Rational software

Rational Application Developer V7 Programming Guide

- Develop Java, Web, XML, database, EJB, Struts, JSF, SDO, EGL, Web services, and portal applications
- Test, debug, and profile with built-in and remote servers
- Deploy applications to WebSphere Application Server and WebSphere Portal

Ueli Wahli
Henry Cui
Craig Fleming
Maan Mehta
Marco Rohr
Pinar Ugurlu
Patrick Gan
Celso Gonzalez
Daniel M. Farrell
Andreas Heerdegen

Redbooks

ibm.com/redbooks

International Technical Support Organization

Rational Application Developer V7 Programming Guide

December 2007

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

First Edition (December 2007)

This edition applies to Version 7.0 of IBM Rational Application Developer for WebSphere Software.

Contents

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 give 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 Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites 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.

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

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

COPYRIGHT LICENSE:

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

Trademarks

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

ClearCase®
ClearQuest®
Cloudscape™
DB2 Universal Database™
DB2®
developerWorks®
Domino®
Extreme Blue™
IBM®
Informix®
iSeries®
OS/390®
Rational®
Rational Rose®
Rational Unified Process®
Redbooks®
Redbooks (logo) ®
RUP®
WebSphere®
XDE™
z/OS®
zSeries®

The following terms are trademarks of other companies:

Oracle, JD Edwards, PeopleSoft, Siebel, and TopLink are registered trademarks of Oracle Corporation and/or its affiliates.

Snapshot, and the Network Appliance logo are trademarks or registered trademarks of Network Appliance, Inc. in the U.S. and other countries.

AMD, the AMD Arrow logo, and combinations thereof, are trademarks of Advanced Micro Devices, Inc.

Flex, Adobe Reader, Adobe, and Portable Document Format (PDF) are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, other countries, or both.

Enterprise JavaBeans, EJB, Java, Java Naming and Directory Interface, Javadoc, JavaBeans, JavaMail, JavaScript, JavaServer, JavaServer Pages, JDBC, JDK, JMX, JNI, JRE, JSP, JVM, J2EE, J2SE, Sun, Sun Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Expression, Internet Explorer, Microsoft, SQL Server, Windows NT, Windows Server, Windows Vista, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, Pentium, Pentium 4, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

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

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

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

Preface

IBM® Rational® Application Developer for WebSphere® Software V7.0 (for short, Rational Application Developer) is the full function Eclipse 3.2 based development platform for developing Java™ 2 Platform Standard Edition (J2SE™) and Java 2 Platform Enterprise Edition (J2EE™) applications with a focus on applications to be deployed to IBM WebSphere Application Server and IBM WebSphere Portal. Rational Application Developer provides integrated development tools for all development roles, including Web developers, Java developers, business analysts, architects, and enterprise programmers.

Rational Application Developer is part of the IBM Rational Software Delivery Platform (SDP), which contains products in four life cycle categories:

- Architecture management, which includes integrated development environments (Application Developer is here)
- Change and release management
- Process and portfolio management
- Quality management

This IBM Redbooks® publication is a programming guide that highlights the features and tooling included with Rational Application Developer V7.0. Many of the chapters provide working examples that demonstrate how to use the tooling to develop applications, as well as achieve the benefits of visual and rapid application development. This publication is an update of *Rational Application Developer V6 Programming Guide*, SG24-6449.

This book consists of six parts:

- Introduction to Rational Application Developer
- Develop applications
- Test and debug applications
- Deploy and profile applications
- Team development
- Appendixes

The team that wrote this book

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

Local team

Ueli Wahli is a Consultant IT Specialist at the IBM International Technical Support Organization in San Jose, California. Before joining the ITSO over 20 years ago, Ueli worked in technical support at IBM Switzerland. He writes extensively and teaches IBM classes worldwide about WebSphere Application Server, and WebSphere and Rational application development products. In his ITSO career, Ueli has produced more than 40 IBM Redbooks. Ueli holds a degree in Mathematics from the Swiss Federal Institute of Technology.

Henry Cui is a Software Developer working at the IBM Toronto lab. Henry has been in the IBM Rational Application Developer service and support team for four years. He has helped many customers resolve design, development, and migration issues with J2EE development. Henry was a member of the WebFacing Eclipse project sponsored by the IBM Extreme Blue™ program at Almaden Research Center. Before joining IBM, Henry worked as a Software

Developer in the COGNICASE/CGI Corporation to develop J2EE applications for big financial institutions. Henry is a frequent contributor of developerWorks® articles and Rational Application Developer technotes. Henry holds a degree in Computer Science from York University.

Craig Fleming is an IT Specialist working for IBM Global Business Services in Auckland, New Zealand. He has been working in various software development roles in New Zealand and Britain for the last twelve years and spent the last six years working for IBM in customer facing roles in the Airline and Transportation industries. Craig has an honors degree in Computer Science from Otago University, New Zealand, and his areas of expertise include J2EE Web services and developing business process applications in BPEL.

Maan Mehta is a WebSphere Portal Architect. He works as an independent consultant and owns Revecore, Inc., a WebSphere Services company. Maan has over 11 years of experience in leading and delivering enterprise application development projects and also WebSphere and Portal Infrastructure projects. Most of Maan’s working experience has been with IBM Software Group (Toronto Lab) and IBM Global Services in Canada. He holds a degree in Computer Science from the Queen’s University at Kingston, Canada, and an Electronics Sciences Degree from the University of Delhi, India. His areas of expertise include analysis and design specifically with technologies such as Portal, Java EE, Flex™, and Ajax. He also has extensive experience in the administration and configuration of WebSphere Application Servers and WebSphere Portal Servers.

Marco Rohr is an Advisory IT Specialist working for IBM Global Business Services in Zurich, Switzerland. He is also an Education Specialist for the application development and WebSphere segment of the IBM training department. He has six years of experience in the enterprise application development field mainly in the public sector. He studied Computer Science at the Engineering school of Rapperswil, Switzerland and he holds a Swiss Federal Certificate in Didactic and Methodology. His areas of expertise include object-oriented analysis and design with UML, implementation of OO concepts in Java SE, and development of the presentation layer of Java EE applications.

Pinar Ugurlu is an Advisory IT Specialist in the IBM Software Group, Turkey. She has five years of experience in application design, development, and consulting. She holds a degree in Computer Engineering from Bilkent University. Her areas of expertise include J2EE programming, portals, and e-business integration. She has written extensively on J2EE development, Web development, application analysis, and application builds.

Patrick Gan is a Senior IT Specialist who works for IBM Global Services, Application Innovation Services Center of Excellence, US. He has eight years of experience and expertise in OOAD/Java Design best practices, J2EE development, SOA, software development methods, and tools. In his current capacity, Patrick's primary responsibility in customer facing engagements is solution design and delivery of custom enterprise solutions. In addition, he has also been involved in the design and development of IBM assets and has authored articles on developerWorks. Patrick has a Computer Science degree from California Polytechnic State University, Pomona.

Remote team

Celso Gonzalez has thirteen years of experience in Software Engineering, four years with IBM Rational. He is currently one of the Rational World Wide Analysis, Design and Construction leaders. His role is to provide expertise to customers and field engineers in domains ranging from business modeling to J2EE development, including requirements management, architecture, and design. Before joining the World Wide Community of Practice team, Celso was part of the Rational Unified Process® development tea, where he contributed to areas such as business modeling, requirements, analysis and design, and legacy evolution. Celso holds degrees in Mathematics, Computer Science, and Philosophy.

Daniel M. Farrell is an IBM Certified Professional and Pre-sales Engineer, from Denver, Colorado. In 1985, Daniel began working with Informix® software products at a national retail and catalog company, a job that would set his direction for the next twenty or so years. He joined IBM as a result of the acquisition of Informix. Daniel has a Masters in Computer Science from Regis University and is currently pursuing a Ph.D. from Clemson in Adult Education and Human Resource Studies. Both Daniel's thesis and dissertation research fourth generation programming languages, business application delivery, and staff development.

Andreas Heerdegen is a Senior IT Specialist at IBM Global Services in Germany. He has more than eight years experience in developing custom applications in the insurance sector. His areas of expertise include agile software development based on the WebSphere and Rational product portfolio. Andreas holds a degree in Physics from CERN, Geneva, in Switzerland.

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

- Rational Application Developer service and support team, for providing the technical support during the residency.
- Serjik Gharapet Dikalehh, IBM Software Services for WebSphere, IBM Canada, for helping with setting up and configuring IBM Rational ClearCase® LT Server v7.0.0.
- Jeff Turnham and Ye (Steven) Jin, IBM Canada, for helping with the Debugger.

- Chris Brealey, Senior Technical Staff Member, IBM Toronto lab, for providing the insights into Web services SDO facade tooling.
- Zina Mostafa and Kelvin Cheung, WebSphere Web Services Tools Development, IBM Toronto lab, for giving the skills transfer of the Web Services Feature Pack.
- Yen Lu, WebSphere Web Services Tools Development, IBM Toronto Lab, for reviewing the Web services chapter and providing valuable feedback.
- Rose Rosario, Rational Application Developer Data Tooling, IBM Software Group, for providing insights into the data model analysis.
- Yvonne Lyon, IBM ITSO, for technical editing.

Thanks to the authors of the previous editions of this book:

- Authors of the previous edition, *Rational Application Developer V6 Programming Guide*, SG24-6449, published in June 2005, were: John Ganci, Fabio Ferraz, Hari Kanangi, Kiriya Keat, George Kroner, Juha Nevalainen, Nicolai Nielsen, Richard Raszka, and Neil Weightman.
- Authors of the first edition, *WebSphere Studio Application Developer Version 5 Programming Guide*, SG24-6957, published in July 2003, were: Ueli Wahli, Ian Brown, Fabio Ferraz, Maik Schumacher, and Henrik Sjostrand.

Become a published author

Join us for a two- to six-week residency program! Help write a book dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You will have the opportunity to team with IBM technical professionals, Business Partners, and Clients.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you will develop a network of contacts in IBM development labs, and increase your productivity and marketability.

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 in one of the following ways:

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

  ibm.com/redbooks

- Send your comments in an e-mail 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

Summary of changes

This section describes the technical changes made in this edition of the book and in previous editions. This edition may also include minor corrections and editorial changes that are not identified.

Summary of changes for ***Rational Application Developer V7 Programming Guide***, **SG24-7501-00**, as created or updated on March 27, 2008.

This book is an update of the IBM Redbooks publication, ***Rational Application Developer V6 Programming Guide,*** **SG24-6449**.

December 2007, First Edition

This revision reflects the addition, deletion, or modification of new and changed information described below.

- RUP® and UML: UML diagrams for EJBs and Web services
- Develop Java applications: Eclipse 3.2, Java SE 6.0 compliance (annotations, typed collections), Cleanup wizard, improved refactoring (history)
- Accelerate development using patterns: New chapter
- Develop Java database applications: Data model reverse engineering, overview diagram, physical model analysis, compare and manage changes, impact analysis, new filter wizard, enhanced SQL statement example, Java SQL stored procedure example, SQLJ example
- Develop XML applications: XML mapping tool, XML schema editor, XML schema validator, use of Service Data Objects with XML, removed DTD examples
- Develop Web applications using JSPs and servlets: Security editor, Web Diagram, Web page template wizard, Web library project with Java model classes
- Develop Web applications using Struts: New Web Diagram Editor
- Develop Web applications using JSF and SDO: Ajax, new Web Diagram Editor, new enhanced and improved JSF components
- Develop applications using EGL: New EGL batch program, modified Web application with list-detail-update Web pages, removed lengthy introduction
- Develop Web applications using EJBs: Improved example

- Develop Web services applications: Improved Web Service wizard, new WSDL editor, Web service discovery dialog, multi-protocol binding, new WS-Security and cloning wizard, skeleton merge, JAX-RPC SDO facade, Ant tasks, Web Services Feature Pack with asynchronous client, security through RAMP policy set, Message Transmission Optimization Mechanism (MTOM), and JAX-WS Web services annotations
- Develop Portal applications: Portal tooling, portlet development using JSF and SDO, portlet client to a Web service
- Servers and server configurations: New profile creation and deletion tool, new settings in server editor, share server profiles, configure security, new Jython tooling
- JUnit testing: JUnit 4.x support, Eclipse Test and Performance Tools Platform (TPTP), automated component testing removed
- Debug local and remote applications: XSLT debugging, small changes to Variables and Debug views
- Deploy enterprise applications: Administrative scripting using Jython
- Profile applications: Integrated Agent Controller, Eclipse Test and Performance Tools Platform (TPTP)
- Rational ClearCase integration: New ClearCase LT version
- CVS integration: Latest CVS version, changes to History view, color coded Annotation view
- Product installation: Use of Installation Manager

Part 1

Introduction to Rational Application Developer

In this part of the book, we introduce IBM Rational Application Developer for WebSphere Software.

The introduction includes packaging, product features, the Eclipse base, installation, licensing, migration, and an overview of the tools.

We then discuss setting up the Workbench, the perspectives, views, and editors, and the different type of projects.

1

Introduction

IBM Rational Application Developer for WebSphere Software V7.0 is an integrated development environment and platform for building Java Platform Standard Edition (Java SE) and Java Platform Enterprise Edition (Java EE) applications with a focus on applications to be deployed to IBM WebSphere Application Server and IBM WebSphere Portal.

This chapter contains an introduction to the concepts, packaging, and features of the Rational Application Developer V7.0 product.

The chapter is organized into the following sections:

- Concepts
- Product packaging
- Product features
- Sample code

Concepts

This section provides an introduction to the IBM Rational Software Delivery Platform (SDP), Eclipse, and Application Developer.

The Rational product suite helps businesses and organizations manage the entire software development process. Software modelers, architects, developers, and testers can use the same team-unifying Rational Software Delivery Platform tooling to be more efficient in exchanging assets, following common processes, managing change and requirements, maintaining status, and improving quality.

IBM Rational Software Delivery Platform

The IBM Rational Software Delivery Platform (SDP) is not a single product, but rather an integrated set of products that share a common technology platform built on the Eclipse 3.2 framework in support of each phase of the development life cycle.

The SDP provides a team-based environment with capabilities that are optimized for the key roles of a development team including business analyst, architect, developer, tester, and deployment manager. It enables a high degree of team cohesion through shared access to common requirements, test results, software assets, and workflow and process guidance. Combined, these capabilities improve both individual and team productivity.

Figure 1-1 shows where the Application Developer product fits within the IBM Rational Software Delivery Platform product set.

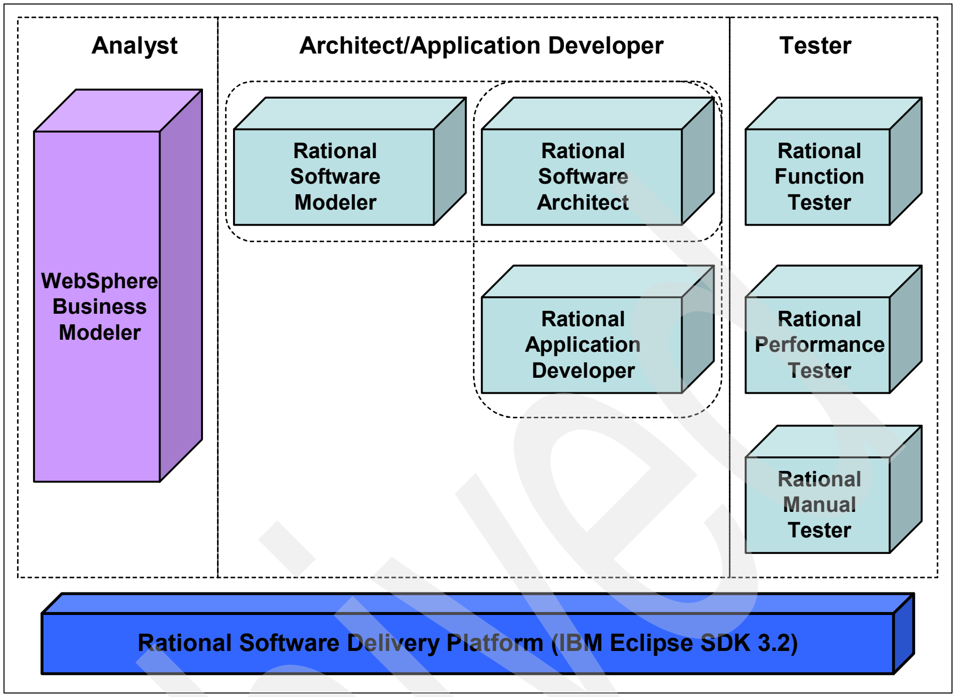

Analyst
Architect/Application Developer
Tester
WebSphere Business Modeler
Rational Software Modeler
Rational Software Architect
Rational Application Developer
Rational Function Tester
Rational Performance Tester
Rational Manual Tester
Rational Software Delivery Platform (IBM Eclipse SDK 3.2)

Figure 1-1 Rational Software Delivery Platform products

The following is a brief description of each of the products included in the IBM Rational Software Delivery Platform that share common tooling based on the IBM Eclipse SDK V3.2:

- **Rational Software Modeler**

  The Software Modeler is a UML-based visual modeling and design tool for system analysts, software architects, and designers who need to clearly define and communicate their architectural specifications to stakeholders.

  This product was known in previous releases as Rational XDE™ Modeler and is targeted at development shops where the business analyst has a distinct role of architecture and design (no development).

- **Rational Software Architect**

  The Software Architect is a design and construction tool that leverages model-driven development with UML to create well-architected applications, including those based on service-oriented architecture (SOA). It unifies modeling, Java structural review, Web services, Java SE, Java EE, database, XML, Web development, and process guidance for architects and senior developers creating applications in Java or C++.

This product was known in previous releases as Rational Rose® and Rational XDE Developer Plus for Java. Software Architect includes architecture and design capability as well as the full Java EE development functionality provided by Application Developer. This product is targeted at architects who are involved in architecture and design activities, as well as application development. If architects only need the modeling functionality, they should use the Rational Software Modeler product. Software Architect can also be used by developers or developers leads pertaining to companies following a model driven development approach.

- **Rational Application Developer**

  The Application Developer is a full suite of development, analysis, and deployment tools for rapidly implementing Java EE applications, Enterprise JavaBeans™, portlets, and Web applications. In previous releases this product was known as Application Developer V6.0 and WebSphere Studio Application Developer V5.x and is targeted at J2EE developers.

  Note: Application Developer V7.0 is the focus of this book.

- **Rational Functional Tester**

  Rational Functional Tester is an automated testing tool that tests Java, HTML, VB.NET, and Windows® applications. It provides the capability to record robust scripts that can be played back to validate new builds of an application.

- **Rational Performance Tester**

  Rational Performance Tester is a multi-user system performance test product designed to test Web applications, and focuses on scalability.

- **Rational Manual Tester**

  This manual test authoring and execution tool promotes test step reuse to reduce the impact of software change on testers and business analysts.

- **WebSphere Business Modeler**

  The WebSphere Business Modeler does not carry the Rational brand name, but is an important product of the Rational Software Delivery Platform. WebSphere Business Modeler targets the business analyst who models business processes. WebSphere Business Modeler can be used to generate Business Process Execution Language (BPEL) definitions to be imported into WebSphere Integration Developer to create applications for WebSphere Process Server. BPEL from WebSphere Business Modeler provides a more seamless move to implementation and eliminates the need to create paper diagrams for the developer. Note that WebSphere Business Modeler is currently based on Eclipse 3.0.

