Concert Repository Workspace:

1. Ensure that you already followed steps “Install the Rational Team Concert Eclipse plug-in” on page 225.
2. Ensure that you already followed steps “Create a Rational Team Concert Repository Workspace” on page 226.
3. In the Eclipse client, click **Window** → **Open Perspective** → **Java** → **OK**.
4. Right-click the Project to share, then select **Team** → **Share Project** (Figure A-2 on page 228)**.**
5. Click **Jazz Source Control**, and click **Next**.
6. Select the **Repository Workspace**, and click **New Component** → **Create a Component**.
7. You can use the same name of the Project you want to share. Click **OK**.
8. Click **OK**, then click **Finish**.
9. In the Eclipse client, click **Windows** → **Show View** → **Pending Changes**.
10. Locate the **Change Set** related to the project shared in Step 7.

11. Right-click and select **Deliver**.

Now your project is under version control and is being controlled by Rational Team Concert SCM functionality. Each modification you do will be tracked and controlled by Rational Team Concert. Follow steps described on “Check in and accept changes” on page 229 to know how to handle your activities on Rational Team Concert.

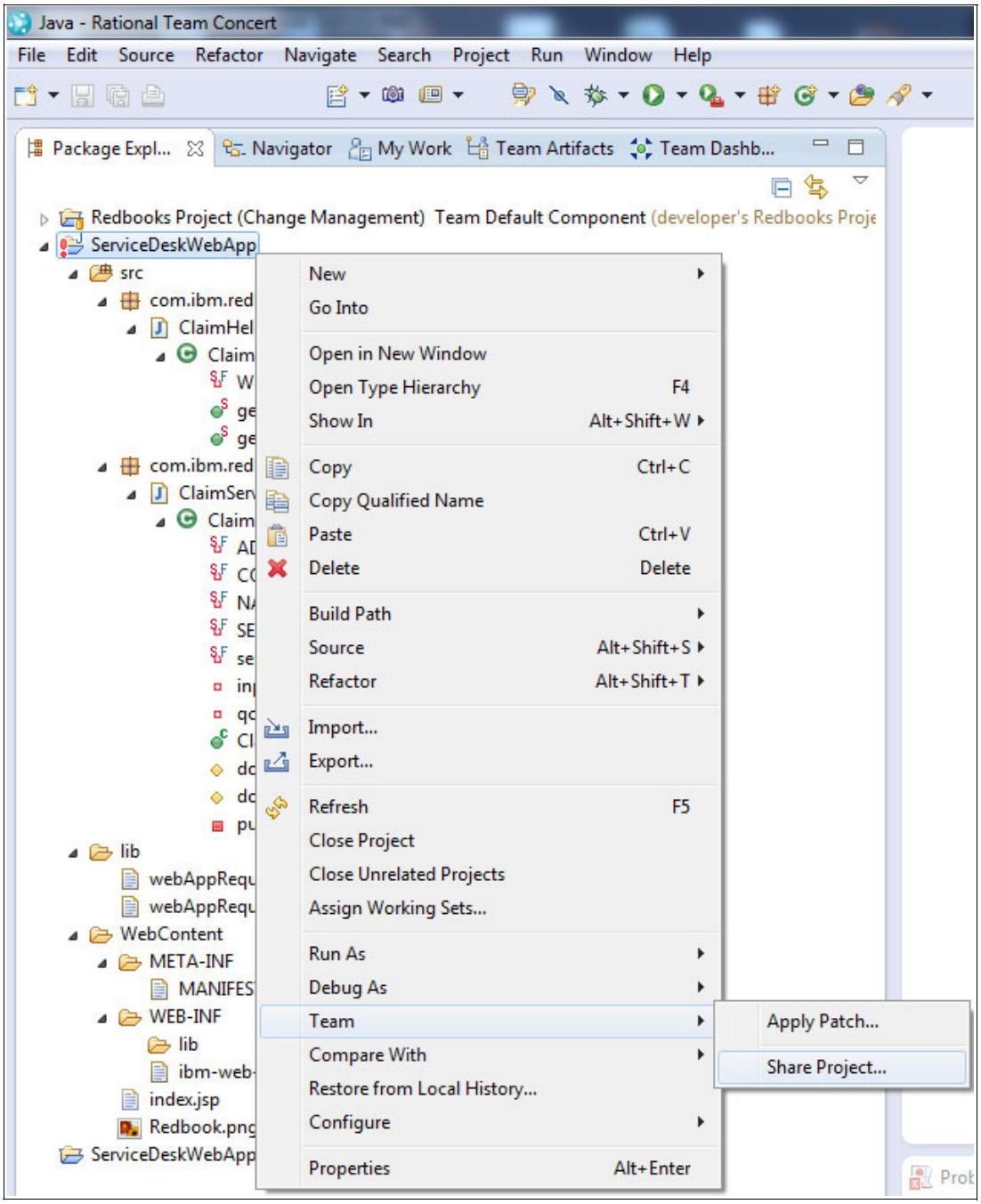


Figure A-2 Sharing a Java Project with Rational Team Concert

Associate the Change Set with a Task

You can associate your Change Set with a Task (a Rational Team Concert Workitem) that has been assigned to you. To do so, follow these steps:

1. Ensure that you already followed steps “Create a Rational Team Concert Repository Workspace” on page 226.
2. Right-click **Change Set**, and select **Related Artifacts** → **Associate Workitem**.
3. Search for the related Task and click **OK**; (see Figure A-3).