Version 7 terminology

As with most IBM products, the convention is to support the current version and two previous versions. As we migrate from Application Developer V6.0 to Application Developer V7.0, most of the core terminology remains the same. However, as there was a major change between WebSphere Studio Application Developer and Rational Application Developer, we have provided Table 1-1 to point out the basic terminology comparison between Version 7 (Rational Application Developer) and Version 5 (WebSphere Studio Application Developer).

Table 1-1 Terminology

Rational Application Developer Version 7	WebSphere Studio Application Developer Version 5
Rational Developer **Note**: Used to describe development products built on common Eclipse base	WebSphere Studio
Rational Application Developer (known as Application Developer)	WebSphere Studio Application Developer
IBM Eclipse SDK 3.2 **Note**: IBM branded and value-added version of Eclipse SDK 3.2	WebSphere Studio Workbench (IBM supported Eclipse 2.x)
Workbench	Workbench
IBM Rational Software Delivery Platform **Note**: Used to describe product set built on common Eclipse 3.2 platform	N/A

Application development challenges

To better grasp the business value that Application Developer V7.0 provides, it is important to understand the challenges businesses face in application development.

Table 1-2 highlights the key application development challenges as well as desired development tooling solutions.

Table 1-2 Application development challenges

Challenges	Solution tooling
Application development is complex, time consuming, and error prone.	Raise productivity by automating time consuming and error prone tasks.
Highly skilled developers are required and in short supply.	Assist less knowledgeable developers where possible by providing wizards, on-line context sensitive help, an integrated environment and visual tooling.
Learning curves are long.	Shorten learning curves by providing Rapid Application Development (RAD) tooling (visual layout and design, re-usable components, code generators) and ensure development tools have consistent way of working.

Key themes of Version 7

There are many very significant enhancements and features in Application Developer V7.0. We have listed the key themes of Version 7 tooling:

- **Open computing**:
  - Stay current with the latest open standards such as Eclipse 3.2, JDK™ 5, Axis, JavaServer™ Faces 1.1, and so forth.
- **Consumability**:
  - Improve the overall experience of trying, buying, downloading, installing/deploying, running, and servicing the product line
  - Provide the ability to migrate from the trial version to the purchased one without uninstalling
- **Compliance**:
  - Optional enforcement of floating licenses
- **SOA**:
  - Create a more generalized approach to SOA
  - Improve the Web services tooling
    - New Web Service wizard
    - New WSDL visual editor

We provide more detail on the these new features in “Summary of new features in Version 7” on page 12, as well as throughout the chapters of this book.

Product packaging

This section highlights the product packaging for Application Developer V7.0.

Rational Developer supported platforms and databases

This section describes the platforms and databases supported by the Rational Developer products.

Supported operating system platforms

Application Developer V7.0 supports the following operating systems:

  - Microsoft® Windows XP Professional with Service Pack 1 or 2
  - Windows XP Professional x64 Edition (running on an AMD™ processor)
  - Microsoft Windows 2000 Professional with Service Pack 4
  - Microsoft Windows 2000 Server with Service Pack 4
  - Microsoft Windows 2000 Advanced Server with Service Pack 4
  - Microsoft Windows Server® 2003 Standard Edition with Service Pack 1
  - Microsoft Windows Server 2003 Enterprise Edition with Service Pack 1
  - Microsoft Windows Vista® (Application Developer Version 7.0.0.2 or later)

- Linux® on Intel®:
  - Red Hat Enterprise Linux Workstation Version 4.0
  - Red Hat Desktop Version 4.0 (running in 32-bit mode)
  - SUSE Linux Enterprise Server (SLES) Version 9 (all service packs; running in 32-bit mode)

The IBM Rational Agent Controller included with Application Developer V7.0 is supported on many platforms running WebSphere Application Server. For details refer to the *Installation Guide, IBM Rational Application Developer V7.0* product guide (install.html) found on the Application Developer V7 Setup CD 1.

Supported databases

Application Developer V7.0 supports the following database products:

- IBM Cloudscape™ V5.1
- IBM DB2® Universal Database™ V7.2, V8.1, V8.2, V9.1
- IBM DB2 Universal Database for iSeries® V5R2, V5.3, V5.4
- IBM DB2 Universal Database for z/OS® and OS/390® V7
- IBM DB2 Universal Database for z/OS V8
- Apache Derby 10.0, 10.1
- Generic JDBC™ 1.0
- Informix Dynamic Server V9.2, V9.3, V9.4, V10.0
- MySQL 4.0, V4.1

- Microsoft SQL Server™ 2000 and Microsoft SQL Server 2005
- Oracle® 8i V8.1.7, Oracle 9i, Oracle 10g
- Sybase Adaptive Server Enterprise V12.x, V15

Application Developer V7 product packaging

Table 1-3 lists the software CDs included with Application Developer V7.0.

Table 1-3 Application Developer V7.0 product packaging

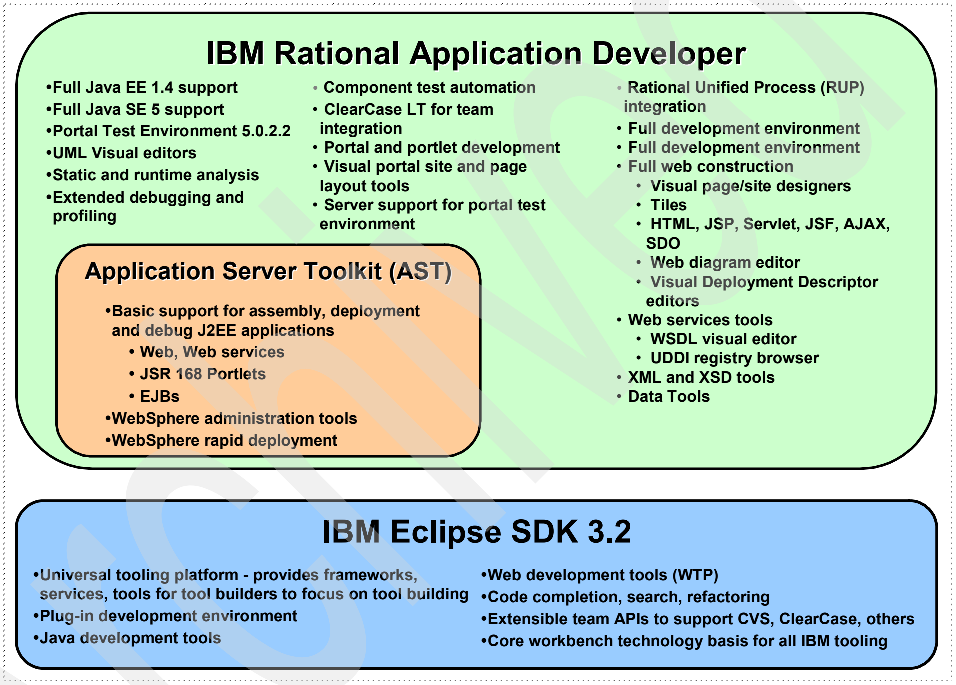
CDs	Windows	Linux
Application Developer V7.0 (10 CDs) Included test WebSphere Application Server environments: ► WebSphere Application Server 5.1 Express ► WebSphere Application Server 5.1 ► WebSphere Application Server 6.0 ► WebSphere Application Server 6.1	X	X
IBM WebSphere Application Server Developer V6.1	X	X
IBM Rational Business Developer Extension (EGL)	X	X
IBM WebSphere Portal V5.1.0.3	X	X
IBM WebSphere Portal V6.0	X	X
IBM Rational Agent Controller **Note**: Support for many additional platforms	X	X
IBM Rational ClearCase LT **Note**: Web download	Web	Web
Crystal Enterprise V11 Professional Edition	X	X
Crystal Enterprise V11 Embedded Edition	X	na
IBM DB2 Universal Database V9.1 Express Edition	X	X

Product features

This section gives you a summary of the new features of Application Developer V7. We provide more detailed information on the new features throughout the chapters of this book.

Figure 1-2 displays a summary of features found in the IBM Rational Developer V7.0 products. We have organized the description of the product features into the following topics:

- Summary of new features in Version 7
- Specification versions
- Eclipse and IBM Rational Software Delivery Platform
- Test server environments
- Installation and licensing
- Migration and coexistence
- Tools



IBM Rational Application Developer

- Full Java EE 1.4 support
- Full Java SE 5 support
- Portal Test Environment 5.0.2.2
- UML Visual editors
- Static and runtime analysis
- Extended debugging and profiling

- Component test automation
- ClearCase LT for team integration
- Portal and portlet development
- Visual portal site and page layout tools
- Server support for portal test environment

- Rational Unified Process (RUP) integration
- Full development environment
- Full development environment
- Full web construction
  - Visual page/site designers
  - Tiles
  - HTML, JSP, Servlet, JSF, AJAX, SDO
  - Web diagram editor
  - Visual Deployment Descriptor editors
- Web services tools
  - WSDL visual editor
  - UDDI registry browser
- XML and XSD tools
- Data Tools

Application Server Toolkit (AST)

- Basic support for assembly, deployment and debug J2EE applications
  - Web, Web services
  - JSR 168 Portlets
  - EJBs
- WebSphere administration tools
- WebSphere rapid deployment

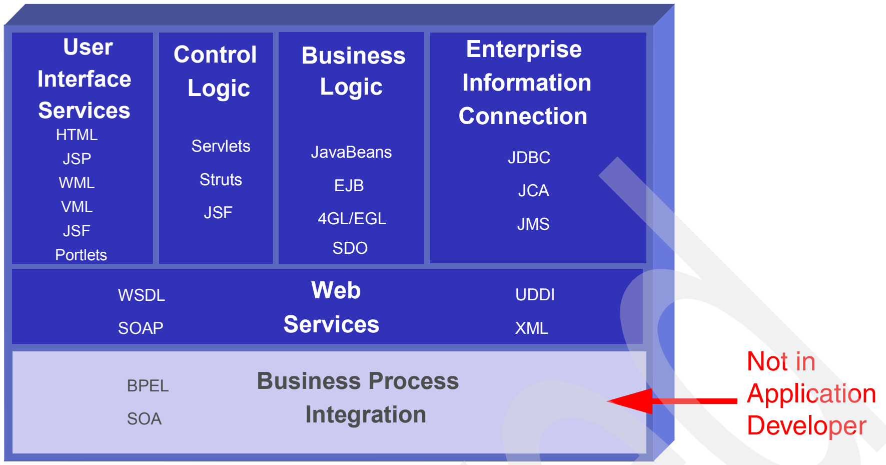
IBM Eclipse SDK 3.2

- Universal tooling platform - provides frameworks, services, tools for tool builders to focus on tool building
- Plug-in development environment
- Java development tools

- Web development tools (WTP)
- Code completion, search, refactoring
- Extensible team APIs to support CVS, ClearCase, others
- Core workbench technology basis for all IBM tooling

Figure 1-2 Tools and features summary

Figure 1-3 provides a summary of technologies supported by Application Developer categorized by the applications component.



Not in Application Developer

Figure 1-3 Supported technologies for developing applications

Summary of new features in Version 7

There are lots of new features in Version 7, many of which we highlight in detail in the remaining chapters of this book. The objective of this section is to summarize the new features in Application Developer V7.0:

- Specification versions: Full support for Java EE V1.4, Java SE V5.0 and IBM WebSphere Application Server V6.1. See "Specification versions" on page 14 for more information on new features.
- Eclipse and IBM Rational Software Delivery Platform: Based on Eclipse 3.2. See "Eclipse and IBM Rational Software Delivery Platform" on page 16 for more information on these new features.
- Application Developer V7.0 supports Java 5. There is tooling for such features as annotations, generics, enums, static import, and variable arguments.
- Web tooling:
  - The Web Diagram Editor is rewritten to leverage the Graphical Modeling Framework (GMF).
  - Drag and drop functionality (from the Palette) in the Web Diagram Editor updates the diagram and (behind the scenes) generates appropriate code (keeping diagram and code in-sync).
- JavaServer Faces (JSF):
  - Full support for JSF 1.1.
  - New version of the IBM JSF Widget Library (JWL) including AJAX-like behavior.

  - Support for JSF portlet bridge.
  - Support for *standard JSF only mode* (which excludes usage of IBM-specific JSF components) as well as support for third party JSF components.
  - Support for multiple faces configuration files.

  For more detailed information and a programming example, refer to Chapter 14, “Develop Web applications using JSF and SDO” on page 587.

- Portal application development:
  - Application Developer V7.0 includes portal development tooling, as well as integrated test environments for WebSphere Portal V5.1 and V6.0.
  - Support for Web Services for Remote Portlet (WSRP).
  - Wizard support for cooperative portlets.
  - Enhanced credential vault support.
  - Support for business process portlet.

  For more detailed information and a programming example, refer to Chapter 19, “Develop portal applications” on page 897.

- Test server environments:
  - Test environments included for WebSphere Application Server V6.1, V6.0, V5.1, WebSphere Portal V6.0, V5.1, and WebSphere Express V5.1.
  - Integration with IBM WebSphere Application Server V6.0 for deployment, testing, and administration is the same (test environment, separate install, and Network Deployment edition).

  See “Test server environments” on page 19 for more information on new features.

- XML:
  - Updated support for XML and XSLT tooling.
  - Updated support for XML schema editing, including visual modeling.
  - Updated support for XML schema to Java code generation.
- Web services:
  - Series of usability improvements in Web services development (improved skeleton merge for top-down Web service creation, simplified editing of WSDL and XML schema, remote WSDL validation).
  - Complex schema support with SDO.
  - Enhanced support for XSD.
  - Support for WSDL and XSD modeling.

For more detailed information and a programming example, refer to Chapter 18, “Develop Web services applications” on page 811.

- Model driven development:
  - Support JET transformations.
  - Updates to general features (such as Project Explorer, model import, and model template).

  For more detailed information and a programming example, refer to Chapter 6, “RUP and UML” on page 173.
- Rational Unified Process (RUP) integration:
  - Process Browser provides search capability for RUP best practices and life cycle for development.
  - Process Advisor view displays RUP information specific to current task.
  - This feature is offered in Application Developer and Rational Software Architect.

  For more detailed information and a programming example, refer to Chapter 6, “RUP and UML” on page 173.
- Debugging:
  - Support for debugging WebSphere Jython administration scripts.
  - Support for DB2 V9.0 Stored Procedure Debug.
- Additional enhancements:
  - Database tools: Many enhancements around DB2 V9.0, support for virtual tables and support for triggers/referential constraints.

Note: The EGL tooling, which previously came as part of the Rational Software Delivery Platform, is now an extension to the platform called Rational Business Developer Extension (RBDe). This extension is acquired separately from the base software install.

Specification versions

This section highlights the specification versions found in Application Developer V7.0, which supports development for the Java Platform Enterprise Edition V1.4. The only differences between Application Developer V6.0 and Application Developer V7.0 are in the Java SE version (now supporting V5.0) and JavaServer Faces (now supporting V1.1).

Table 1-4 includes a comparison of the J2EE specification versions. Table 1-5 includes a comparison of the WebSphere Application Server specification versions.

Table 1-4 J2EE specification versions

Specification	Application Developer V7.0	Application Developer V6.0
IBM Java Runtime Environment (JRE™)	**1.5**	1.4.2
JavaServer Pages™ (JSP™)	2.0	2.0
Java Servlet	2.4	2.4
Enterprise JavaBeans (EJB™)	2.1	2.1
Java Message Service (JMS)	1.1	1.1
Java Transaction API (JTA)	1.0	1.0
JavaMail™	1.3	1.3
Java Activation Framework (JAF)	1.0	1.0
Java API for XML Processing (JAXP)	1.2	1.2
J2EE Connector	1.5	1.5
Java API for XML RPC (JAX-RPC)	1.1	1.1
SOAP with Attachments API for Java (SAAJ)	1.2	1.2
Web services	1.1	1.1
Web Services Feature Pack with JAX-WS 2.0, JAXB 2.0, SOAP 1.2, SAAJ 1.3	optional feature	not supported
Java Authentication and Authorization Service (JAAS)	1.2	1.2
Java API for XML Registries (JAXR)	1.0	1.0
J2EE Management API	1.0	1.0
Java Management Extensions (JMX™)	1.2	1.2
J2EE Deployment API	1.1	1.1
Java Authorization Service Provider Contract for Containers (JAAC)	1.0	1.0

Table 1-5 WebSphere Application Server specification versions

Specification	Application Developer V7.0	Application Developer V6.0
JavaServer Faces (JSF)	**1.1**	1.0 (JSR 127)
Service Data Objects (SDO)	1.0	1.0
Struts	1.1	1.1

Eclipse and IBM Rational Software Delivery Platform

This section provides an overview of the Eclipse Project, as well as how Eclipse relates to the IBM Rational Software Delivery Platform and Application Developer V7.0.

Eclipse Project

The Eclipse Project is an open source software development project devoted to creating a development platform and integrated tooling. Figure 1-4 shows the high-level Eclipse Project architecture and shows the relationship of the following sub projects:

- Eclipse Platform
- Eclipse Java Development Tools (JDT)
- Eclipse Plug-in Development Environment (PDE)

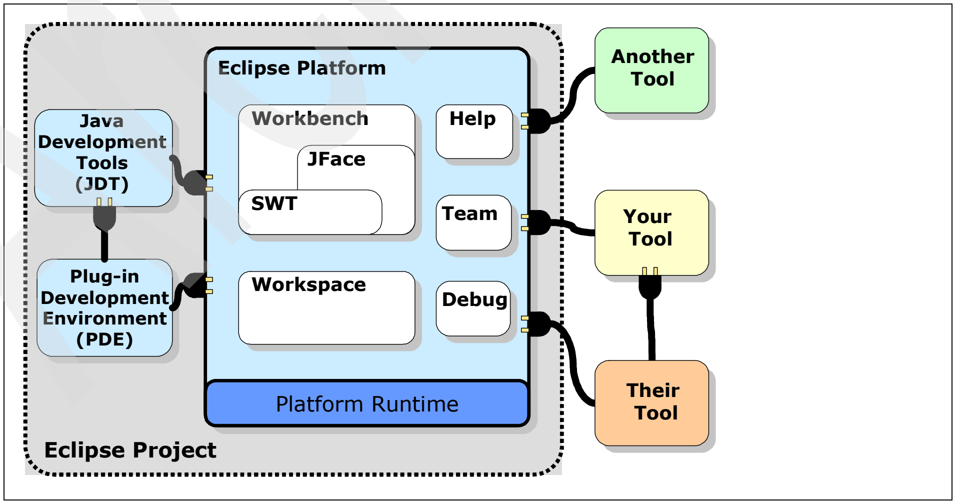


Figure 1-4 Eclipse Project overview

With a common public license that provides royalty free source code and world-wide redistribution rights, the Eclipse Platform provides tool developers with great flexibility and control over their software technology.

Industry leaders like IBM, Borland, Merant, QNX Software Systems, RedHat, SuSE, TogetherSoft, and WebGain formed the initial eclipse.org board of directors of the Eclipse open source project.

More detailed information on Eclipse can be found at:

http://www.eclipse.org

Eclipse Platform

The Eclipse Platform provides a framework and services that serve as a foundation for tools developers to integrate and extend the functionality of the platform. The platform includes a workbench, concept of projects, user interface libraries (JFace, SWT), built-in help engine, and support for team development and debug. The platform can be leveraged by a variety of software development purposes including modeling and architecture, integrated development environment (Java, C/C++, Cobol), testing, and so forth.

Eclipse Java Development Tools (JDT)

The JDT provides the plug-ins for the platform specifically for a Java-based integrated development environment, as well as the development of plug-ins for Eclipse. The JDT adds the concepts of Java projects, perspectives, views, editors, wizards, and refactoring tools to extend the platform.

Eclipse Plug-in Development Environment (PDE)

The PDE provides the tools to facilitate the development of Eclipse plug-ins.

Eclipse Software Developer Kit (SDK)

The Eclipse SDK consists of the software created by the Eclipse Project (Platform, JDT, PDE), which can be licensed under the Eclipse Common Public License agreement, as well as other open source third-party software licensed separately.

Note: The Eclipse SDK does not include a Java Runtime Environment (JRE) and must be obtained separately and installed for Eclipse to run.

IBM Eclipse SDK V3.2

The IBM Eclipse SDK V3.2 is an IBM branded and value added version of the Eclipse SDK V3.2. The IBM Eclipse SDK V3.2 includes additional plug-ins and the IBM Java Runtime Environment (JRE) V1.5 and V1.4.2.

The IBM Eclipse SDK V3.2 is highly desirable for many reasons:

- It offers the ability for loose or tight integration of tooling with the platform as well as industry standard tools and repositories.
- It provides frameworks, services, and tools that enable tool builders to focus on tool building, not on building tool infrastructure. ISVs can develop new tools to take advantage of the infrastructure and integration points for seamless integration between the tools and the SDK.
- It provides flexibility for rapid support for new standards and technologies (for example, Web services).
- It provides a consistent way of representing and maintaining objects.
- The tools have been designed to support a roles-based development model in which the assets created in one tool are consistent for other tools (for example, XML created in one tool and used in another).
- It provides support for the full life cycle, including test, debug, and deployment.

The following components are integrated into the IBM Eclipse SDK V3.2:

- Eclipse SDK 3.2 (Platform, JDT, PDE)
- Eclipse Modeling Framework (EMF)
- Eclipse Hyades
- C/C++ Development Tooling (CDT)
- Graphical Editing Framework (GEF)
- XML Schema Infoset Model (XSD)
- UML 2.0 Metamodel Implementation (UML2)
- IBM Java Runtime Environment (JRE) V1.5 and V1.4.2

In summary, the IBM Eclipse SDK V3.2, which provides an open, portable, and universal tooling platform, serves as the base for the IBM Rational Software Delivery Platform common to many Rational products, including Application Developer V7.0.

Test server environments

Application Developer V7.0 supports a wide range of server environments for running, testing, and debugging application code.

Since Application Developer V6.0, the IBM WebSphere Application Server integrated testing environment is the actual full blown installation of IBM WebSphere Application Server. In previous versions of WebSphere Studio, the test environment used was a scaled down version of WebSphere Application Server. As said above, Application Developer V7.0 ships with WebSphere Application Server V5.1, V6.0, and V6.1 test environments.

We have categorized the test server environments (Table 1-6) as follows:

- Integrated test servers—Refers to the test servers included with the Application Developer.
- Test servers available separately—Refers to the test servers that are supported by the Application Developer, but are available separately from the Rational Developer products.

Table 1-6 Test server environments

Installation option	Test server
IBM WebSphere Application Server V6.x	IBM WebSphere Application Server V6.1
	IBM WebSphere Application Server V6.0
IBM WebSphere Application Server V5.x	IBM WebSphere Application Server V5.1
	IBM WebSphere Application Server Express V5.1
IBM WebSphere Portal	IBM WebSphere Portal V5.1
	IBM WebSphere Portal V6.0
Test servers available separately	Apache Tomcat V3.2, V4.0, V4.1, V5.0, V5.5
	BEA WebLogic V8.1, V9.0, V9.2
	JBoss V3.2.3, V4.0
	ObjectWeb JOnAS V4.0
	Oracle OC4J Standalone Server V10.1.3

Installation and licensing

This section provides a summary for licensing, installation, product updates, and uninstallation for Application Developer V7.0. Licensing, installation, product updates, and uninstallation are now achieved through the IBM Installation Manager (which is installed along with Application Developer V7.0). It is important to note that whenever using IBM Installation Manager on a previously installed product, that particular product must not be in use when making updates.

Installation

This section provides a summary of the new installation architecture, and the key software and hardware requirements.

All Version 7.0 products include the Rational Software Delivery Platform and plug-ins. If the Rational Software Delivery Platform is found during the installation; only the plug-ins for the new product are installed. Products that contain the functionality of an installed product as well as additional plug-ins will upgrade the installed product. Products with less functionality than the installed products will be blocked from installation.

The system hardware requirements are as follows:

- Pentium® III 800 MHz or higher
- 1 GB RAM minimum
- 1024x768 video resolution or higher
- 2 GB or free hard disk space to install all components.
- The temporary (/tmp Linux or c:\temp Windows) should have at least 500 MB of free space. Additional disk space is required for the resources that you develop.

Installation steps: The detailed steps to perform installation of Application Developer V7 are described in “Installing IBM Rational Application Developer” on page 1283.

Licensing

Acquiring and activating licenses in Application Developer V7.0 has changed since the prior release.

To start, there are various types of licenses:

- Authorized user—These keys are to be used on an individual machine by any user but can only be used on the machine to which they are registered.
- Floating licenses—These keys are for a single software product that can be shared among multiple team members. The total number of concurrent users cannot exceed the number of floating licenses purchased.
- Named user licenses—These are similar to floating licenses and can be used with the Rational License Server. Named user licenses are associated to specific user IDs; this restricts the licenses to being checked out only by specific users. The user IDs are listed in the Rational options file (`rational.opt`) on the License Server. The number of licenses registered must be equal to the number of user IDs in the list.

All licenses can be in one of the following forms:

- Permanent—A permanent license is one that has been purchased and does not expire.
- Temporary—A temporary license is meant to expire in a short period of time. Our temporary licenses expire in fifteen (15) or thirty (30) days.
- Term license agreement (TLA)—A TLA license expires but acts more like a permanent license. It has an issue date and expiration date. (All Rational internal licenses are TLA.)

All licenses can be moved from machine to machine at any time.

License administration is handled by the Rational License Key Center (RLKC). For information on the RLKC, visit this Web site:

```
https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=rational
```

This new way of managing licenses by importing activation kits allows you to upgrade from a trial version to a purchased one by importing a new activation kit, without having to uninstall the trial version.

For more information on licensing, refer to the following file:

```
<IBMInstallMgr>\Installation Manager\eclipse\documentation\readme.html
```

License installation: Once you have obtained a license, invoke the IBM Installation Manager. The detailed steps to import a license are described in "Installing the license for Rational Application Developer" on page 1291.

Updates

Once Application Developer has been installed, the Installation Manager provides and interface to update the product. This replaces the Rational Product Updater provided in Application Developer V6.0.

Refer to “Updating Rational Application Developer” on page 1293 for detailed instructions.

Uninstall

Application Developer V7.0 can be uninstalled interactively through the IBM Installation Manager.

Refer to “Uninstalling Rational Application Developer” on page 1294 for detailed instructions.

Migration and coexistence

This section highlights the Application Developer V7.0 migration features, and the coexistence and compatibility with Application Developer V6.0 and WebSphere Studio.

Migration

Migration includes the migration of Application Developer V6.0 as well as J2EE projects and code assets.

Application Developer V7.0 includes the ability to migrate Application Developer V6.0 edition installations. Projects exported from Application Developer V6.0, such as EARs, WARs, JARs, or with project interchange zip files, are automatically migrated when imported into Application Developer V7.0. Once modified into Application Developer V7.0, the artifacts can no longer be modified in prior version of the IDE, they can only be open as read-only.

Compatibility with previous versions

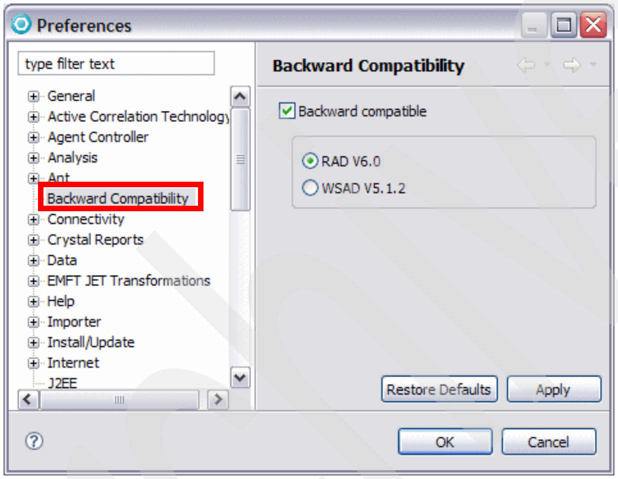
Application Developer V7.0 includes a compatibility option to facilitate sharing projects with Application Developer V6.0 and WebSphere Studio V5.1.x.

Projects can be shared with Application Developer V6.0 developers or WebSphere Studio developers through a SCM (such as ClearCase or CVS) or using project interchange zip files.

Metadata and project structure are not updated to the new Application Developer format. A `.compatibility` file is added to projects and is used to track the timestamps of resources.

Compatibility support can be removed when finished developing in a mixed environment.

The compatibility feature can be accessed by selecting **Windows** → **Preferences** → **Backward compatibility** (Figure 1-5).



☑ Backward compatible

● RAD V6.0

○ WSAD V5.1.2

Figure 1-5 Backward compatibility feature

Migration considerations

Migration information is available on specific components:

- Migrating J2EE projects from J2EE 1.3 to J2EE 1.4
- Migrating JavaServer Faces resources in a Web Project
- Migrating JavaServer Faces resources with Faces Client Components
- WDO to SDO migration
- Portal Applications with and without JavaServer Faces
- Considerations around using the Debugger
- EGL reserved words

Previous installations of the IBM Agent Controller should be uninstalled before installing a new version.

Tools

Application Developer V7.0 includes a wide array of tooling to simplify or eliminate tedious and error-prone tasks, and provide the ability for Rapid Web Development. We have listed the key areas of tooling included with Application Developer:

- Java development tools (JDT)
- Relational database tools
- XML tools
- Web development tools
- Struts tools
- JSF development tools
- SDO development tools
- Enterprise Generation Language tools
- EJB tools
- Portal tools
- Web services tools
- Team collaboration tools
- Debugging tools
- Performance profiling and analysis tools
- Server configuration tools
- Testing tools
- Deployment tools
- Plug-in development tools

> **Note:** Each of the chapters of this book provides a description of the tools related to the given topic and demonstrates how to use the Application Developer tooling.

Sample code

The chapters are written so that you can follow along and create the code from scratch. In places where there is lots of typing involved, we have provided snippets of code to cut and paste.

Alternatively, you can import the completed sample code from a project interchange file. For details on the sample code (download, unpack, description, import interchange file, create databases), refer to Appendix B, “Additional material” on page 1307.

Important: Fix Pack 7.0.0.2 and 7.0.0.3

Much of this book was originally researched and written using Application Developer V7.0. During the course of completing this book, we upgraded the system to 7.0.0.2, or even 7.0.0.3 for the Web Services Feature Pack. In some cases, our samples require at least 7.0.0.2 for the sample to run properly.

For more information, refer to the following link, which contains information on the latest fixes available for Application Developer:

`http://www-306.ibm.com/software/awdtools/developer/application/support/index.html`

Summary

This chapter introduced the concepts behind Application Developer and gave an overview of the features of the various members of the Rational product suite, and where Application Developer fits with regard to the other products. A summary of the version numbers of the various features was given and the Installation Manager was introduced.

2

Programming technologies

This chapter describes a number of example application development scenarios, based on a simple banking application. Throughout these examples, we will review the Java and supporting technologies, as well as highlight the tooling provided by Application Developer V7.0, which can be used to facilitate implementing the programming technologies.

This chapter is organized into the following sections:

- Desktop applications
- Static Web sites
- Dynamic Web applications
- Enterprise JavaBeans
- J2EE Application Clients
- Web services
- Messaging systems

Desktop applications

By *desktop applications* we mean applications in which the application runs on a single machine and the user interacts directly with the application using a user interface on the same machine.

When this idea is extended to include database access, some work might be performed by another process, possibly on another machine. Although this begins to move us into the client-server environment, the application is often only using the database as a service—the user interface, business logic, and control of flow are still contained within the desktop application. This contrasts with full client-server applications in which these elements are clearly separated and might be provided by different technologies running on different machines.

This type of application is the simplest type we will consider. Many of the technologies and tools involved in developing desktop applications, such as the Java editor and the XML tooling, are used widely throughout all aspects of Application Developer.

The first scenario deals with a situation in which a bank requires an application to allow workers in a bank call center to be able to view and update customer account information. We will call this the Call Center Desktop.

Simple desktop applications

A starting point for the Call Center Desktop might be a simple stand-alone application designed to run on desktop computers.

Java 2 Platform Standard Edition (Java SE) provides all the elements necessary to develop such applications. It includes, among other elements, a complete object-oriented programming language specification, a wide range of useful classes to speed development, and a runtime environment in which programs can be executed. In this chapter, in order to be inline with the latest support of Java EE level (1.4) in Application Developer V7.0, all discussion of Java SE will be at version 1.4. Although, it is to be noted that Application Developer V7.0 supports Java SE V5.0.

The complete J2SE specification can be found at:

```
http://java.sun.com/j2se/
```

Java language

Java is a general purpose, object-oriented language. The basic language syntax is similar to C and C++, although there are significant differences. Java is a higher-level language than C or C++, in that the developer is presented with a more abstracted view of the underlying computer hardware and is not expected to take direct control of issues such as memory management. The compilation process for Java does not produce directly executable binaries, but rather an intermediate byte code, which can be executed directly by a virtual machine or further processed by a just-in-time compiler at runtime to produce platform-specific binary output.

Core Java APIs

With Java SE V5.0 release, the updates and new features comprise an Ease of Development theme. Here are some of the features introduced in this release:

- **Meta data**: Meta data provides the ability to associate data at a class, method, or field level. This meta data can then be discovered by the compiler at runtime and can serve virtually any purpose. One justification for using meta data is to help reduce the effort for development and/or deployment.
- **Generics**: Generic type is one of the more anticipated features in Java SE 5.0. The first place to receive generic type support is in the collections API. In summary, the collections API provides common functionality such as linked lists, array lists, and hash maps that can store more than one Java type. The downside to this approach is that type mismatches cannot be detected at compile time. The first notification of an issue is a `ClassCastException` at runtime. The generic type feature aims to solve this problem by allowing type safety at compile time.
- **Autoboxing**: In the Java language, a primitive has corresponding primitive wrapper classes that provide convenience methods specific to that primitive. Converting between primitive types and their corresponding wrapper class counterparts often requires unnecessary amounts of extra coding. The autoboxing and auto-unboxing is a construct that automatically handles the conversion between primitives and primitive wrapper classes (and vice versa).
- **Enhanced for loop**: The `Iterator` class is used heavily by the collections API. Specifically, it provides a mechanism to navigate sequentially through a collection. For this intent, the enhanced *for loop* can now replace the iterator.
- **Static imports**: Static import feature allows the reference of static constants from a class without needing to inherit from it.
- **Variable arguments**: The varargs functionality allows multiple arguments to be passed as parameters to methods.

For more information on the aforementioned Java SE V5.0 features, refer to the following JSRs:

- JSR 201 (http://jcp.org/en/jsr/detail?id=201): Contains four of these language changes; enhanced for loop, enumerated types, static import and autoboxing.

- JSR 175 (http://jcp.org/en/jsr/detail?id=175): Specifies the new meta data functionality.
- JSR 14 (http://jcp.org/en/jsr/detail?id=14): Details generic types.

Java Virtual Machine

The Java Virtual Machine (JVM™) is a runtime environment designed for executing compiled Java byte code, contained in the .class files, which result from the compilation of Java source code. Several different types of JVM exist, ranging from simple interpreters to just-in-time compilers that dynamically translate byte code instructions to platform-specific instructions as required.

Requirements for the development environment

The developer of the Call Center Desktop should have access to a development tool, providing a range of features to enhance developer productivity:

- A specialized code editor, providing syntax highlighting
- Assistance with completing code and correcting syntactical errors
- Facilities for visualizing the relationships between the classes in the application
- Assistance with documenting code
- Automatic code review functionality to ensure that code is being developed according to recognized best practices
- A simple way of testing applications

Application Developer V7.0 provides developers with an integrated development environment with these features.

Database access

It is very likely that the Call Center Desktop will have to access data residing in a relational database, such as IBM DB2 Universal Database.

Java SE V1.4 includes several integration technologies:

- JDBC is the Java standard technology for accessing data stores.
- Java Remote Method Invocation (RMI) is the standard way of enabling remote access to objects within Java.
- Java Naming and Directory Interface™ (JNDI) is the standard Java interface for naming and directory services.

- Java IDL is the Java implementation of the Interface Definition Language (IDL) for the Common Object Request Broker Architecture (CORBA), allowing Java programs to access objects hosted on CORBA servers.

We focus on the Java DataBase Connectivity (JDBC) technology in this section.

JDBC

Java SEV1.4 includes JDBC V3.0. In earlier versions of Java SE, the classes contained in the `jaxa.sql` package were known as the JDBC Core API, whereas those in the `javax.sql` package were known as the JDBC Standard Extension API (or Optional Package), but now the V1.4 JDBC includes both packages as standard. Since Java 2 Platform Enterprise Edition V1.4 (Java EE V1.4, which we will come to shortly) is based on Java SE V1.4, all these features are available when developing Java EE V1.4 applications as well.

More information on JDBC can be found at:

`http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/`

Although JDBC supports a wide range of data store types, it is most commonly used for accessing relational databases using SQL. Classes and interfaces are provided to simplify database programming, such as:

- `java.sql.DriverManager` and `javax.sql.DataSource` can be used to obtain a connection to a database system.
- `java.sql.Connection` represents the connection that an application has to a database system.
- `java.sql.Statement`, `PreparedStatement`, and `CallableStatement` represent executable statements that can be used to update or query the database.
- `java.sql.ResultSet` represents the values returned from a statement that has queried the database.
- Various types such as `java.sql.Date` and `java.sql.Blob` are Java representations of SQL data types that do not have a directly equivalent primitive type in Java.

Requirements for the development environment

The development environment should provide access to all the facilities of JDBC V3.0. However, since JDBC V3.0 is an integral part of Java SE V1.4, this requirement has already been covered in “Simple desktop applications” on page 28. In addition, the development environment should provide:

- A way of viewing information about the structure of an external database
- A mechanism for viewing sample contents of tables

- Facilities for importing structural information from a database server so that it can be used as part of the development process
- Wizards and editors allowing databases, tables, columns, relationships, and constraints to be created or modified
- A feature to allow databases created or modified in this way to be exported to an external database server
- A wizard to help create and test SQL statements

These features would allow developers to develop test databases and work with production databases as part of the overall development process. They could also be used by database administrators to manage database systems, although they might prefer to use dedicated tools provided by the vendor of their database systems.