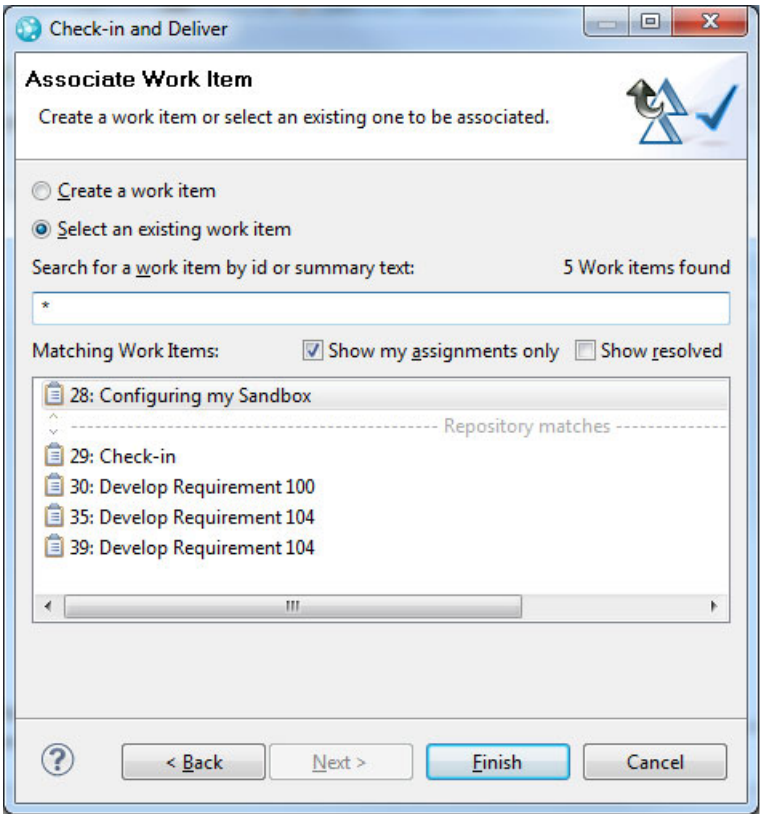


Figure A-3 Associate the Change Set with an assigned Task

4. Right-click **Change Set**, and select **Deliver**.
5. Proceed to “Check in and accept changes”.

Check in and accept changes

Using the Rational Team Concert, all the changes that you make during the coding will be tracked by Rational Team Concert as a Change Set.

A change set is a collection of files or folders that are applied to a flow target. Change sets are visible in the Pending Changes view.

Each time that you make a modification to a file or folder and then save it, you are required to check in and deliver the change to the Rational Team Concert server to save your changes on the server.

Every time that other team members make a change on the code and deliver it to the Rational Team Concert server, you will receive a notification about that new available modification. You can choose which modifications that you accept to download and update your machine.

Checking in and accepting changes is a continuous effort for developers during all development activities, as this ensures that your Sandbox is up-to-date.

Check in your changes

Checking in your changes is done as follows:

1. In the Eclipse client, click **Windows** → **Show View** → **Pending Changes**.

   A list displays, showing the pending changes you need to **Check-in and Deliver** to the server. It also displays the changes made by other team members that you need to **Accept** to keep your local Sandbox updated. Rational Team Concert groups all the changes as **Change Sets**.

2. Locate the **Change Set** to check in.
3. Right-click and select **Check-in and Deliver**.
4. Enter a **Change Set** comment, and click **Next**.
5. Associate the current **Change Set** with your assigned **Task**. Search for the Task and click **Finish**; (see Figure A-1 on page 226).

   Now, the Change Set that contains your modifications is associated with the Task. You can see all Change Sets that are related to a Task by navigating to the Links Tab for a specific Task (see Figure A-4).