IBM Rational Application Developer V7.0 includes these features.

Graphical user interfaces

A further enhancement of the Call Center Desktop would be to make the application easier to use by providing a graphical user interface (GUI).

Abstract Window Toolkit (AWT)

The Abstract Window Toolkit (AWT) is the original GUI toolkit for Java. It has been enhanced since it was originally introduced, but the basic structure remains the same. The AWT includes the following items:

- A wide range of user interface components, represented by Java classes such as [`java.awt.`] `Frame`, `Button`, `Label`, `Menu`, and `TextArea`.
- An event-handling model to deal with events such as button clicks, menu choices, and mouse operations
- Classes to deal with graphics and image processing
- Layout manager classes to help with positioning components in a GUI
- Support for drag-and-drop functionality in GUI applications

The AWT is implemented natively for each platform’s JVM. AWT interfaces typically perform relatively quickly and have the same look-and-feel as the operating system, but the range of GUI components that can be used is limited to the lowest common denominator of operating system components, and the look-and-feel cannot be changed.

More information on the AWT can be found at:

http://java.sun.com/j2se/1.4.2/docs/guide/awt/

Swing

Swing is a newer GUI component framework for Java. It provides Java implementations of the components in the AWT and adds a number of more sophisticated GUI components, such as tree views and list boxes. For the basic components, Swing implementations have the same name as the AWT component with a J prefix and a different package structure, for example, `java.awt.Button` becomes `javax.swing.JButton` in Swing.

Swing GUIs do not normally perform as quickly as AWT GUIs, but have a richer set of controls and have a pluggable look-and-feel.

More information on Swing can be found at:

http://java.sun.com/j2se/1.4.2/docs/guide/swing/

Standard Widget Toolkit

The Standard Widget[1] Toolkit (SWT) is the GUI toolkit provided as part of the Eclipse Project and used to build the Eclipse GUI itself. The SWT is written entirely in Java and uses the Java Native Interface (JNI™) to pass the calls through to the operating system where possible. This is done to avoid the *lowest common denominator* problem. The SWT uses native calls where they are available and builds the component in Java where they are not.

In many respects, the SWT provides the best of both worlds (AWT and Swing):

- It has a rich, portable component model, like Swing.
- It has the same look-and-feel as the native operating system, like the AWT.
- GUIs built using the SWT perform well, like the AWT, since most of the components simply pass through to operative system components.

A disadvantage of the SWT is that, unlike the AWT and Swing, it is not a standard part of Java SE V1.4. Consequently, any application that uses the SWT has to be installed along with the SWT class libraries. However, the SWT, like the rest of the components that make up Eclipse, is open source and freely distributable under the terms of the Common Public License.

More information on the SWT can be found at:

http://www.eclipse.org/swt/

[1] In the context of windowing systems, a widget is a reusable interface component, such as a menu, scroll bar, button, text box, or label.

Java components providing a GUI

There are two types of Java components that might provide a GUI:

- Stand-alone Java applications: Launched in their own process (JVM). This category would include Java EE Application Clients, which we will come to later.
- Java applets: Normally run in a JVM provided by a Web browser or a Web browser plug-in.

An applet normally runs in a JVM with a very strict security model, by default. The applet is not allowed to access the file system of the machine on which it is running and can only make network connections back to the machine from which it was originally loaded. Consequently, applets are not normally suitable for applications that require access to databases, since this would require the database to reside on the same machine as the Web server. If the security restrictions are relaxed, as might be possible if the applet was being used only on a company intranet, this problem is not encountered.

An applet is downloaded on demand from the Web site that is hosting it. This gives an advantage in that the latest version is automatically downloaded each time it is requested, so distributing new versions is trivial. On the other hand, it also introduces disadvantages in that the applet will often be downloaded several times even if it has not changed, pointlessly using bandwidth, and the developer has little control over the environment in which the applet will run.

Requirements for the development environment

The development environment should provide a specialized editor that allows a developer to design GUIs using a variety of component frameworks (such as the AWT, Swing, or the SWT). The developer should be able to focus mainly on the visual aspects of the layout of the GUI, rather than the coding that lies behind it. Where necessary, the developer should be able to edit the generated code to add event-handling code and business logic calls. The editor should be dynamic, reflecting changes in the visual layout immediately in the generated code and changes in the code immediately in the visual display. The development environment should also provide facilities for testing visual components that make up a GUI, as well the entire GUI.

Application Developer V7.0 includes a visual editor for Java classes, which offers this functionality.

Extensible Markup Language (XML)

Communication between computer systems is often difficult because different systems use different data formats for storing data. XML has become a common way of resolving this problem.

It can be desirable for a Call Center Desktop application to be able to exchange data with other applications. For example, we might want to be able to export tabular data so that it can be read into a spreadsheet application to produce a chart, or we might want to be able to read information about a group of transactions that can then be carried out as part of an overnight batch operation.

A convenient technology for exchanging information between applications is XML, which is a standard, simple, flexible way of exchanging data. The structure of the data is described in the XML document itself, and there are mechanisms for ensuring that the structure conforms to an agreed format—these are known as Document Type Definitions (DTDs) and XML schemas (XSDs).

XML is increasingly also being used to store configuration information for applications. For example, many aspects of Java EE V1.4 use XML for configuration files called *deployment descriptors*, and WebSphere Application Server V6 uses XML files for storing its configuration settings.

For more information on XML, see the home of XML—the World Wide Web Consortium (W3C) at:

`http://www.w3c.org/XML/z`

Using XML in Java code

Java SE V1.4 includes the Java API for XML Processing (JAXP). JAXP contains several elements:

- A parser interface based on the Document Object Model (DOM) from the W3C, which builds a complete internal representation of the XML document
- The Simple API for XML Parsing (SAX), which allows the document to be parsed dynamically using an event-driven approach
- XSL Transformations (XSLT), which uses Extensible Stylesheet Language (XSL) to describe how to transform XML documents from one form into another

Since JAXP is a standard part of Java SE V1.4, all these features are available in any Java code running in a JVM.

Requirements for the development environment

In addition to allowing developers to write code to create and parse XML documents, the development environment should provide features that allow developers to create and edit XML documents and related resources. In particular:

- An XML editor that will check the XML document for well-formedness (conformance with the structural requirements of XML) and for consistency with a DTD or XML Schema
- Wizards for:
  - Creating XML documents from DTDs and XML schemas
  - Creating DTDs and XML schemas from XML documents
  - Converting between DTDs and XML schemas
  - Generating JavaBeans to represent data stored in XML documents
  - Creating XSL
- An environment to test and debug XSL transformations

Application Developer V7.0 includes all these features.

Static Web sites

A static Web site is one in which the content viewed by users accessing the site using a Web browser is determined only by the contents of the file system on the Web server machine. Because the user's experience is determined only by the content of these files and not by any action of the user or any business logic running on the server machine, the site is described as static.

In most cases, the communication protocol used for interacting with static Web sites is the Hypertext Transfer Protocol (HTTP).

In the context of our sample scenario, the bank might want to publish a static Web site in order to inform customers of bank services, such as branch locations and opening hours, and to inform potential customers of services provided by the bank, such as account interest rates. This kind of information can safely be provided statically, since it is the same for all visitors to the site and it changes infrequently.

Hypertext Transfer Protocol (HTTP)

HTTP follows a request/response model. A client sends an HTTP request to the server providing information about the request method being used, the requested Uniform Resource Identifier (URI), the protocol version being used, various other header information and often other details, such as details from a form completed on the Web browser. The server responds by returning an HTTP response consisting of a status line, including a success or error code, and other header information followed by a the HyperText Markup Language (HTML) code for the static page requested by the client.

Full details of HTTP can be found at:

`http://www.w3c.org/Protocols/`

Information on HTML can be found at:

`http://www.w3c.org/MarkUp/`

Methods

HTTP 1.1 defines several request methods: GET, HEAD, POST, PUT, DELETE, OPTIONS, and TRACE. Of these, only GET and POST are commonly used in Web applications:

- GET requests are normally used in situations where the user has entered an address into the address or location field of a Web browser, used a bookmark or favorite stored by the browser, or followed a hyperlink within an HTML document.
- POST requests are normally used when the user has completed an HTML form displayed by the browser and has submitted the form for processing. This request type is most often used with dynamic Web applications, which include business logic for processing the values entered into the form.

Status codes

The status code returned by the server as the first line of the HTTP response indicates the outcome of the request. In the event of an error, this information can be used by the client to inform the user of the problem. In some situations, such as redirection to another URI, the browser will act on the response without any interaction from the user. The classes of status code are:

- 1xx: Information—The request has been received and processing is continuing.
- 2xx: Success—The request has been correctly received and processed; an HTML page accompanies a 2xx status code as the body of the response.
- 3xx: Redirection—The request did not contain all the information required or the browser needs to take the user to another URI.
- 4xx: Client error—The request was incorrectly formed or could not be fulfilled.
- 5xx: Server error—Although the request was valid, the server failed to fulfill it.

The most common status code is `200` (`OK`), although `404` (`Not Found`) is very commonly encountered. A complete list of status codes can be found at the W3C site mentioned above.

Cookies

Cookies are a general mechanism that server-side connections can use to both store and retrieve information on the client side of the connection. Cookies can contain any piece of textual information, within an overall size limit per cookie of 4 kilobytes. Cookies have the following attributes:

- Name: The name of the cookie.
- Value: The data that the server wants passed back to it when a browser requests another page.
- Domain: The address of the server that sent the cookie and that receives a copy of this cookie when the browser requests a file from that server. The domain can be set to equal the subdomain that contains the server so that multiple servers in the same subdomain receive the cookie from the browser.
- Path: Used to specify the subset of URLs in a domain for which the cookie is valid.
- Expires: Specifies a date string that defines the valid lifetime of that cookie.
- Secure: Specifies that the cookie is only sent if HTTP communication is taking place over a secure channel (known as HTTPS).

A cookie's life cycle proceeds as follows:

- The user gets connected to a server that wants to record a cookie.
- The server sends the name and the value of the cookie in the HTTP response.
- The browser receives the cookie and stores it.
- Every time the user sends a request for a URL at the designated domain, the browser sends any cookies for that domain that have not expired with the HTTP request.
- Once the expiration date has been passed, the cookie crumbles.

Non-persistent cookies are created without an expiry date—they will only last for the duration of the user's browser session. Persistent cookies are set once and remain on the user's hard drive until the expiration date of the cookie. Cookies are widely used in dynamic Web applications, which we address later in this chapter, for associating a user with server-side state information.

More information on cookies can be found at:

`http://wp.netscape.com/newsref/std/cookie_spec.html`

HyperText Markup Language (HTML)

HTML is a language for publishing hypertext on the Web. HTML uses tags to structure text into headings, paragraphs, lists, hypertext links, and so forth. Table 2-1 lists some of the basic HTML tags.

Table 2-1 Some basic HTML tags

Tag	Description
`<html>`	Tells the browser that the following text is marked up in HTML. The closing tag </html> is required and is the last tag in your document.
`<head>`	Defines information for the browser that might or might not be displayed to the user. Tags that belong in the <head> section are <title>, <meta>, <script>, and <style>. The closing tag </head> is required.
`<title>`	Displays the title of your Web page, and is usually displayed by the browser at the top of the browser pane. The closing tag </title> is required.
`<body>`	Defines the primary portion of the Web page. Attributes of the <body> tag enables setting of the background color for the Web pages, the text color, the link color, and the active and visited link colors. The closing tag </body> is required.

Cascading style sheets (CSS)

Although Web developers can use HTML tags to specify styling attributes, the best practice is to use a cascading style sheet (CSS). A CSS file defines a hierarchical set of style rules that the creator of an HTML (or XML) file uses in order to control how that page is rendered in a browser or viewer, or how it is printed.

CSS allows for separation of presentation content of documents from the content of documents. A CSS file can be referenced by an entire Web site to provide continuity to titles, fonts, and colors.

Below is a rule for setting the H2 elements to the color red. Rules are made up of two parts: *Selector* and *declaration*. The selector (H2) is the link between the HTML document and the style sheet, and all HTML element types are possible selectors. The declaration has two parts: Property (color) and value (red):

```
H2 { color: red }
```

More information on CSS can be found at:

http://www.w3.org/Style/CSS/

Requirements for the development environment

The development environment should provide:

- An editor for HTML pages, providing WYSIWYG (*what you see is what you get*), HTML code, and preview (browser) views to assist HTML page designers
- A CSS editor
- A view showing the overall structure of a site as it is being designed
- A built-in Web server and browser to allow Web sites to be tested

IBM Rational Application Developer V7.0 provides all of these features.

Dynamic Web applications

By *Web applications* we mean applications that are accessed using HTTP (Hypertext Transfer Protocol), usually using a Web browser as the client-side user interface to the application. The flow of control logic, business logic, and generation of the Web pages for the Web browser are all handled by software running on a server machine. Many different technologies exist for developing this type of application, but we will focus on the Java technologies that are relevant in this area.

Since the technologies are based on Java, most of the features discussed in , “Desktop applications” on page 28, are relevant here as well (the GUI features are less significant). In this section we focus on the additional features required for developing Web applications.

In the context of our example banking application, thus far we have provided workers in the bank’s call center with a desktop application to allow them to view and update account information and members of the Web browsing public with information about the bank and its services. We will now move into the Internet banking Web application, called *RedBank* in this document. We want to extend the system to allow bank customers to access their account information online, such as balances and statements, and to perform some transactions, such as transferring money between accounts and paying bills.

Simple Web applications

The simplest way of providing Web-accessible applications using Java is to use Java servlets and JavaServer Pages (JSPs). These technologies form part of the Java Enterprise Edition (Java EE), although they can also be implemented in systems that do not conform to the Java EE specification, such as Apache Jakarta Tomcat:

http://jakarta.apache.org/tomcat/

Information on these technologies (including specifications) can be found at the following locations:

- Servlets:

  http://java.sun.com/products/servlet/

- JSPs:

  http://java.sun.com/products/jsp/

In this book we discuss Java EE V1.4, since this is the version supported by Application Developer V7.0 and IBM WebSphere Application Server V6.0. Java EE V1.4 requires Servlet V2.4 and JSP V2.0. Full details of Java EE V1.4 can be found at:

http://java.sun.com/j2ee/

Servlets

A *servlet* is a Java class that is managed by server software known as a *Web container* (sometimes referred to as a *servlets container* or *servlets engine*). The purpose of a servlet is to read information from an HTTP request, perform some processing, and generate some dynamic content to be returned to the client in an HTTP response.

The Servlet Application Programming Interface includes a class, `javax.servlet.http.HttpServlet`, which can be subclassed by a developer. The developer needs to override methods such as the following to handle different types of HTTP requests (in these cases, POST and GET requests; other methods are also supported):

```
public void doPost (HttpServletRequest request,
                    HttpServletResponse response)
public void doGet (HttpServletRequest request,
                   HttpServletResponse response)
```

When a HTTP request is received by the Web container, it consults a configuration file, known as a *deployment descriptor,* to establish which servlets class corresponds to the URL provided. If the class is already loaded in the Web container and an instance has been created and initialized, the Web container invokes a standard method on the servlets class:

```
public void service (HttpServletRequest request,
                     HttpServletResponse response)
```

The service method, which is inherited from HttpServlet, examines the HTTP request type and delegates processing to the doPost or doGet method as appropriate. One of the responsibilities of the Web container is to package the HTTP request received from the client as an HttpServletRequest object and to create an HttpServletResponse object to represent the HTTP response that will ultimately be returned to the client.

Within the doPost or doGet method, the servlet developer can use the wide range of features available within Java, such as database access, messaging systems, connectors to other systems, or Enterprise JavaBeans.

If the servlet has not already been loaded, instantiated, and initialized, the Web container is responsible for carrying out these tasks. The initialization step is performed by executing the method:

```
public void init ()
```

And there is a corresponding method:

```
public void destroy ()
```

This is called when the servlet is being unloaded from the Web container.

Within the code for the doPost and doGet methods, the usual processing pattern is:

- Read information from the request. This often includes reading cookie information and getting parameters that correspond to fields in an HTML form.
- Check that the user is in the appropriate state to perform the requested action.
- Delegate processing of the request to the appropriate type of business object.
- Update the user's state information.
- Dynamically generate the content to be returned to the client.

The last step could be carried out directly in the servlet code by writing HTML to a PrintWriter object obtained from the HttpServletResponse object:

```
PrintWriter out = response.getWriter();
out.println("<html><head><title>Page title</title></head>");
out.println("<body>The page content:");
// etc...
```

This approach is not recommended, because the embedding of HTML within the Java code means that HTML page design tools, such as those provided by Rational Application Developer, cannot be used. It also means that development roles cannot easily be separated—Java developers must maintain HTML code.

The best practice is to use a dedicated display technology, such as JSP, covered next.

JavaServer Pages (JSPs)

JSPs provide a server-side scripting technology that enables Java code to be embedded within Web pages, so JSPs have the appearance of HTML pages with embedded Java code. When the page is executed, the Java code can generate dynamic content to appear in the resulting Web page. JSPs are compiled at runtime into servlets that execute to generate the resulting HTML. Subsequent calls to the same JSP simply execute the compiled servlet.

JSP scripting elements (some of which are shown in Table 2-2) are used to control the page compilation process, create and access objects, define methods, and manage the flow of control.

Table 2-2 Examples of JSP scripting elements

Element	**Meaning**
Directive	Instructions that are processed by the JSP engine when the page is compiled to a servlet `<%@ ... %>` or `<jsp:directive.page ... />`
Declaration	Allows variables and methods to be declared `<%! ... %>` or `<jsp:declaration> ... </jsp:declaration>`
Expression®	Java expressions, which are evaluated, converted to a String and entered into the HTML `<%= ... %>` or `<jsp:expression ... />`
Scriptlet	Blocks of Java code embedded within a JSP `<% ... %>` or `<jsp:scriptlet> ... </jsp:scriptlet>`
Use Bean	Retrieves an object from a particular scope or creates an object and puts it into a specified scope `<jsp:useBean ... />`
Get Property	Calls a getter method on a bean, converts the result to a String, and places it in the output `<jsp:getProperty ... />`
Set Property	Calls a setter method on a bean `<jsp:setProperty ... />`
Include	Includes content from another page or resource `<jsp:include ... />`
Forward	Forwards the request processing to another URL `<jsp:forward ... />`

The JSP scripting elements can be extended, using a technology known as *tag extensions* (or *custom tags*), to allow the developer to make up new tags and associate them with code that can carry out a wide range of tasks in Java. Tag extensions are grouped in *tag libraries*, which we will discuss shortly.

Some of the standard JSP tags are only provided in an XML-compliant version, such as `<jsp:useBean ... />`. Others are available in both traditional form (for example, `<%= ... %>` for JSP expressions) or XML-compliant form (for example, `<jsp:expression ... />`). These XML-compliant versions have been introduced in order to allow JSPs to be validated using XML validators.

JSPs generate HTML output by default—the Multipurpose Internet Mail Extensions (MIME) type is text/html. It might be desirable to produce XML (text/xml) instead in some situations. For example, a developer might want to produce XML output, which can then be converted to HTML for Web browsers, Wireless Markup Language (WML) for wireless devices, or VoiceXML for systems with a voice interface. Servlets can also produce XML output in this way—the content type being returned is set using a method on the `HttpServletResponse` object.

Tag libraries

Tag libraries are a standard way of packaging tag extensions for applications using JSPs.

Tag extensions address the problem that arises when a developer wishes to use non-trivial processing logic within a JSP. Java code can be embedded directly in the JSP using the standard tags described above. This mixture of HTML and Java makes it difficult to separate development responsibilities (the HTML/JSP designer has to maintain the Java code) and makes it hard to use appropriate tools for the tasks in hand (a page design tool will not provide the same level of support for Java development as a Java development tool). This is essentially the reverse of the problem described when discussing servlets above. To address this problem, developers have documented the *View Helper* design pattern, as described in *Core J2EE Patterns: Best Practices and Design Strategies* by Crupi, et al. The pattern catalog contained in this book is also available at:

`http://java.sun.com/blueprints/corej2eepatterns/Patterns/`

Tag extensions are the standard way of implementing View Helpers for JSPs. Using tag extensions, a Java developer can create a class that implements some view-related logic. This class can be associated with a particular JSP tag using a tag library descriptor (TLD). The TLD can be included in a Web application, and the tag extensions defined within it can then be used in JSPs. The JSP designer can use these tags in exactly the same way as other (standard) JSP tags. The JSP specification includes classes that can be used as a basis for tag extensions and (new in JSP v2.0) a simplified mechanism for defining tag extensions that does not require detailed knowledge of Java.

Many convenient tags are provided in the JSP Standard Tag Library (JSTL), which actually includes several tag libraries:

- Core tags: Flow control (such as loops and conditional statements) and various general purpose actions.
- XML tags: Allow basic XML processing within a JSP.
- Formatting tags: Internationalized data formatting.
- SQL tags: Database access for querying and updating.
- Function tags: Various string handling functions.

Tag libraries are also available from other sources, such as those from the Jakarta Taglibs Project (`http://jakarta.apache.org/taglibs/`), and it is also possible to develop tag libraries yourself.

Expression Language

Expression Language (EL) was originally developed as part of the JSTL, but it is now a standard part of JSP (from V2.0). EL provides a standard way of writing expressions within a JSP using implicit variables, objects available in the various scopes within a JSP and standard operators. EL is defined within the JSP V2.0 specification.

Filters

Filters are objects that can transform a request or modify a response. They can process the request before it reaches a servlet, and/or process the response leaving a servlet before it is finally returned to the client. A filter can examine a request before a servlet is called and can modify the request and response headers and data by providing a customized version of the request or response object that wraps the real request or response. The deployment descriptor for a Web application is used to configure specific filters for particular servlets or JSPs. Filters can also be linked together in chains.

Life cycle listeners

Life cycle events enable listener objects to be notified when servlet contexts and sessions are initialized and destroyed, as well as when attributes are added or removed from a context or session.

Any listener interested in observing the `ServletContext` life cycle can implement the `ServletContextListener` interface, which has two methods, `contextInitialized` (called when an application is first ready to serve requests) and `contextDestroyed` (called when an application is about to shut down).

A listener interested in observing the `ServletContext` attribute life cycle can implement the `ServletContextAttributesListener` interface, which has three methods, `attributeAdded` (called when an attribute is added to the `ServletContext`), `attributeRemoved` (called when an attribute is removed from the `ServletContext`), and `attributeReplaced` (called when an attribute is replaced by another attribute in the `ServletContext`).

Similar listener interfaces exist for `HttpSession` and `ServletRequest` objects:

- `javax.servlet.http.HttpSessionListener`: `HttpSession` life cycle events.
- `javax.servlet.HttpSessionAttributeListener`: Attributes events on an `HttpSession`.
- `javax.servlet.HttpSessionActivationListener`: Activation or passivation of an `HttpSession`.
- `javax.servlet.HttpSessionBindingListener`: Object binding on an `HttpSession`.
- `javax.servlet.ServletRequestListener`: Processing of a `ServletRequest` has begun.
- `javax.servlet.ServletRequestAttributeListener`: Attribute events on a `ServletRequest`.

Requirements for the development environment

The development environment should provide:

- Wizards for creating servlets, JSPs, listeners, filters, and tag extensions
- An editor for JSPs that enables the developer to use all the features of JSP in an intuitive way, focussing mainly on page design
- An editor for Web deployment descriptors allowing these components to be configured
- Validators to ensure that all the technologies are being used correctly
- A test environment that will allow dynamic Web applications to be tested and debugged

Application Developer V7.0 includes all these features.

Figure 2-1 shows the interaction between the Web components and a relational database, as well as the desktop application discussed in “Desktop applications” on page 28.

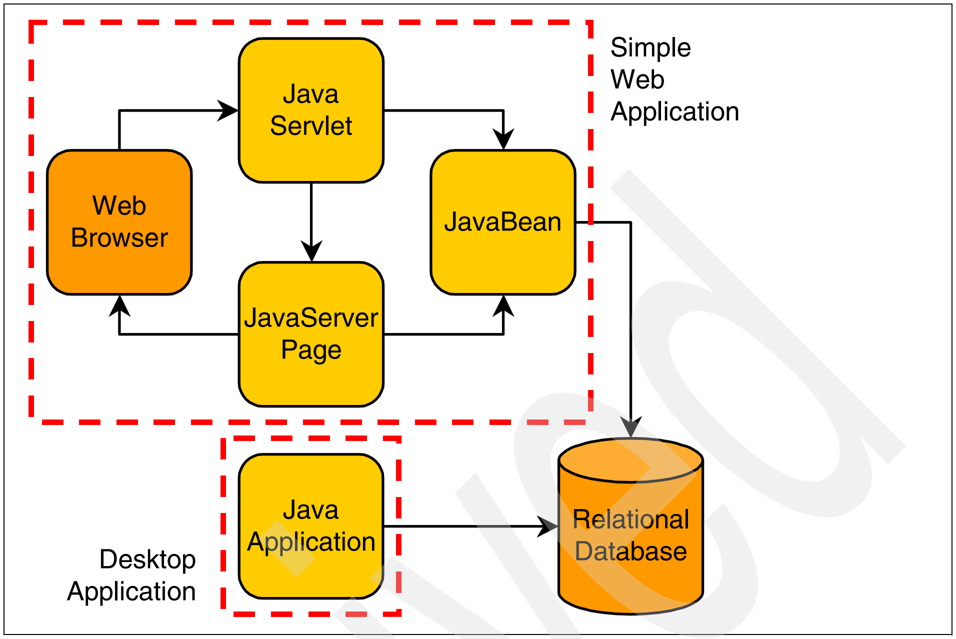


Figure 2-1 Simple Web application

Struts

The model-view-controller (MVC) architecture pattern is used widely in object-oriented systems as a way of dividing applications into sections with well-defined responsibilities:

- **Model**: Manages the application domain's concepts, both behavior and state. It responds to requests for information about its state and responds to instructions to change its state.
- **View**: Implements the rendering of the model, displaying the results of processing for the use, and manages user input.
- **Controller**: Receives user input events, determines which action is necessary, and delegates processing to the appropriate model objects.

In dynamic Web applications, the servlet normally fills the role of *controller*, the JSP fills the role of *view* and various components, and JavaBeans or Enterprise JavaBeans fill the role of *model*. The MVC pattern is described in more detail in Chapter 13, "Develop Web applications using Struts" on page 541, as will Struts.

In the context of our banking scenario, this technology does not relate to any change in functionality from the user's point of view. The problem being addressed here is that, although many developers might want to use the MVC pattern, Java EE V1.4 does not provide a standard way of implementing it. The developers of the RedBank Web application want to design their application according to the MVC pattern, but do not want to have to build everything from the ground up.

Struts was introduced as a way of providing developers with an MVC framework for applications using the Java Web technologies—servlets and JSPs. Complete information on Struts is available at:

http://struts.apache.org/

Struts provides a controller servlets, called `ActionServlet`, which acts as the entry point for any Struts application. When the `ActionServlet` receives a request, it uses the URL to determine the requested action and uses an `ActionMapping` object, created when the application starts up, based on information in an XML file called `struts-config.xml`. From this `ActionMapping` object, the Struts `ActionServlet` determines the action-derived class that is expected to handle the request.

The `Action` object is then invoked to perform the required processing. This `Action` object is provided by the developer using Struts to create a Web application and can use any convenient technology for processing the request. The `Action` object is the route into the model for the application. Once processing has been completed, the `Action` object can indicate what should happen next—the `ActionServlet` will use this information to select the appropriate response agent (normally a JSP) to generate the dynamic content to be sent back to the user. The JSP represents the view for the application.

Struts provides other features, such as form beans, to represent data entered into HTML forms and JSP tag extensions to facilitate Struts JSP development.

Requirements for the development environment

Because Struts applications are also Web applications, all the functionality described in “Simple Web applications” on page 40, is relevant in this context as well. In addition, the development environment should provide:

- Wizards to create:
  - A Struts and Tiles enabled dynamic Web application
  - A new Struts `Action` class and corresponding `ActionMapping`
  - A new `ActionForm` bean
  - A new Struts exception type
  - A new Struts module
- An editor to modify the `struts-config.xml` file
- A graphical editor to display and modify the relationship between Struts elements, such as actions, action forms, and view components

In addition, the basic Web application tools should be Struts-aware. The wizard for creating Web applications should include a simple mechanism for adding Struts support, and the wizard for creating JSPs should offer to add the necessary Struts tag libraries.

IBM Rational Application Developer V7.0 provides all of these features.

Figure 2-1 on page 47 still represents the structure of a Web application using Struts. Although Struts provides us with a framework on which we can build our own applications, the technology is still the same as for basic Web applications.

JavaServer Faces (JSF) and Service Data Objects (SDO)

When we build a GUI using stand-alone Java applications, we can include event-handling code, so that when UI events take place they can be used immediately to perform business logic processing or update the UI. Users are familiar with this type of behavior in desktop applications, but the nature of Web applications has made this difficult to achieve using a browser-based interface; the user interface provided through HTML is limited, and the request-response style of HTTP does not naturally lead to flexible, event-driven user interfaces.

Many applications require access to data, and there is often a requirement to be able to represent this data in an object-oriented way within applications. Many tools and frameworks exist for mapping between data and objects (we will see J2EE's standard system, CMP entity beans, later in this chapter), but often these are proprietary or excessively heavy weight systems.

In the RedBank Web application we want to make the user interface richer, while still allowing us to use the MVC architecture described in “Struts” on page 47. In addition, our developers want a simple, lightweight, object-oriented database access system, which will remove the need for direct JDBC coding.

JavaServer Faces (JSF)

JSF is a framework for developing Java Web applications. The JSF framework aims to unify techniques for solving a number of common problems in Web application design and development, such as:

- **User interface development**: JSF allows direct binding of user interface (UI) components to model data. It abstracts request processing into an event-driven model. Developers can use extensive libraries of prebuilt UI components that provide both basic and advanced Web functionality.
- **Navigation**: JSF introduces a layer of separation between business logic and the resulting UI pages; stand-alone flexible rules drive the flow of pages.

- **Session and object management**: JSF manages designated model data objects by handling their initialization, persistence over the request cycle, and cleanup.
- **Validation and error feedback**: JSF allows direct binding of reusable validators to UI components. The framework also provides a queue mechanism to simplify error and message feedback to the application user. These messages can be associated with specific UI components.
- **Internationalization**: JSF provides tools for internationalizing Web applications, supporting number, currency, time, and date formatting, and externalizing of UI strings. JSF is easily extended in a variety of ways to suit the requirements of your particular application. You can develop custom components, renderers, validators, and other JSF objects and register them with the JSF runtime.

The JSF specification can be found at:

`http://www.jcp.org/en/jsr/detail?id=127`

Service Data Objects (SDO)

SDO is a data programming architecture and API for the Java platform that unifies data programming across data source types; provides robust support for common application patterns; and enables applications, tools, and frameworks to more easily query, view, bind, update, and introspect data.

SDO was originally developed by IBM and BEA Systems and is now the subject of a Java specification request (JSR 235), but has not yet been standardized under this process.

SDOs are designed to simplify and unify the way in which applications handle data. Using SDO, application programmers can uniformly access and manipulate data from heterogeneous data sources, including relational databases, XML data sources, Web services, and enterprise information systems.

The SDO architecture consists of three major components:

- **Data object**: The data object is designed to be an easy way for a Java programmer to access, traverse, and update structured data. Data objects have a rich variety of strongly and loosely typed interfaces for querying and updating properties. This enables a simple programming model without sacrificing the dynamic model required by tools and frameworks. A data object can also be a composite of other data objects.
- **Data graph**: SDO is based on the concept of disconnected data graphs. A data graph is a collection of tree-structured or graph-structured data objects. Under the disconnected data graphs architecture, a client retrieves a data graph from a data source, mutates the data graph, and can then apply the

data graph changes to the data source. The data graph also contains some meta data about the data object, including change summary and meta data information. The meta data API allows applications, tools, and frameworks to introspect the data model for a data graph, enabling applications to handle data from heterogeneous data sources in a uniform way.

- **Data mediator**: The task of connecting applications to data sources is performed by a data mediator. Client applications query a data mediator and get a data graph in response. Client applications send an updated data graph to a data mediator to have the updates applied to the original data source. This architecture allows applications to deal principally with data graphs and data objects, providing a layer of abstraction between the business data and the data source.

More information on JSF and SDO can be found in the IBM Redbooks publication, *WebSphere Studio V5.1.2 JavaServer Faces and Service Data Objects*, SG24-6361. This book covers the use of these technologies in WebSphere Studio rather than the Rational Software Development Platform, but the coverage of the technologies is still very useful.

Requirements for the development environment

The development environment should provide tooling to create and edit pages based on JSF, to modify the configuration files for JSF applications, and to test them. For SDO, the development environment should provide wizards to create SDOs from an existing database (bottom-up mapping) and should make it easy to use the resulting objects in JSF and other applications.

IBM Rational Application Developer V7.0 includes these features.

Figure 2-2 shows how JSF and SDO can be used to create a flexible, powerful MVC-based Web application with simple database access.

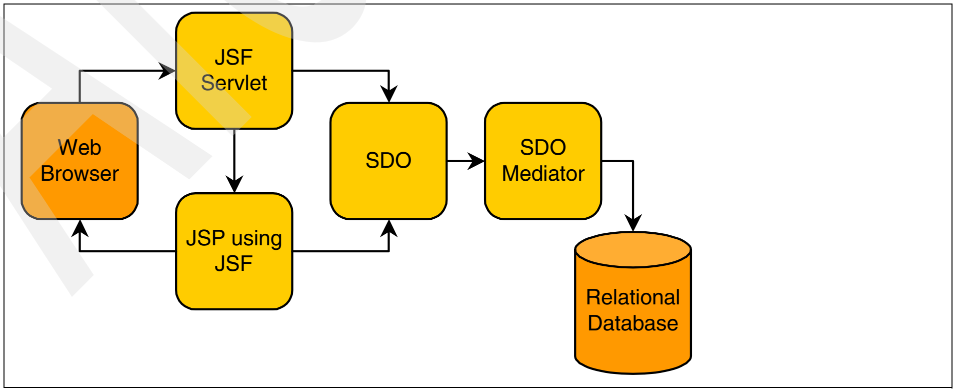


Figure 2-2 JSF and SDO

AJAX

AJAX is an acronym that stands for Asynchronous JavaScript™ and XML. AJAX is a Web 2.0 development technique used for creating interactive Web applications. The intent is to make Web pages feel more responsive by exchanging small amounts of data with the server behind the scenes. Technically, in order to send and process a request to the serverside, the Web page will not require a reload. Javascript is utilized to gather DOM values on the page and data is transmitted from the browser to the server through the `XMLHTTPRequest` object.

Figure 2-3 illustrates the overall AJAX interaction between the client browser and the server-side application. Refer to the list below the figure for the generic sequence of steps that get executed on a typically AJAX submission.

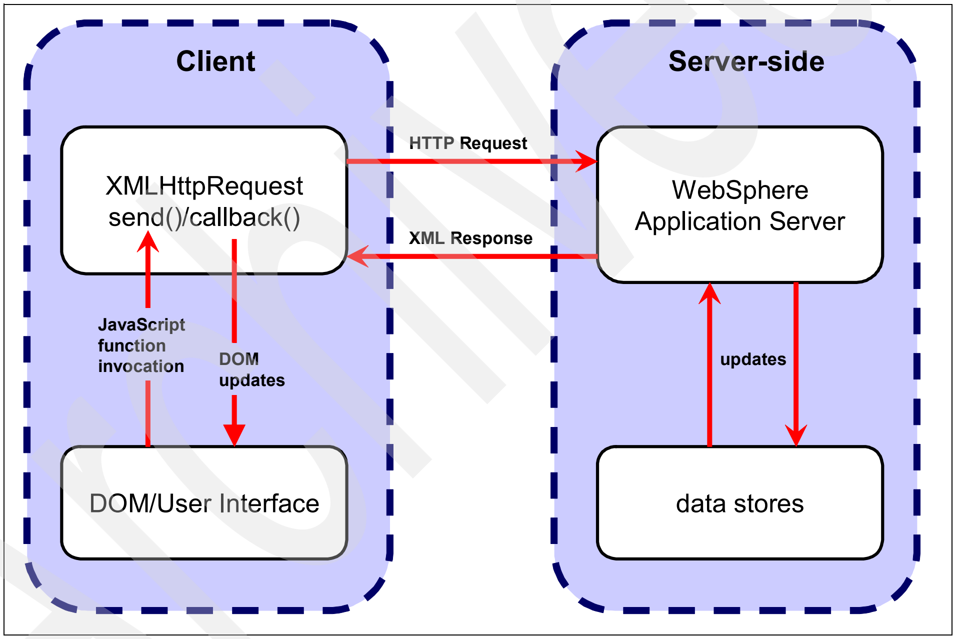


Figure 2-3 AJAX overview

- **JavaScript functional invocation**—Typically, when a AJAX call takes place, the first step is to emulate what typically happens in a synchronous GET or POST. That is, collect the information from the page that is necessary for processing the server-side request. In the case of AJAX, the collection of these values occurs through a JavaScript function (and retrieves DOM values).

- **XMLHttpRequest.send**—Once we have collected all the necessary information from the DOM, a JavaScript object is created and eventually stores all the collected DOM values. In Mozilla-based browsers, this JavaScript object is a `XMLHttpRequest` object. Once fully configured, the `XMLHttpRequest` object is sent to the server-side via HTTP Request.
- **Server-side processing**—Assuming that the application is based on Java EE, from a Web tier perspective, the AJAX request can either be handled by one of the following Web technologies: Servlet, JSF or Portlet. Once received, the AJAX request from the server-side will continue to execute as usual (accessing a data store, executing business logic, and so forth).
- **XML response**—Once the server completes processing, the resulting data is returned in a response, typically in XML format.
- **XMLHttpRequest.callback**—The callback function processes the response. As part of the response, JavaScript function would have to take the response values and update the page (by changing/updating DOM values).

Portal applications

Portal applications run on a Portal Server and consist of portal pages that are composed of portlets. Portlets can share and exchange resources and information to provide a seamless Web interface.