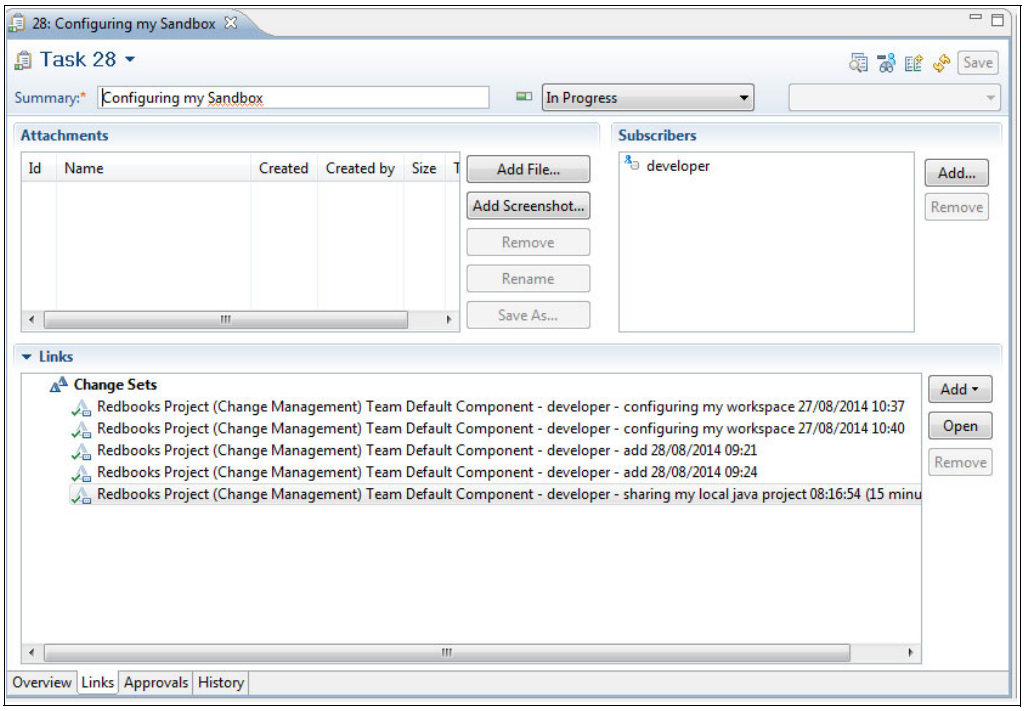


Figure A-4 You can display all Change Sets that are related to a specific Task

Accepting others' changes

When other team members deliver changes to the server, you need to accept those changes to keep your Sandbox updated. To do so, follow these steps:

1. In the Rational Team Concert client, click **Windows** → **Show View** → **Pending Changes**. A list of pending changes displays. You are expected to accept these (see Figure A-5).

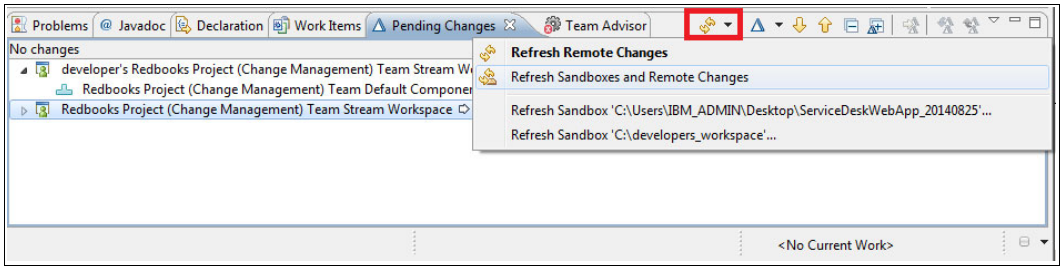


Figure A-5 Searching for local and remote changes to accept

2. Locate the Change Set to download to your machine.
3. Right-click and select **Accept** (Figure A-6)

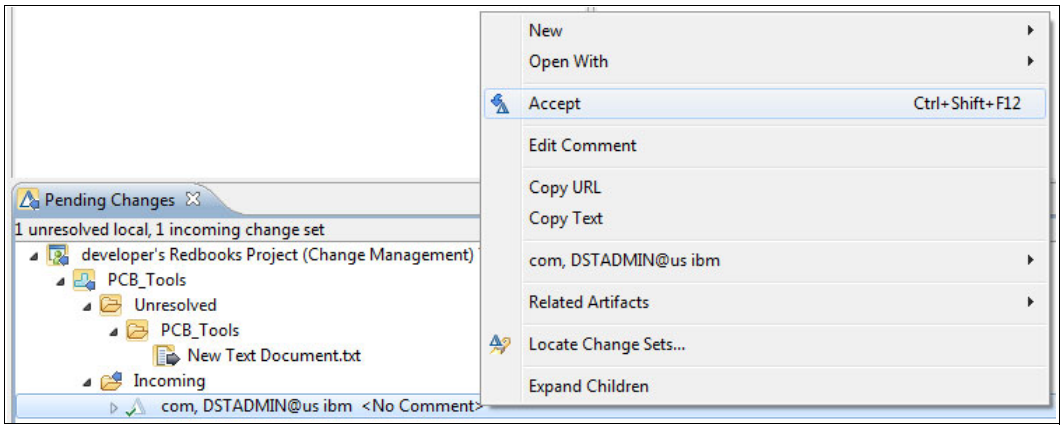


Figure A-6 Accepting changes

Your Sandbox is now updated with the files from the server.

Rational Team Concert dashboards

During the software development project, the project leader tracks project activities using Rational Team Concert dashboards (see Figure A-7 on page 232).

1. Open the Rational Team Concert web page.
2. Select your Rational Team Concert Project Area.
3. Click over **Project Dashboards** and select the wanted Dashboard that you want to access.

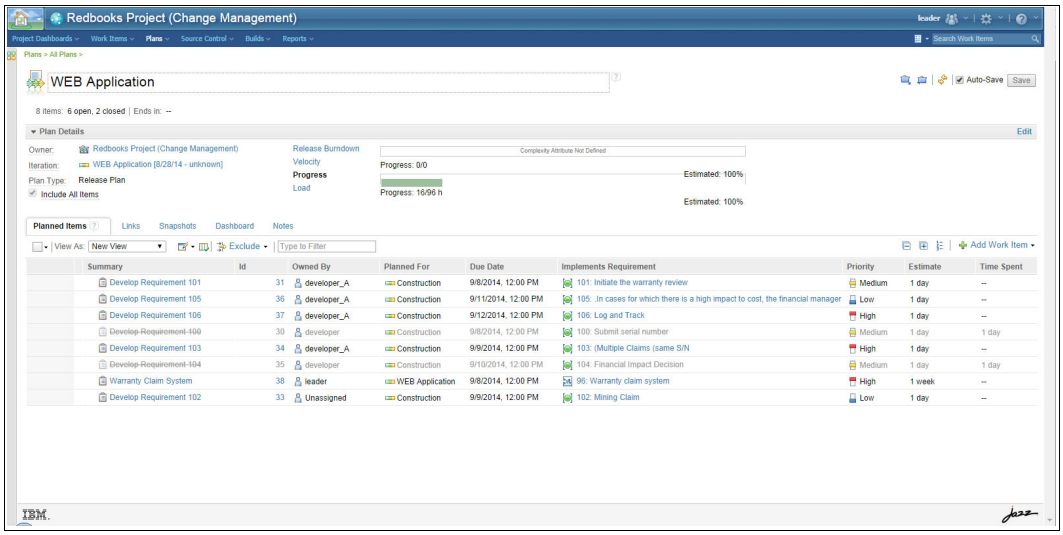


Figure A-7 Tracking project health using Rational Team Concert dashboards

Rational Team Concert and UrbanCode Deploy for continuous integration

Rational Team Concert implements a Team Build component, which provides support for the automation, monitoring, and awareness of a team's regular builds. This component provides a model for representing the team's build definitions, build engines, and build results. The model supports teams with different build technologies.

You can use the Rational Team Concert Build Agent with UrbanCode Deploy to improve continuous delivery by publishing build artifacts to a new UrbanCode Deploy component version. The build artifacts are then available in UrbanCode Deploy to deploy manually or on a regular schedule.

You can also automatically update a test or production environment by configuring the post-build deploy option to request that UrbanCode Deploy run an application process. Figure A-8 on page 233 shows the process as carried out for the scenario described in 1.2, "The business scenario" on page 14.

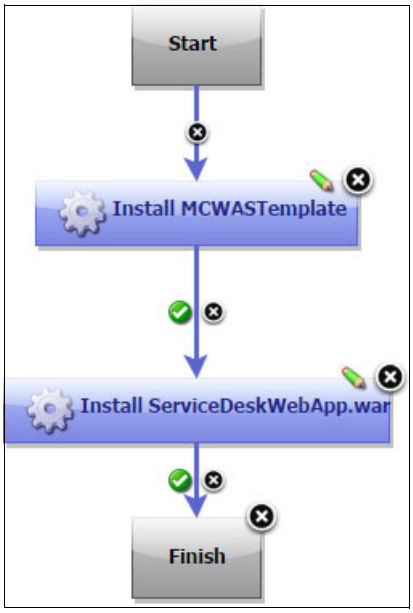


Figure A-8 The application deployment process used in our business scenario

Configure UrbanCode Deploy

For this process, see 6.2, “Prepare the environment” on page 179.

Configure Rational Team Concert

To configure Rational Team Concert, use the following steps:

1. In the Eclipse client, click **Window** and select **Show view** → **Team Artifacts**.
2. Click **OK**.
3. Expand the project area, then right-click **Builds** and select **New Build Definition**.
4. In the New Build Definition wizard, on the New Build Definition page, accept the default settings, and click **Next**.
5. On the General Information page, in the Available build templates pane, select an applicable template, and click **Next**.
6. Select Jazz Source Control, and click **Next**.
7. On the Post-Build page, to configure a deployment, select **Post-build Deploy**, and click **Next**.
8. On the Additional Configuration page, select any applicable options, and click **Finish**.
9. Click the Jazz Source Control tab.
10. Select the developer’s Workspace that was created in “Create a Rational Team Concert Repository Workspace” on page 226.
11. Click the Post-build Deploy tab:
    a. In the Trigger Policy section, select the options for triggering a post-build deploy.
    b. In the UrbanCode Deploy server information section, enter the connection information.
    c. To verify the connection details, click **Test connection**.

d. In the Publish Artifacts section, enter the names of the files to upload to a component version on UrbanCode Deploy. In our scenario, we used `ServiceDeskWebApp.war`.

e. In the version field, enter:

```
$(buildLabel)
```

f. In the Include files field, enter:

```
ServiceDeskWebApp.war
```

g. In the Properties field, enter:

```
buildResultUUID=$(buildResultUUID)
snapshotUUID=$(team.scm.snapshotUUID)
workspaceUUID=$(team.scm.workspaceUUID)
```

h. In Links field, enter:

```
Build
Result=${repositoryAddress}resource/itemOid/com.ibm.team.BuildResult/${build
ResultUUID}
Workspace=${repositoryAddress}resource/itemOid/com.ibm.team.scm.Workspace/${
team.scm.workspaceUUID}
```

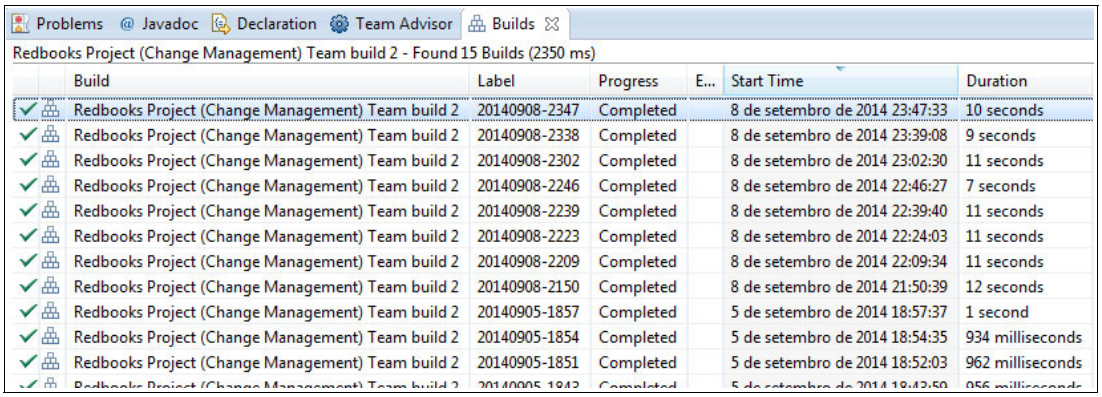
12. In the Process section:

a. Select **Deploy**.

b. Enter information about the UrbanCode application process that was created in “Application process” on page 177.