Portal applications have several important features:

- They can collect content from a variety of sources and present them to the user in a single unified format.
- The presentation can be personalized so that each user sees a view based on their own characteristics or role.
- The presentation can be customized by the user to fulfill their specific needs.
- They can provide collaboration tools, which allow teams to work in a virtual office.
- They can provide content to a range of devices, formatting and selecting the content appropriately according to the capabilities of the device.

In the context of our sample scenario, we can use a portal application to enhance the user experience. The RedBank Web application can be integrated with the static Web content providing information about branches and bank services. If the customer has credit cards, mortgages, personal loans, savings accounts, shares, insurance, or other products provided by the bank or business partners, these could also be seamlessly integrated into the same user interface, providing the customer with a convenient single point of entry to all these services.

The content or these applications can be provided from a variety of sources, with the portal server application collecting the content and presenting it to the user. The user can customize the interface to display only the required components, and the content can be varied to allow the customer to connect using a Web browser, a personal digital assistant (PDA), or mobile phone.

Within the bank, the portal can also be used to provide convenient intranet facilities for employees. Sales staff can use a portal to receive information on the latest products and special offers, information from human resources, leads from colleagues, and so on.

IBM WebSphere Portal

WebSphere Portal runs on top of WebSphere Application Server, using the J2EE standard services and management capabilities of the server as the basis for portal services. WebSphere Portal provides its own deployment, configuration, administration, and communication features.

The WebSphere Portal Toolkit is provided as part of the Rational Application Developer as a complete development environment for developing portal applications. A wizard allows a developer to begin development of a new portlet application, generating a skeleton portlet application as a project in Rational Application Developer and required deployment descriptors. The Portlet Toolkit also provides debugging support for portal developers.

Java Portlet specification

The Java Portlet V1.0 specification (`http://jcp.org/en/jsr/detail?id=168`) has been developed to provide a standard for the development of Java portlets for portal applications. WebSphere Portal V5.1 supports the Java Portlet standard.

Requirements for the development environment

The development environment should provide wizards for creating portal applications and the associated components and configuration files, as well as editors for all these files. A test environment should be provided to allow portal applications to be executed and debugged.

Application Developer V7.0 includes the WebSphere Portal Toolkit and the WebSphere Portal V5.1/V6.0 Integrated Test Environment.

Figure 2-4 shows how portal applications fit in with other technologies mentioned in this chapter.

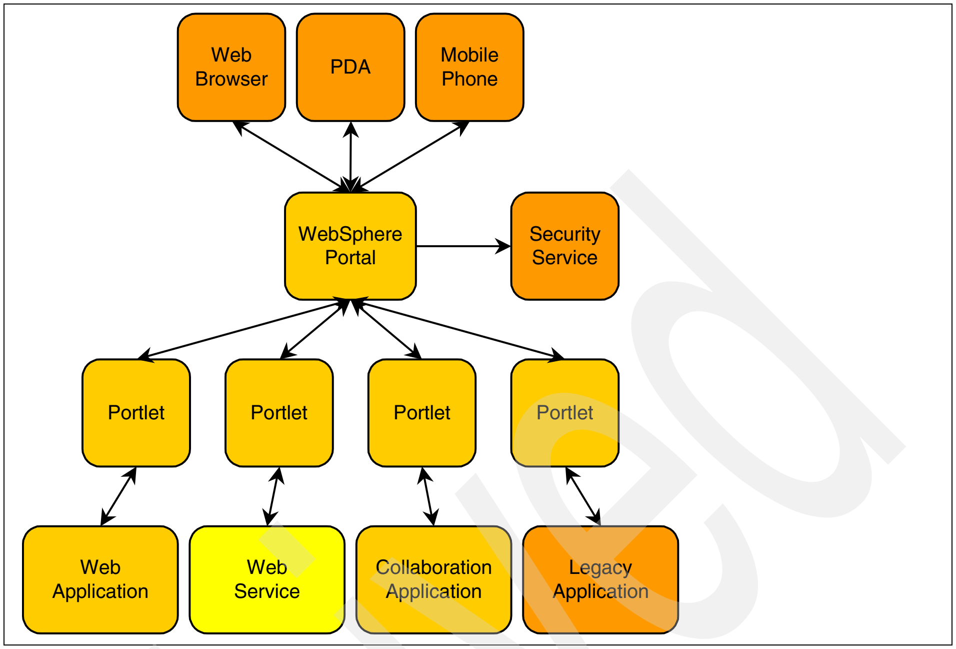


Figure 2-4 Portal applications

Enterprise JavaBeans

Now that the RedBank Web application is up and running, more issues arise. Some of these relate to the services provided to customers and bank workers and some relate to the design, configuration, and functionality of the systems that perform the back-end processing for the application.

First, we want to provide the same business logic in a new application that will be used by administration staff working in the bank's offices. We would like to be able to reuse the code that has already been generated for the RedBank Web application without introducing the overhead of having to maintain several copies of the same code. Integration of these business objects into a new application should be made as simple as possible.

Next, we want to reduce development time by using an object-relational mapping system that will keep an in-memory, object-oriented view of data with the relational database view automatically, and provide convenient mapping tools to set up the relationships between objects and data. This system should be capable of dealing with distributed transactions, since the data might be located on several different databases around the bank's network.

Since we are planning to make business logic available to multiple applications simultaneously, we want a system that will manage such issues as multithreading, resource allocation, and security so that developers can focus on writing business logic code without having to worry about infrastructure matters such as these.

Finally, the bank has legacy systems, not written in Java, that we would like to be able to update to use the new functionality provided by these business objects. We would like to use a technology that will allow this type of interoperability between different platforms and languages.

We can get all this functionality by using Enterprise JavaBeans (EJBs) to provide our back-end business logic and access to data. Later, we will see how EJBs can also allow us to integrate messaging systems and Web services clients with our application logic.

Different types of EJBs

This section describes several types of EJBs, including session, entity, and message driven beans.

Session EJBs

Session EJBs are task-oriented objects, which are invoked by an EJB client. They are non-persistent and will not survive an EJB container shutdown or crash. There are two types of session EJB: *Stateless* and *stateful*.

Session beans often act as the external face of the business logic provided by EJBs. The session facade pattern, described in many pattern catalogs including *Core J2EE Patterns: Best Practices and Design Strategies* by Crupi, et al., describes this idea. The client application that needs to access the business logic provided by some EJBs sees only the session beans. The low-level details of the persistence mechanism are hidden behind these session beans (the session bean layer is known as the session facade). As a result of this, the session beans that make up this layer are often closely associated with a particular application and might not be reusable between applications.

It is also possible to design reusable session beans, which might represent a common service that can be used by many applications.

Stateless session EJBs

Stateless session EJBs are the preferred type of session EJB, since they generally scale better than stateful session EJBs. Stateless beans are pooled by the EJB container to handle multiple requests from multiple clients. In order to permit this pooling, stateless beans cannot contain any state information that is specific to a particular client. Because of this restriction, all instances of a stateless bean are equivalent, allowing the EJB container to assign an instance to any client.

Stateful session EJBs

Stateful session EJBs are useful when an EJB client needs to call several methods and store state information in the session bean between calls. Each stateful bean instance must be associated with exactly one client, so the container is unable to pool stateful bean instances.

Entity EJBs

Entity EJBs are designed to provide an object-oriented view of data. The data itself is stored by an external persistence mechanism, such as a relational database or other enterprise information system. Entity beans should normally be designed to be general purpose, not designed to work with one particular application—the application-specific logic should normally be placed in a layer of session EJBs that use entity beans to access data when required, as described above.

Once an entity bean has been created, it can be found using a key or using some other search criteria. It persists until it is explicitly removed. Two main types of entity bean exist: Container-managed or bean-managed persistence.

Container-managed persistence (CMP)

The EJB container handles all database access required by the entity bean. The bean's code contains no database access (SQL) calls. As a result, the bean's code is not tied to a specific database. Because of this flexibility, even if you redeploy the same entity bean on different J2EE servers that use different databases, you do not have to modify or recompile the bean's code. The container must provide an object-relational mapping tool to allow a developer or deployer to describe how the attributes of an entity bean map onto columns in tables of a database.

Bean-managed persistence (BMP)

The developer handles all storage-specific access required by the entity bean. This allows the developer to use non-relational storage options and features of relational databases that are not supported by CMP entity beans, such as complex SQL and stored procedures.

Message-driven EJBs (MDBs)

MDBs are designed to receive and process messages.They can be accessed only by sending a message to the messaging server that the bean is configured to listen to. MDBs are stateless and can be used to allow asynchronous communication between a client EJB logic via some type of messaging system.

MDBs are normally configured to listen to Java Message Service (JMS) resources, although from EJB V2.1, other messaging systems can also be supported. MDBs are normally used as adapters to allow logic provided by session beans to be invoked via a messaging system; as such, they can be thought of as an asynchronous extension of the session facade concept described above, known as the message facade pattern. Message-driven beans can only be invoked in this way and therefore have no specific client interface.

Other EJB features

This section describes other EJB features not discussed previously.

Container-managed relationships (CMR)

From EJB V2.0, the EJB container is able to manage the relationships between entity beans. All the code required to manage the relationships is generated automatically as part of the deployment process:

- One-to-one relationships: A CMP entity bean is associated with a single instance of another CMP entity bean (for example, a customer has one address).
- One-to-many relationships: A CMP entity bean is associated with multiple instances of another CMP entity bean (for example, an account has many transactions).
- Many-to-many relationship: Multiple instances of a CMP entity bean are associated with multiple instances of another CMP entity bean (for example, a customer has many accounts, an account belongs to many customers).

The methods required to traverse the relationships are generated as part of the development and code generation mechanisms.

EJB query language (EJB QL)

EJB QL is used to specify queries for CMP entity beans. It is based on SQL and allows searches on the persistent attributes of an enterprise bean and allows container-managed relationships to be traversed as part of the query. EJB QL defines queries in terms of the declared structure of the EJBs themselves, not the underlying data store; as a result, the queries are independent of the bean's mapping to a persistent store.

An EJB query can be used to define a finder method or a select method of a CMP entity bean. Finder and select methods are specified in the bean's deployment descriptor using the `<ejb-ql>` element. Queries specified in the deployment descriptor are compiled into SQL during deployment.

An EJB QL query is a string that contains the following elements:

- A SELECT clause that specifies the enterprise beans or values to return
- A FROM clause that names the bean collections
- A WHERE clause that contains search predicates over the collections

Only SELECT statements can be created using EJB QL. EJB QL is like a cut-down version of SQL, using the abstract persistence schema defined by the EJBs rather than tables and columns. The language has been extended in EJB V2.1 compared with V2.0. A query can contain input parameters that correspond to the arguments of the finder or select method.

Local and remote interfaces

EJBs were originally designed around remote invocation using the Java Remote Method Invocation (RMI) mechanism, and later extended to support standard Common Object Request Broker Architecture (CORBA) transport for these calls using RMI over the Internet Inter-ORB Protocol (RMI-IIOP). Since many EJB clients make calls to EJBs that are in the same container, from EJB V2.0 onwards, local interfaces can be used instead for calls when the client is in the same JVM as the target EJB. For EJB-to-EJB and servlet-to-EJB calls, this avoids the expensive network call. A session or entity EJB can be defined as having a local and/or remote interface. MDBs do not have any interfaces, so this issue does not arise.

EJB timer service

The EJB timer service was introduced with EJB V2.1. A bean provider can choose to implement the `javax.ejb.TimedObject` interface, which requires the implementation of a single method, `ejbTimeout`. The bean creates a `Timer` object by using the `TimerService` object obtained from the bean's `EJBContext`. Once the `Timer` object has been created and configured, the bean will receive messages from the container according to the specified schedule; the container calls the `ejbTimeout` method at the appropriate interval.

Requirements for the development environment

The development environment should provide wizards for creating the various types of EJB, tools for mapping CMP entity beans to relational database systems and test facilities.

IBM Rational Application Developer V7.0 provides all these features.

Figure 2-5 shows how EJBs work with other technologies already discussed. The provision of remote interfaces to EJBs means that the method calls made by the JavaBean to the session EJB could take place across a network connection, allowing the application to be physically distributed across several machines. Performance considerations might make this option less attractive, but it is a useful technology choice in some situations. As we will see later, using remote interfaces is sometimes the only available choice.

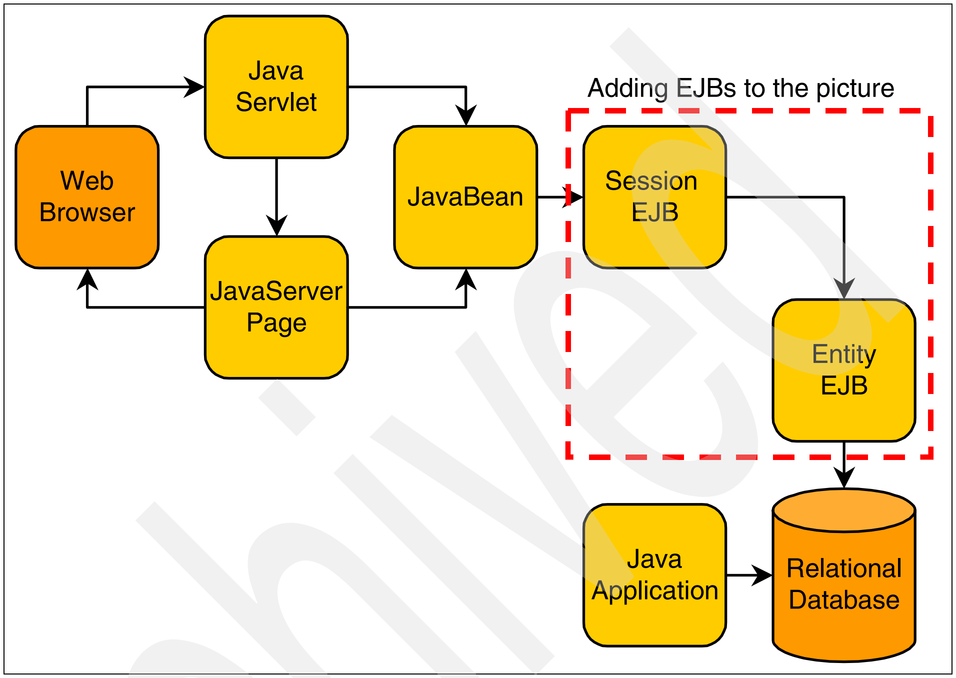


Figure 2-5 EJBs as part of an enterprise application

J2EE Application Clients

Java EE Application Clients are one of the four types of components defined in the Java EE specification—the others being EJBs, Web components (servlets and JSPs), and Java applets. They are stand-alone Java applications that use resources provided by a Java EE application server, such as EJBs, data sources and JMS resources.

In the context of our banking sample application, we want to provide an application for bank workers who are responsible for creating accounts and reporting on the accounts held at the bank. Since a lot of the business logic for accessing the bank's database has now been developed using EJBs, we want to avoid duplicating this logic in our new application. Using a J2EE Application Client for this purpose will allow us to develop a convenient interface, possibly a GUI, while still allowing access to this EJB-based business logic. Even if we do not want to use EJBs for business logic, a Java EE Application Client will allow us to access data sources or JMS resources provided by the application server and will allow us to integrate with the security architecture of the server.

Application Programming Interfaces (APIs)

The Java EE specification (available from http://java.sun.com/j2ee/) requires the following APIs to be provided to Java EE Application Clients, in addition to those provided by a standard Java SE JVM:

- EJB V2.1 client-side APIs
- JMS V1.1
- JavaMail V1.3
- JavaBeans Activation Framework (JAF) V1.0
- Java API for XML Processing (JAXP) V1.2
- Web Services for J2EE V1.1
- Java API for XML-based Remote Procedure Call (JAX-RPC) V1.1
- SOAP with Attachments API for Java (SAAJ) V1.2
- Java API for XML Registries (JAXR) V1.0
- Java EE Management 1.0
- Java Management Extensions (JMX) V1.2

Security

The Java EE specification requires that the same authentication mechanisms should be made available for Java EE Application Clients as for other types of Java EE components. The authentication features are provided by the Java EE Application Client container, as they are in other containers within Java EE. A Java EE platform can allow the Java EE Application Client container to communicate with an application server to use its authentication services; WebSphere Application Server allows this.

Naming

The Java EE specification requires that Java EE Application Clients should have exactly the same naming features available as are provided for Web components and EJBs. Java EE Application Clients should be able to use the Java Naming and Directory Interface (JNDI) to look up objects using object references as well as real JNDI names. The reference concept allows a deployer to configure references that can be used as JNDI names in lookup code. The references are bound to real JNDI names at deployment time, so that if the real JNDI name is subsequently changed, the code does not have to be modified or recompiled—only the binding needs to be updated.

References can be defined for:

- EJBs (for J2EE Application Clients, only remote references, because the client cannot use local interfaces)
- Resource manager connection factories
- Resource environment values
- Message destinations
- User transactions
- ORBs

Code to look up an EJB might look like this (this is somewhat simplified):

```
accountHome = (AccountHome)initialContext
                            .lookup("java:comp/env/ejb/account");
```

`java:comp/env/` is a standard prefix used to identify references, and ejb/account would be bound at deployment time to the real JNDI name used for the Account bean.

Deployment

The Java EE specification only specified the packaging format for Java EE Application Clients, not how these should be deployed—this is left to the Platform provider. The packaging format is specified, based on the standard Java JAR format, and it allows the developer to specify which class contains the *main* method to be executed at run time.

Java EE application clients for the WebSphere Application Server platform run inside the *Application Client for WebSphere Application Server*. This is a product that is available for download from developerWorks, as well the WebSphere Application Server installation CD.

Refer to the WebSphere Application Server Information Center for more information about installing and using the Application Client for WebSphere Application Server.

The Application Client for WebSphere Application Server provides a **`launchClient`** command, which sets up the correct environment for Java EE Application Clients and runs the main class.

Requirements for the development environment

In addition to the standard Java tooling, the development environment should provide a wizard for creating Java EE Application Clients, editors for the deployment descriptor for a Java EE Application Client module, and a mechanism for testing the Java EE Application Client.

Application Developer V7.0 provides these features.

Figure 2-6 shows how Java EE Application Clients fit into the picture; since these applications can access other Java EE resources, we can now use the business logic contained in our session EJBs from a stand-alone client application. Java EE Application Clients run in their own JVM, normally on a different machine from the EJBs, so they can only communicate using remote interfaces.