13. Click **Save**.

When using Rational Team Concert integrated with UrbanCode Deploy for continuous integration, the project leader can define a system integration role for the system integrator architect. This way, the system integrator architect can request a deployment directly, using Rational Team Concert. Additionally, all of the team members can monitor the Rational Team Concert build engines using the Rational Team Concert Eclipse plug-in (see Figure A-9, and Figure A-10 on page 235), or Rational Team Concert web dashboard widgets (Figure A-11 on page 236).



Build	Label	Progress	E...	Start Time	Duration
Redbooks Project (Change Management) Team build 2	20140908-2347	Completed		8 de setembro de 2014 23:47:33	10 seconds
Redbooks Project (Change Management) Team build 2	20140908-2338	Completed		8 de setembro de 2014 23:39:08	9 seconds
Redbooks Project (Change Management) Team build 2	20140908-2302	Completed		8 de setembro de 2014 23:02:30	11 seconds
Redbooks Project (Change Management) Team build 2	20140908-2246	Completed		8 de setembro de 2014 22:46:27	7 seconds
Redbooks Project (Change Management) Team build 2	20140908-2239	Completed		8 de setembro de 2014 22:39:40	11 seconds
Redbooks Project (Change Management) Team build 2	20140908-2223	Completed		8 de setembro de 2014 22:24:03	11 seconds
Redbooks Project (Change Management) Team build 2	20140908-2209	Completed		8 de setembro de 2014 22:09:34	11 seconds
Redbooks Project (Change Management) Team build 2	20140908-2150	Completed		8 de setembro de 2014 21:50:39	12 seconds
Redbooks Project (Change Management) Team build 2	20140905-1857	Completed		5 de setembro de 2014 18:57:37	1 second
Redbooks Project (Change Management) Team build 2	20140905-1854	Completed		5 de setembro de 2014 18:54:35	934 milliseconds
Redbooks Project (Change Management) Team build 2	20140905-1851	Completed		5 de setembro de 2014 18:52:03	962 milliseconds

Figure A-9 Reviewing build status using the Rational Team Concert Eclipse plug-in