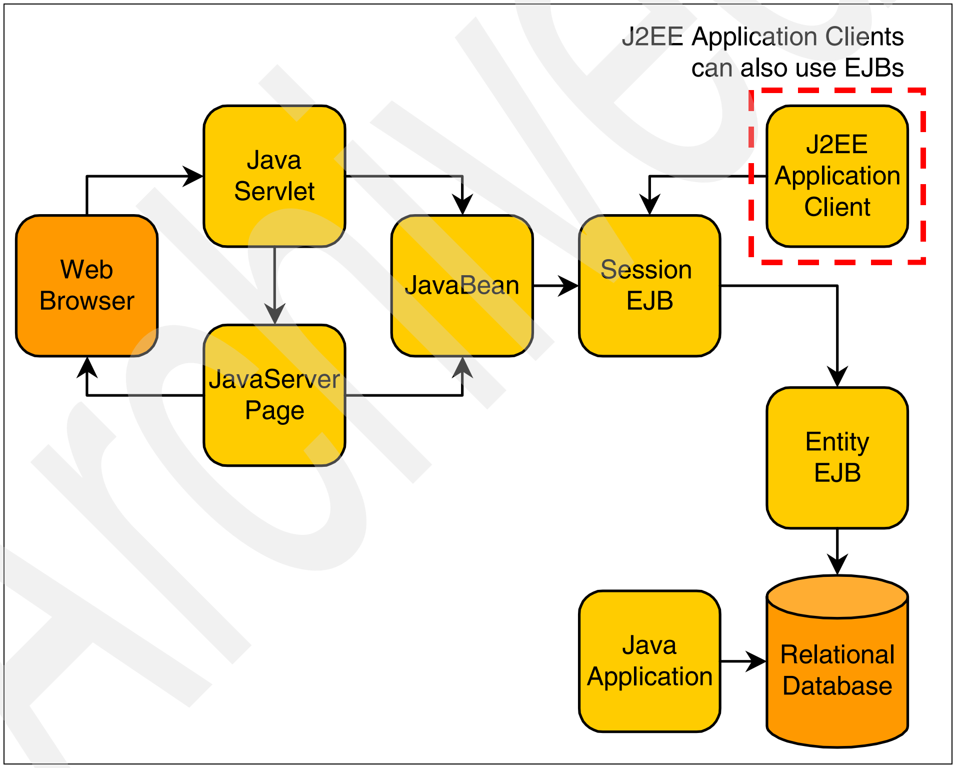


Figure 2-6 J2EE Application Clients

Web services

The bank’s computer system is now quite sophisticated, comprising:

- A database for storing the bank’s data
- A Java application allowing bank employees to access the database
- A static Web site, providing information on the bank’s branches, products, and services
- A Web application, providing Internet banking facilities for customers, with various technology options available
- An EJB back-end, providing:
  - Centralized access to the bank’s business logic through session beans
  - Transactional, object-oriented access to data in the bank’s database through entity beans
- A Java EE Application Client that can use the business logic in session beans

So far, everything is quite self-contained. Although clients can connect from the Web to use the Internet banking facilities, the business logic is all contained within the bank’s systems, and even the Java application and Java EE Application Client are expected to be within the bank’s private network.

The next step in developing our service is to enable mortgage agents, who search many mortgage providers to find the best deal for their customers, to access business logic provided by the bank to get the latest mortgage rates and repayment information. While we want to enable this, we do not want to compromise security, and we need to take into account that fact that the mortgage brokers might not be using systems based on Java at all.

The League of Agents for Mortgage Enquiries has published a description of services that its members might use to get this type of information. We want to conform to this description in order to allow the maximum number of agents to use our bank’s systems.

We might also want to be able to share information with other banks; for example, we might want to exchange information on funds transfers between banks. Standard mechanisms to perform these tasks have been provided by the relevant government body.

These issues are all related to interoperability, which is the domain addressed by Web services. Web services will allow us to enable all these different types of communication between systems. We will be able to use our existing business logic where applicable and develop new Web services easily where necessary.

Web services in Java EE V1.4

Web services provide a standard means of communication among different software applications. Because of the simple foundation technologies used in enabling Web services, it is very simple to a Web service regardless of the Platform, operating system, language, or technology used to implement it.

A *service provider* creates a Web service and publishes its interface and access information to a *service registry* (or *service broker*). A *service requestor* locates entries in the *service registry*, then binds to the *service provider* in order to invoke its Web service.

Web services use the following standards:

- **SOAP**: A XML-based protocol that defines the messaging between objects
- **Web Services Description Language** (WSDL): Describes Web services interfaces and access information
- **Universal Description, Discovery, and Integration** (UDDI): A standard interface for service registries, which allows an application to find organizations and services

The specifications for these technologies are available at:

```
http://www.w3.org/TR/soap/
http://www.w3.org/TR/wsdl/
http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.html
```

Figure 2-7 shows how these technologies fit together.

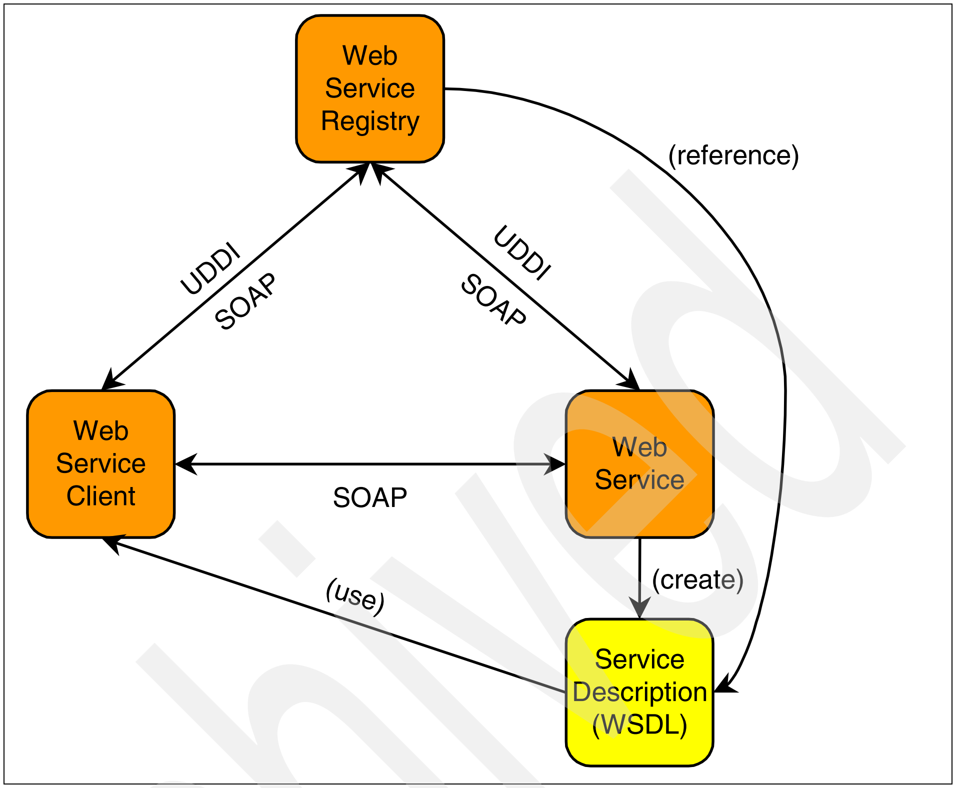


Figure 2-7 Web services foundation technologies

Web services are now included in the Java EE specification (from V1.4), so all Java EE application servers that support Java EE V1.4 have exactly the same basic level of support for Web services; some will also provide enhancements as well.

The key Web services-related Java technologies that are specified in the Java EE V1.4 specification are:

- Web Services for J2EE V1.1:

  `http://jcp.org/en/jsr/detail?id=921`

> **Note:** The URL quoted above for the Web Services for Java EE V1.1 specification leads to a page entitled *JSR 921: Implementing Enterprise Web Services 1.1*. This is the V1.1 maintenance release for Web Services for Java EE, which was originally developed under JSR 109.

- Java API for XML-based Remote Procedure Call (JAX-RPC) V1.1:

  `http://java.sun.com/xml/jaxrpc/`

The Web services support in Java EE also relies on the underlying XML support in the Platform, provided by JAXP and associated specifications.

Web Services for Java EE defines the programming and deployment model for Web services in Java EE. It includes details of the client and server programming models, handlers (a similar concept to servlets filters), deployment descriptors, container requirements, and security.

JAX-RPC defines the various APIs required to enable XML-based remote procedure calls in Java.

Since interoperability is a key goal in Web services, an open, industry organization known as the Web Services Interoperability Organization (WS-I, `http://ws-i.org/`) has been created to allow interested parties to work together to maximize the interoperability between Web services implementations. WS-I has produced the following set of interoperability profiles:

- WS-I Basic Profile

  `http://ws-i.org/Profiles/BasicProfile-1.0.html`
- WS-I Simple SOAP Binding Profile

  `http://ws-i.org/Profiles/SimpleSoapBindingProfile-1.0.html`
- WS-I Attachments Profile

  `http://ws-i.org/Profiles/AttachmentsProfile-1.0.html`

The Java EE V1.4 specification allows for ordinary Java resources and stateless session EJBs to be exposed as Web services. The former, known as *JAX-RPC service endpoint implementations*, can be hosted within the Java EE application server's Web container, and the latter can be hosted within the EJB container.

Requirements for the development environment

The development environment should provide facilities for creating Web services from existing Java resources—both JAX-RPC service endpoint implementations and stateless session EJBs. As part of the creation process, the tools should also produce the required deployment descriptors and WSDL files. Editors should be provided for WSDL files and deployment descriptors.

The tooling should also allow skeleton Web services to be created from WSDL files and should provide assistance in developing Web services clients, based on information obtained from WSDL files.

A range of test facilities should be provided, allowing a developer to test Web services and clients as well as UDDI integration.

Application DeveloperV7.0 provides all this functionality.

Figure 2-8 shows how the Web services technologies fit into the overall programming model we have been discussing.

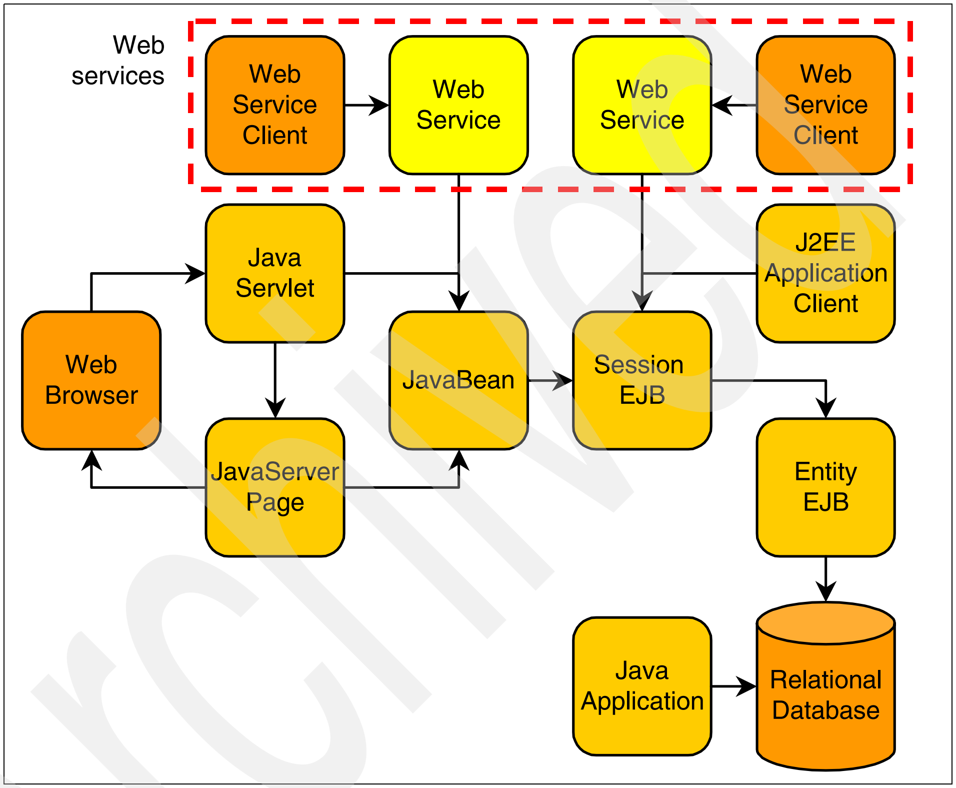


Figure 2-8 Web services

Messaging systems

Although Web services offer excellent opportunities for integrating disparate systems, they currently have some drawbacks:

- Security specifications are available for Web services, but they have not been fully implemented on all platforms at this stage.
- Web services do not provide guaranteed delivery, and there is no widely implemented, standard way of reporting the reliability of a Web service.

- There is no standard way of scaling Web services to the degree required by modern information technology systems, no way of distributing Web services transparently across servers, and no way of recovering from system failures.

The bank has numerous automatic teller machines (ATMs), each of which has a built-in computer providing user interface and communication support. The ATMs are designed to communicate with the bank's central computer systems using a secure, reliable, highly scalable messaging system. We would like to integrate the ATMs with our system so that transactions carried out at an ATM can be processed using the business logic we have already implemented. Ideally, we would also like to have the option of using EJBs to handle the messaging for us.

Many messaging systems exist that provide features like these. IBM's solution in this area is IBM WebSphere MQ, which is available on many platforms and provides application programming interfaces in several languages. From the point of view of our sample scenario, WebSphere MQ provides Java interfaces that we can use in our applications—in particular, we will consider the interface that conforms to the Java Message Service (JMS) specification. The idea of JMS is similar to that of JDBC—a standard interface providing a layer of abstraction for developers wishing to use messaging systems without being tied to a specific implementation.

Java Message Service (JMS)

JMS defines (among other things):

- A messaging model: The structure of a JMS message and an API for accessing the information contained within a message. The JMS interface is, `javax.jms.Message`, implemented by several concrete classes, such as `javax.jms.TextMessage`.
- Point-to-point (PTP) messaging: A queue-based messaging architecture, similar to a mailbox system. The JMS interface is `javax.jms.Queue`.
- Publish/subscribe (Pub/Sub) messaging: A topic-based messaging architecture, similar to a mailing list. Clients subscribe to a topic and then receive any messages that are sent to the topic. The JMS interface is `javax.jms.Topic`.

More information on JMS can be found at:

`http://java.sun.com/products/jms/`

Message-driven EJBs (MDBs)

MDBs were introduced into the EJB architecture at V2.0 and have been extended in EJB V2.1. MDBs are designed to consume incoming messages sent from a destination or endpoint system the MDB is configured to listen to. From the point of view of the message-producing client, it is impossible to tell how the message is being processed—whether by a stand-alone Java application, a MDB, or a message-consuming application implemented in some other language. This is one of the advantages of using messaging systems; the message-producing client is very well decoupled from the message consumer (similar to Web services in this respect).

From a development point of view, MDBs are the simplest type of EJB, since they do not have clients in the same sense as session and entity beans. The only way of invoking an MDB is to send a message to the endpoint or destination that the MDB is listening to. In EJB V2.0, MDBs only dealt with JMS messages, but in EJB V2.1 this is extended to other messaging systems. The development of an MDB is different depending on the messaging system being targeted, but most MDBs are still designed to consume messages via JMS, which requires the bean class to implement the javax.jms.MessageListener interface, as well as javax.ejb.MessageDrivenBean.

A common pattern in this area is the message facade pattern, as described in *EJB Design Patterns: Advanced Patterns, Processes and Idioms* by Marinescu. This book is available for download from:

`http://theserverside.com/articles/`

According to this pattern, the MDB simply acts as an adapter, receiving and parsing the message, then invoking the business logic to process the message using the session bean layer.

Requirements for the development environment

The development environment should provide a wizard to create MDBs and facilities for configuring the MDBs in a suitable test environment. The test environment should also include a JMS-compliant server (this is a Java EE V1.4 requirement anyway).

Testing MDBs is challenging, since they can only be invoked by sending a message to the messaging resource that the bean is configured to listen to. However, WebSphere Application Server V6.1, which is provided as a test environment within Rational Application Developer, includes an embedded JMS messaging system that can be used for testing purposes. A JMS client must be developed to create the test messages.

Figure 2-9 shows how messaging systems and MDBs fit into the application architecture.

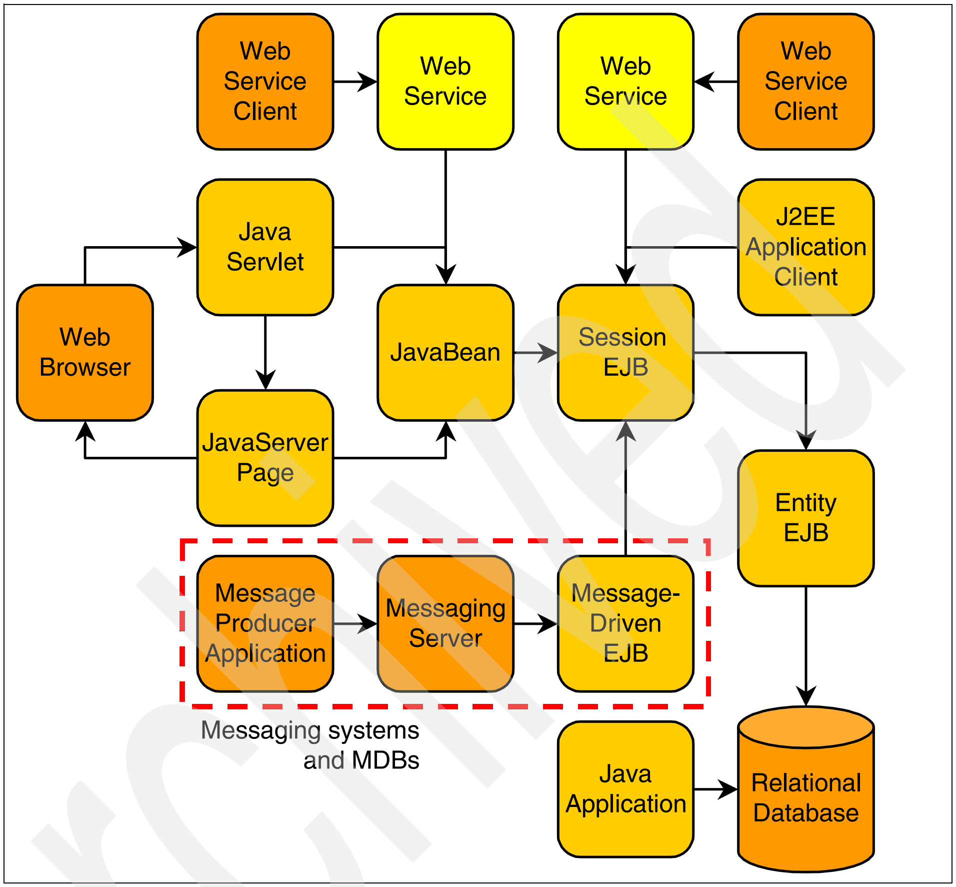


Figure 2-9   Messaging systems

Summary

In this chapter we reviewed the basic technologies supported by Application Developer: Desktop applications, static Web sites, dynamic Web applications, Enterprise JavaBeans, Web services, and messaging.

3

Workbench setup and preferences

After installing IBM Rational Application Developer V7.0, the Workbench is configured with a default configuration to make it easier to navigate for new users. Developers are made aware of enabling the capabilities of Application Developer, if needed. Alternatively, developers can configure the Workbench preferences for their needs manually at any time. This chapter describes the most commonly used Application Developer preferences.

The chapter is organized into the following sections:

- Workbench basics
- Preferences
- Java development preferences

Workbench basics

After starting Application Developer, after the installation, you see a single window with the Welcome page (Figure 3-1). The Welcome page can be accessed subsequently by selecting **Help** → **Welcome** from the Workbench menu bar. The Welcome page is provided to guide a new user of Application Developer to the various aspects of the tool.