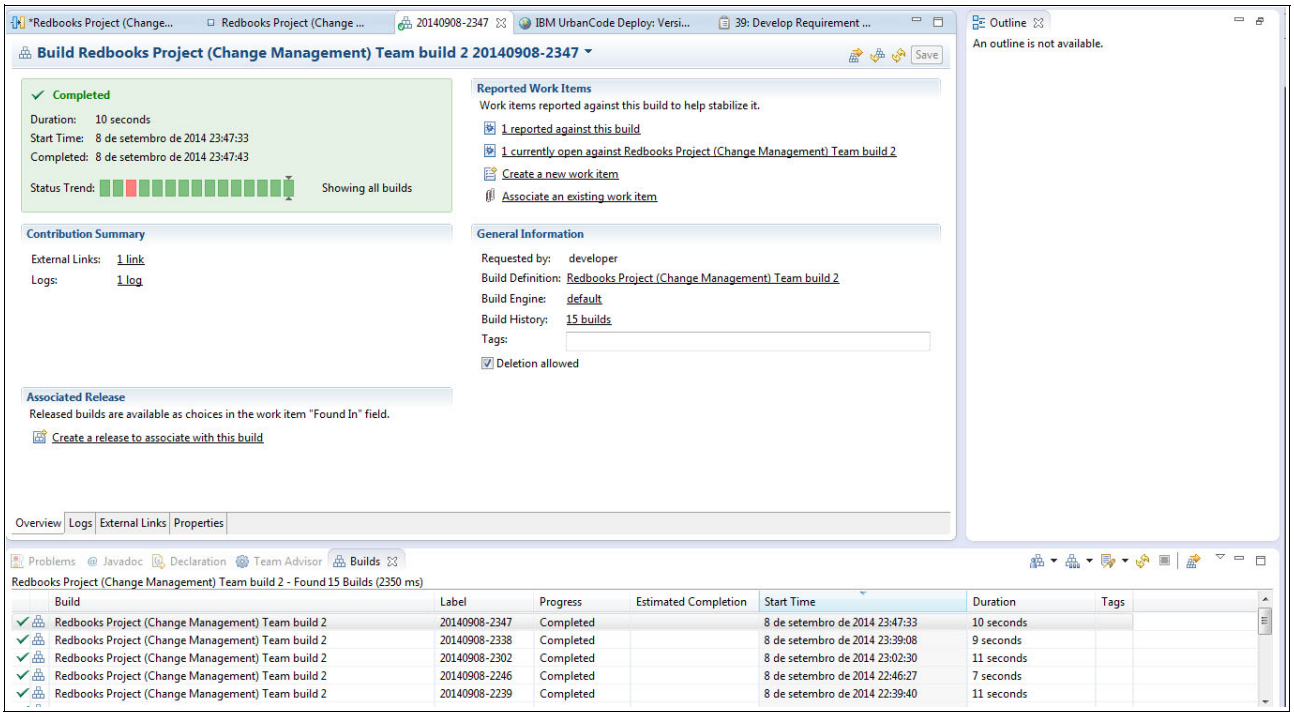


Figure A-10 Showing Build Engine Status

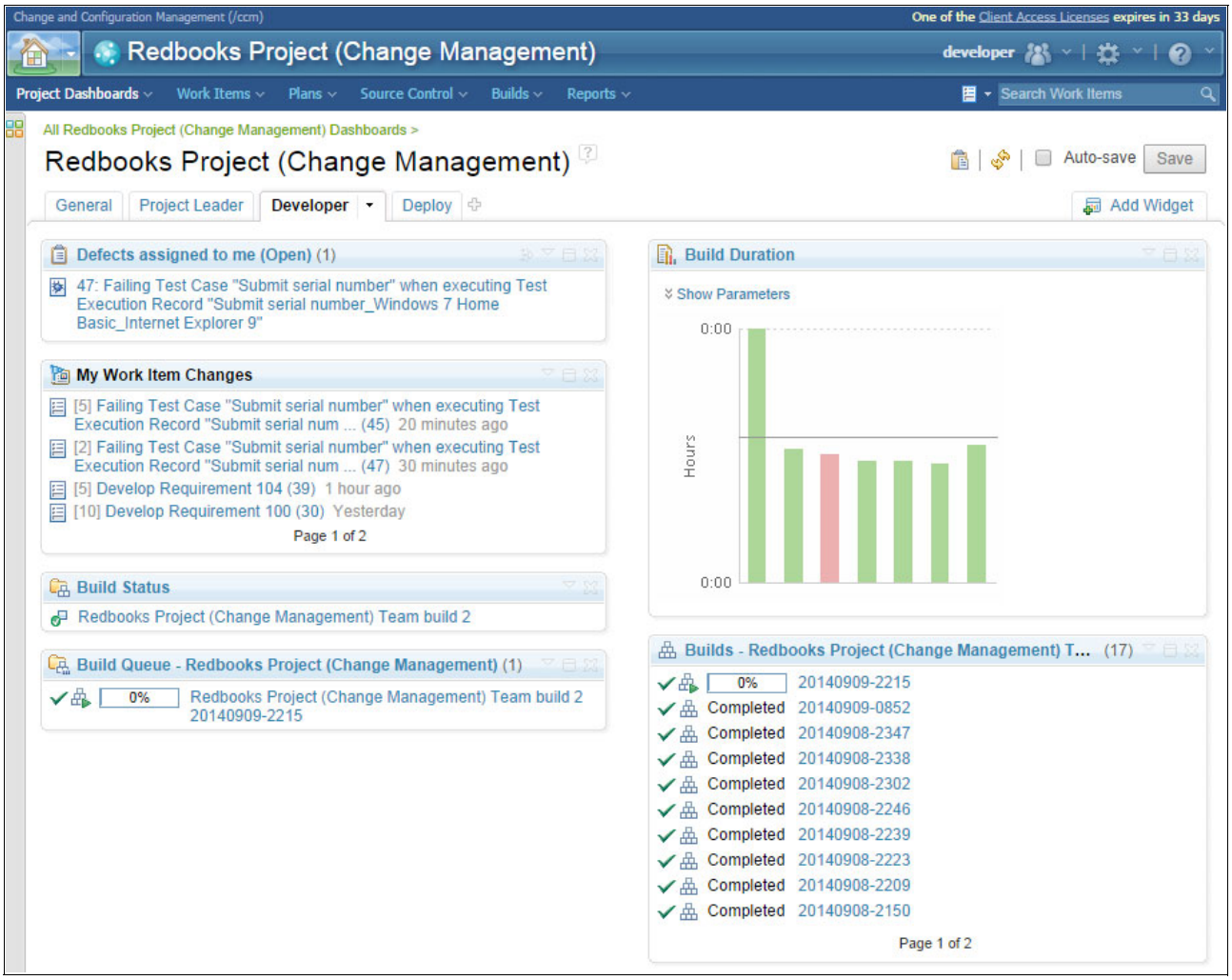


Figure A-11 Build Engine Status using Rational Team Concert web dashboard widgets

B

Importing solution applications

In this appendix, we describe how to import the integration projects and web application, as used in the scenario described in 1.2, “The business scenario” on page 14.

Download and extract the web material

Create a subdirectory on your workstation and download the contents of the web material files into this folder. Extract the `sg248243.zip` file into this folder to access the `ServiceDeskESB.zip` and `ServiceDeskWebApp.war` archive files.

Import the integration projects

The IBM Integration Bus Toolkit has to be used to import integration projects that contain the integration workflows.

To import the flows to the Integration Bus Toolkit workspace, follow these steps:

1. Select **File** → **Import** → **General** → **Existing Projects into Workspace** and click **Next**.
2. In the Import window, select the **Select archive file** and click **Browse** to point to the `ServiceDeskESB.zip` file.
3. Ensure that both the `ServiceDeskESB` and `ServiceDeskESBLib` projects are selected as shown on Figure B-1 on page 238 and click **Finish**.

© Copyright IBM Corp. 2014. All rights reserved.

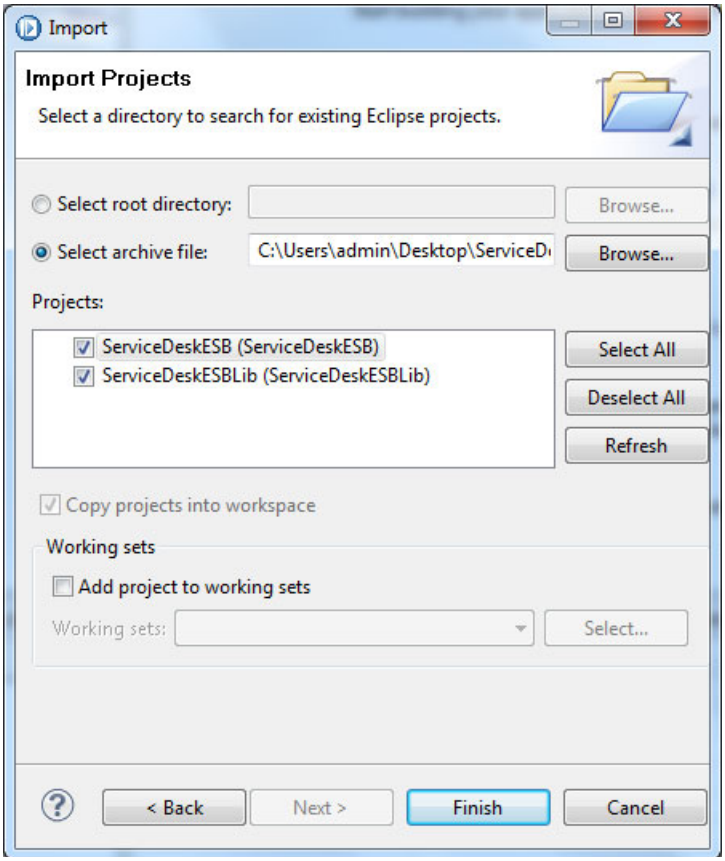


Figure B-1 Importing the integration projects using the Integration Bus Toolkit

Import the web application

Follow this procedure to import the web application using the Eclipse-based integrated development environment (IDE):

1. Click **File** → **Import** → **Java EE** → **WAR file**.
2. In the Import window, click **Browse** and point to the `ServiceDeskWebApp.war` application.
3. Select your server target runtime (for example, the Liberty profile runtime). Clear the **Add project to an EAR** check box, and click **Finish**. If you do not select any runtime, you get compilation errors.

Note: To keep track of the source code and enforce a common development policy, you can use IBM Rational Collaborative Lifecycle Management Solution tools. Appendix A, “Integrating the Eclipse environment with IBM Rational Collaborative Lifecycle Management” on page 223 shows the procedure for sharing this piece of solution using IBM Rational Collaborative Lifecycle Management.

C

Database definitions

This appendix provides the database tables used to build the solution covered in this IBM Redbooks publication.

© Copyright IBM Corp. 2014. All rights reserved.

Create entities and attributes for the business scenario

To create the solution for the Redbooks Company Service Desk scenario, the following database and entities are defined one time. The entities must be defined under a database. For this IBM Redbooks publication, `SG248243` database is created using the command below on a DB2 machine:

```
db2 create database SG248243 using codeset UTF-8 territory en
```

The entities can change in numbers according to your solution. These entities are created by the database administrators. Also, names of the entities can differ according to the solution or according to company database standards. The entities used in this Redbooks publication are:

- SAW_STATUS
- SAW_REQUESTS

Adding attributes to entities

Status entities

The attributes that are added to SAW_STATUS entity are shown in Figure C-1.

Columns

Key	Name	Data type	Length	Nullable
	STATUS_ID	BIGINT	8	No
	STATUS_NAME	CHARACTER	20	No

Figure C-1 Attributes for status table

The status table has predefined values. These values are shown in Figure C-2.

STATUS_ID	STATUS_NAME
0	Pending
1	Approved
2	Rejected

Figure C-2 Status values used in the solution

Each claim request starts with a status of *Pending*. According to the output from human task activities or the system, the status can change to *Approved* or *Rejected*.

Request entities

The attributes added to the SAW_REQUESTS entity are shown in Figure C-3.

Columns

Key	Name	Data type	Length	Nullable
	REQUEST_ID	BIGINT	8	No
	SERIAL_NUMBER	CHARACTER	20	No
	NAME	CHARACTER	100	No
	TRACKING_NUMBER	CHARACTER	20	No
	ADDRESS	CHARACTER	150	No
	STATUS_ID	BIGINT	8	No
	CREATION_TIME	TIMESTAMP	10	Yes
	REMARKS	CHARACTER	200	Yes
	COMMENTS	CHARACTER	200	Yes

Figure C-3 Attributes for the SAW_REQUESTS entity

Database entity relations

The relation of the entities is shown in Figure C-4.

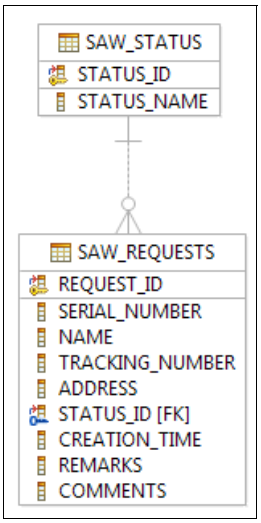


Figure C-4 Entity relations

The primary key is the *REQUEST_ID* of the requests entity. The foreign key is the *STATUS_ID* to the status entity.

Example C-1 shows the Data Definition Language (DDL) for creating the entities and inserting the data.

Example C-1 DDL for creating the entities for the solution

```
CREATE TABLE SAW_STATUS (
      "STATUS_ID" BIGINT NOT NULL,
      "STATUS_NAME" CHAR(20) NOT NULL
   )

   DATA CAPTURE NONE;

ALTER TABLE SAW_STATUS ADD CONSTRAINT "CC1408484744159" PRIMARY KEY ("STATUS_ID");

COMMENT ON TABLE SAW_STATUS IS 'SG24-8243 Redbook Status table';

insert into SAW_STATUS values (0, 'PENDING');
insert into SAW_STATUS values (1, 'APPROVED');
insert into SAW_STATUS values (2, 'REJECTED');

CREATE TABLE SAW_REQUESTS (
      "REQUEST_ID" BIGINT NOT NULL GENERATED ALWAYS AS IDENTITY ( START WITH 1
INCREMENT BY 1 MINVALUE 1 MAXVALUE 9223372036854775807 NO CYCLE NO CACHE),
      "SERIAL_NUMBER" CHAR(20) NOT NULL,
      "NAME" CHAR(100) NOT NULL,
      "TRACKING_NUMBER" CHAR(20) NOT NULL,
      "ADDRESS" CHAR(150) NOT NULL,
      "STATUS_ID" BIGINT NOT NULL,
      "CREATION_TIME" TIMESTAMP,
      "REMARKS" CHAR(200),
      "COMMENTS" CHAR(200)
   )
```

```
   DATA CAPTURE NONE COMPRESS YES;

ALTER TABLE SAW_REQUESTS ADD CONSTRAINT "CC1408485016720" PRIMARY KEY
("REQUEST_ID");

ALTER TABLE SAW_REQUESTS ADD CONSTRAINT "CC1408485022210" FOREIGN KEY
("STATUS_ID") REFERENCES SAW_STATUS("STATUS_ID");

COMMENT ON TABLE SAW_REQUESTS IS 'SG24-8243 Redbook Requests table';
```

In order to run the DDL in Example C-1 on page 241, save the DDL into a file (such as `Redbook.sql`). Connect to the `SG248243` that is created using the command with the db2 user and the password:

```
db2 connect to SG248243 user db2user using password
```

Then, you can run the DDL with the command as:

```
db2 -tvf <path>/Redbook.sql
```

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 publication provides additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- *Rapid deployment of integrated WebSphere solutions in your cloud*, REDP-5132

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 Rational Functional Tester:

  http://www-03.ibm.com/software/products/en/functional

- IBM Rational Products:

  http://jazz.net

- Open Unified Process (OpenUP):

  http://epf.eclipse.org/wikis/openup

- Deploying business rules:

  http://pic.dhe.ibm.com/infocenter/dmanager/v8r0m1/topic/com.ibm.wodm.dserver.rules.deploying/topics/wodm_dserver_rules_deploying.html

- Defining an event runtime server connection

  http://pic.dhe.ibm.com/infocenter/dmanager/v8r0m1/topic/com.ibm.wodm.dserver.events.dev/topics/tsk_dse_definingruntime_serverconn.html

- How to model processes, see *Deploying business rules:*

  http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.wle.editor.doc/modeling/topic/establishing_process_flow.html

- Process Designer:

  http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.admin.doc/managinglib/topic/adding_servers.html?lang=en

- Adding a JRules Decision Service component to a service:

  http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.wle.editor.doc/topics/building_rule_service_C.html?lang=en

© Copyright IBM Corp. 2014. All rights reserved.

- Testing a Decision service:

  http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.wle.editor.doc/topics/trule_service_testing.html?lang=en

- Setting up a user interface for your human task:

  http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.wid.bpel.doc/topics/tnclient.html?lang=en

- Understanding and using undercover agents:

  http://www-01.ibm.com/support/knowledgecenter/SSFPJS_8.0.1/com.ibm.wbpm.wle.editor.doc/modeling/topic/using_undercover_agents.html

- Deploying processes to offline servers:

  http://www-01.ibm.com/support/knowledgecenter/SSFTBX_8.0.1/com.ibm.wbpm.admin.doc/topics/releasing_installing_procs_D.html?lang=en

- UrbanCode Deploy:

  http://pic.dhe.ibm.com/infocenter/ucdeploy/v6r0/topic/com.ibm.udeploy.install.doc/topics/install_ch.html

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Creating Integrated IBM WebSphere Solutions using Application Lifecycle Management

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

Creating Integrated IBM WebSphere Solutions using Application Lifecycle Management

Creating an end to end solution using the WebSphere portfolio

Delivering effective application lifecycle management using CLM

Increasing control and agility using BPM and ODM

This IBM Redbooks publication demonstrates, through a practical solution and step-by-step implementation instructions, how customers can use the IBM Rational Application Lifecycle Management (ALM) portfolio to build and manage an integrated IBM WebSphere Application. Building a business application (mobile and desktop) that uses WebSphere Application Server, IBM MQ, IBM Integration Bus (IIB), Business Process Management (BPM), Operational Decision Management (ODM), and Mobile.

IBM Redpaper publication, *Rapid deployment of integrated WebSphere solutions in your cloud*, REDP-5132, is an extension to this IBM Redbooks publication. Using the same practical solution covered in this Redbooks publication, REDP-5132 demonstrates how the IBM PureApplication System is a “logical extension” versus a “whole new world”, covering PureApplication Patterns and the new PureApplication as a service on Softlayer. Showing how PureApplication patterns can be used to speed up the installation process of the products as described can be found at the following website:

http://www.redbooks.ibm.com/abstracts/redp5132.html?Open

The intended audience for this book is architects, developers, administrators, and DevOps personnel.

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

SG24-8243-00 ISBN 0738440280