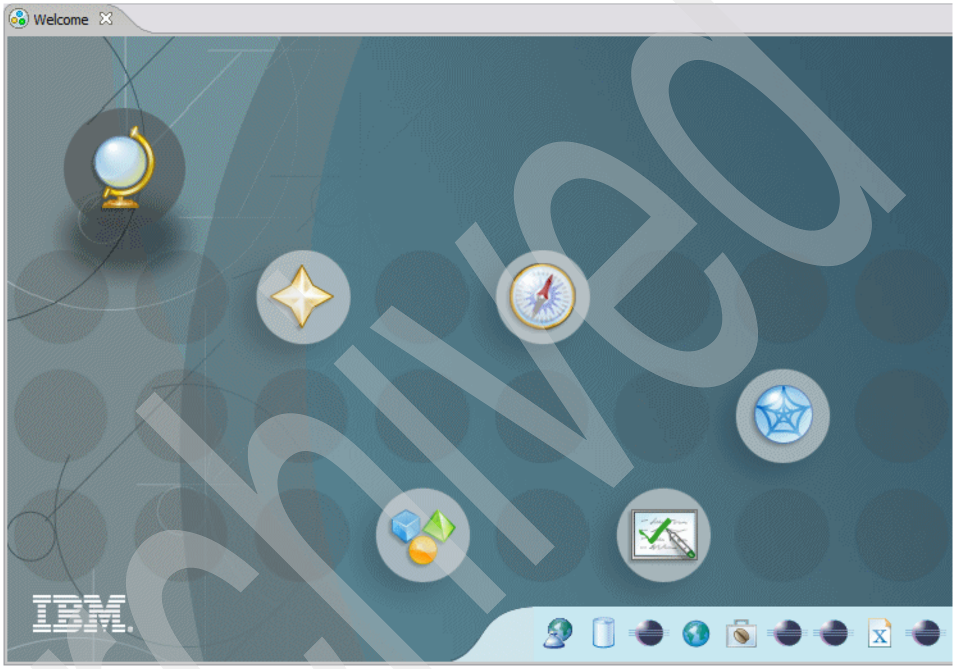


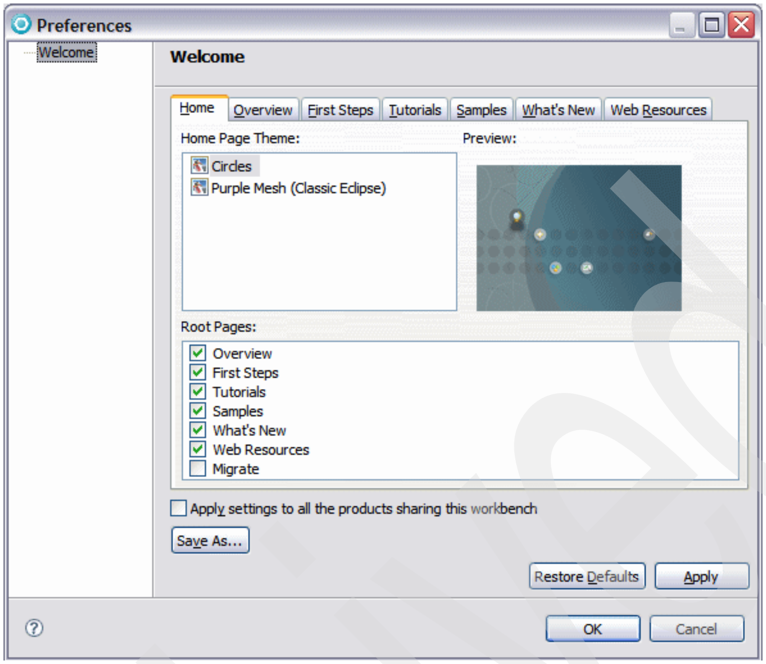
Figure 3-1 Application Developer Workbench Welcome page

The Welcome page presents six icons, each including a description that is visible through hover help (moving the mouse over an icon to display a description). Table 3-1 provides a summary of each icon.

Table 3-1 Welcome page assistance capabilities

Icon Image	Name	Description
	Overview	Provides an overview of the key functions in Application Developer.
	What's New	A description of the major new features and highlights of the product.
	Tutorials	Tutorial screens to learn how to use key features Application Developer. Provides a link to Tutorials Gallery.
	Samples	Sample code for the user to begin working with "live" examples with minimal assistance. Provides a link to Samples Gallery
	First Steps	Step-by-step guidance to help first-time users to perform some key tasks.
	Web Resources	URL links to Web pages where you can find relevant and timely tips, articles, updates, and references to industry standards.

The Welcome page appearance can also be customized through the preferences page. You can click the **Customize Page** icon on the top right corner of the Welcome page to open the Preferences dialog (Figure 3-2). You can use this preference page to select one of the pre-defined themes, which affects the overall look of the welcome. You can also select which pages will be displayed, and the visibility, layout, and priority of the items within each page.



Home Page Theme:

- Circles
- Purple Mesh (Classic Eclipse)

Root Pages:

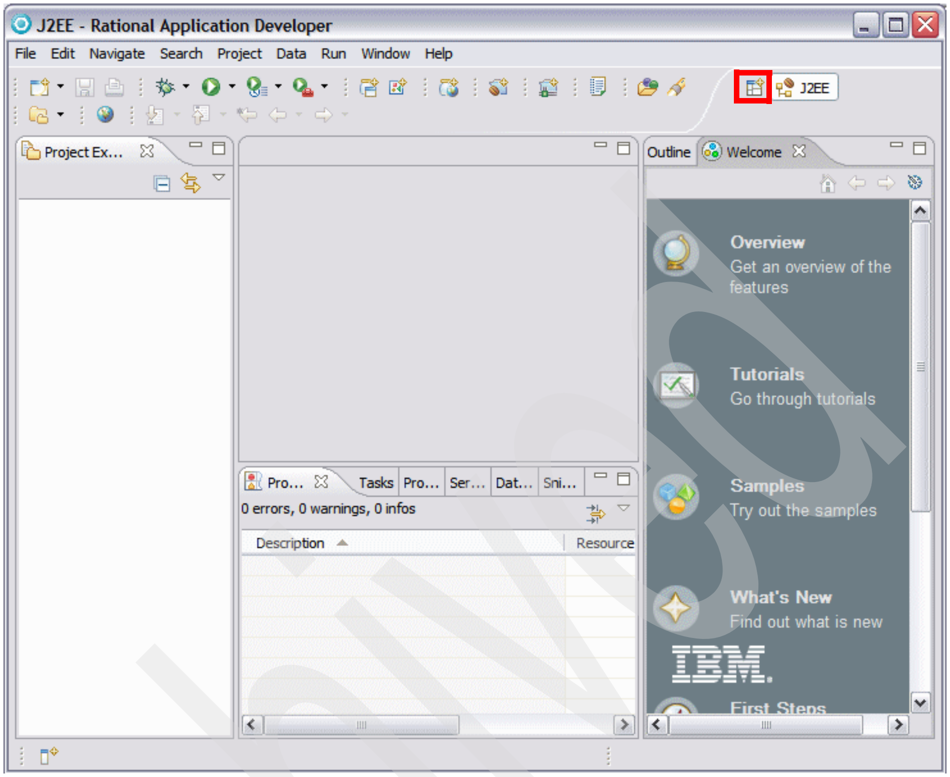
- ☑ Overview
- ☑ First Steps
- ☑ Tutorials
- ☑ Samples
- ☑ What's New
- ☑ Web Resources
- ☐ Migrate

☐ Apply settings to all the products sharing this workbench

Figure 3-2   Welcome page preferences

Users experienced with Application Developer or the concepts that the product provides can close the Welcome page by clicking the **X** for the view to close it down, or clicking the icon in the top right corner arrow. They will then be presented with the default perspective, the J2EE perspective. Each perspective in Application Developer contains multiple views, such as the Project Explorer view, Outline view, and others. More information regarding perspectives and views are provided in Chapter 4, “Perspectives, views, and editors” on page 117.

The top right of the window has a shortcut icon (Figure 3-3), which allows you to open available perspectives, and places them in the shortcut bar next to it. Once the icons are on the shortcut bar, you are able to navigate between perspectives that are already open. The name of the active perspective is shown in the title of the window, and its icon is in the shortcut bar on the right side as a pushed button.



0 errors, 0 warnings, 0 infos

Figure 3-3 J2EE perspective in Rational Application Developer

The term *Workbench* refers to the desktop development environment. Each Workbench window of Application Developer contains one or more perspectives. Perspectives contain views and editors and control what appears in certain menus and toolbars.

Workspace basics

When you start up Application Developer, you are prompted to provide a workspace to start up. On the first startup, the path looks something like the following sample, depending on the installation path:

```
c:\Documents and Settings\<user>\IBM\rationalsdp7.0\workspace
```

Where `<user>` is the Windows user ID you used to login.

The Application Developer workspace is a private work area created for the individual developer, and it holds the following information:

- Application Developer environment meta data, such as configuration information and temporary files

- Projects that they have created, which includes source code, project definition, or configuration files, and generate files, such as class files

Resources that are modified and saved are reflected on the local file system. Users can have many workspaces on their local file system to contain different projects that they are working on, or different versions. Each of these workspaces can be configured differently, because they have their own copy of meta data with configuration data for that workspace.

Important: Although workspace meta data stores configuration information, this does not mean that the meta data can be transferred between workspaces. In general, we do not recommend copying or using the meta data in one workspace, with another workspace. The recommended approach is to create a new workspace and then configure it appropriately.

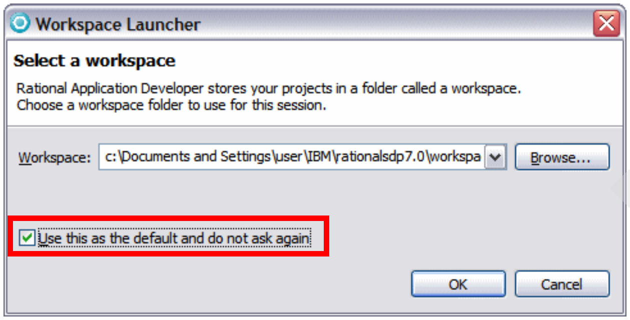
Application Developer allows you to open more than one Workbench at a time. It opens another window into the same workspace, allowing you to work in two differing perspectives. Changes that are made in one window are reflected to the other windows. You are not permitted to work in more than one window at a time, that is, you cannot switch between windows while in the process of using a wizard in one window.

Opening a new Workbench in another window is done by selecting **Window** → **New Window**, and a new Workbench with the same perspective will open in a new window. As well as opening a perspective inside the current Workbench window, new perspectives can also be opened in their own window. By default, new perspectives are opened in the current window. This default behavior can be configured using **Window** → **Preferences** → **General** → **Perspectives** (see “Perspectives preferences” on page 93).

The default workspace can be started on first startup of Application Developer by specifying the workspace location on the local machine and selecting the check box **Use this as the default and do not ask again**, as shown in Figure 3-4.

This ensures that on the next startup of Application Developer that the workspace automatically uses the directory specified initially, and it will not prompt for the workspace in the future.

Note: See “Setting the workspace with a prompt dialog” on page 81 describing how to reconfigure Application Developer prompting for the workspace at startup.


Workspace Launcher

Select a workspace

Rational Application Developer stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: c:\Documents and Settings\user\IBM\rationalsdp7.0\workspa

☑ Use this as the default and do not ask again

Figure 3-4 Setting the default workspace on startup

The other way to enforce the use of a particular workspace is by using the **`-data <workspace>`** command-line argument on Application Developer, where `<workspace>` is a path name on the local machine where the workspace is located, and should be a full path name to remove any ambiguity of location of the workspace.

> **Tip:** On a machine where there are multiple workspaces used by the developer, a shortcut would be the recommended approach in setting up the starting workspace location. The target would be:
>
> ```
> "<RAD Install Dir>\eclipse.exe" -product com.ibm.rational.rad.product.ide
>                                 -data <workspace>
> ```

By using the `-data` argument, you can start a second instance of Application Developer that uses a different workspace. For example, if your second instance should use the `MyWorkspace` folder, you can launch Application Developer with this command (assuming that the product has been installed in the default installation directory):

```
c:\Program Files\IBM\SDP70\eclipse.exe -data c:\MyWorkspace
```

There are a number of arguments that you can add when launching Application Developer; some useful arguments are explained in Table 3-2. More advanced arguments can be found by searching for `Running Eclipse` in **Help** → **Help Contents**.

Table 3-2 Startup parameters

Command	Description
`-configuration` *`configurationFileURL`*	The location for the Platform configuration file, expressed as a URL. The configuration file determines the location of the Platform, the set of available plug-ins, and the primary feature. Note that relative URLs are not allowed. The configuration file is written to this location when Application Developer is installed or updated.
`-consolelog`	Mirrors the Eclipse platform's error log to the console used to run Eclipse. Handy when combined with **`-debug`**.
`-data` *`<workspace directory>`*	Starts Application Developer with a specific workspace located in <workspace directory>.
`-debug` *`[optionsFile]`*	Puts the platform in debug mode and loads the debug options from the file at the given location, if specified. This file indicates which debug points are available for a plug-in and whether or not they are enabled. If a file location is not given, the platform looks in the directory that eclipse was started from for a file called `.options`. Both URLs and file system paths are allowed as file locations.
`-refresh`	Option for performing a global fresh of the workspace on startup to reconcile any changes made on the file system since the platform was last run.
`-showlocation` *`[workspaceName]`*	Option for displaying the location of the workspace in the window title bar. In 3.2, an optional workspace name argument was added that displays the provided name in the window title bar instead of the location of the workspace.
`-vm vmPath`	This optional option allows you to set the location of Java Runtime Environment (JRE) to run Application Developer. Relative paths are interpreted relative to the directory that Eclipse was started from.
`-vmargs -Xmx512M`	For large-scale development, you should modify your VM arguments to make more heap available. This example allows the Java heap to grow to 256 MB. This might not be enough for large projects.

Memory considerations

Use the `-vmargs` argument to set limits to the memory that is used by Application Developer. For example, with 1 GB RAM, you might be able to get better performance by limiting the memory:

```
-vmargs -Xmx512M
```

You can also modify `VMArgs` initialization parameters in the `eclipse.ini` file (under the installation directory):

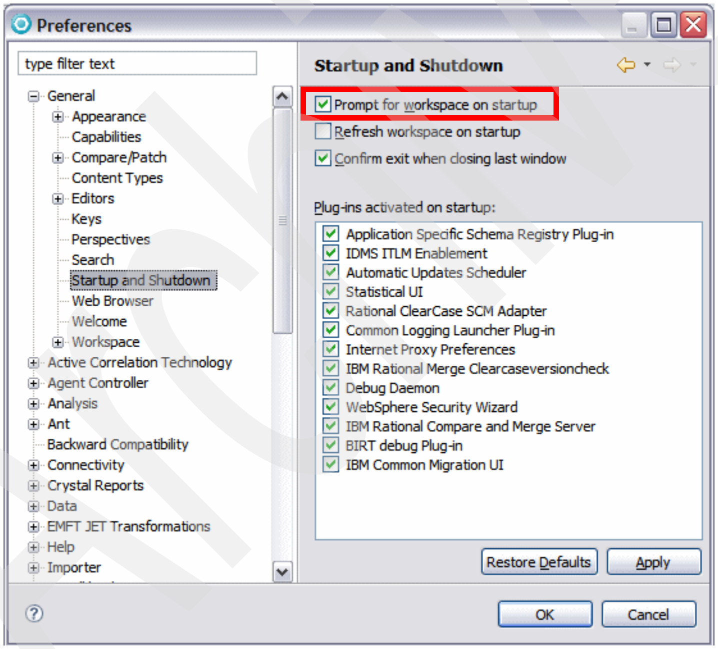
```
VMArgs=-Xms256M -Xmx512M
```

These arguments significantly limit the memory utilization. Setting the `-Xmx` argument below 512M does begin to degrade performance.

Setting the workspace with a prompt dialog

The default behavior on installation is that Application Developer prompts for the workspace on startup. If you selected the check box on the startup screen to not ask again (Figure 3-4 on page 79), there is a procedure to turn on this option, described as follows:

- Select **Window** → **Preferences**.
- In the Preferences dialog select **General** → **Startup and Shutdown**.
- Select **Prompt for workspace on startup** and click **OK** (Figure 3-5).



Startup and Shutdown

☑ Prompt for workspace on startup
□ Refresh workspace on startup
☑ Confirm exit when closing last window

Plug-ins activated on startup:

☑ Application Specific Schema Registry Plug-in
☑ IDMS ITLM Enablement
☑ Automatic Updates Scheduler
☑ Statistical UI
☑ Rational ClearCase SCM Adapter
☑ Common Logging Launcher Plug-in
☑ Internet Proxy Preferences
☑ IBM Rational Merge Clearcaseversioncheck
☑ Debug Daemon
☑ WebSphere Security Wizard
☑ IBM Rational Compare and Merge Server
☑ BIRT debug Plug-in
☑ IBM Common Migration UI

Figure 3-5 Setting the prompt dialog box for workspace selection on startup

On the next startup of Application Developer, the workspace prompt dialog appears, asking the user which workspace to use.

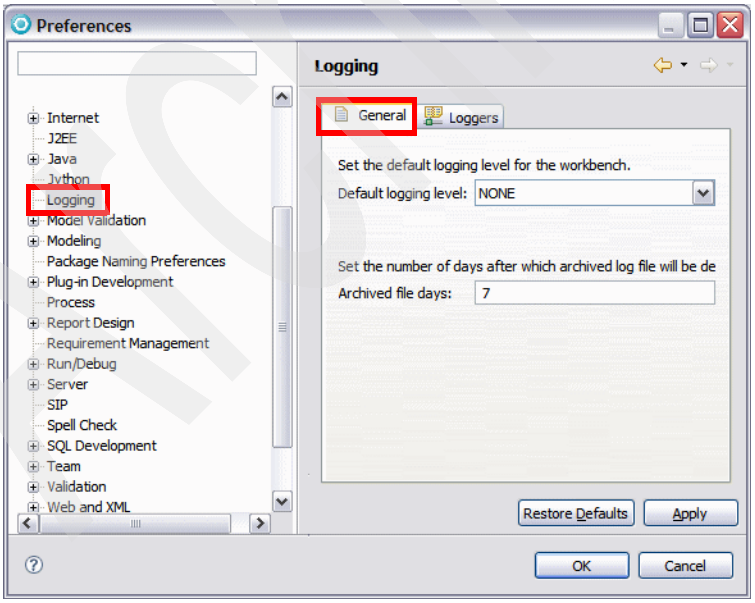
Application Developer logging

Application Developer provides logging functionality for plug-in developers to log and trace important events, primarily expected or unexpected errors. Log files are a crucial part of the Application Developer problem determination process.

The primary reason to use log files is if you encounter unexpected program behavior. In some cases, an error message tells you explicitly to look at the error log.

The logging functionality provided by Application Developer Log and Trace Analyzer implements *common logging*, which enables plug-ins to log events in a common logging log file and logging agent. The Log and Trace Analyzer provides a preferences window to configure the logging level for each of the plug-ins that are configured to log events to the common log file and logging agent. To set the level of records reported to the common log file and logging agent, do these steps:

- Select **Window** → **Preferences**, and select **Logging**.
- Select a default logging level for the workbench from the Default logging level list (Figure 3-6). You can also set the number of days after which archived log files will be deleted. The default is 7 days. If you specify 0 for the number of days, the archived files is never deleted.



Set the default logging level for the workbench.

Default logging level: NONE

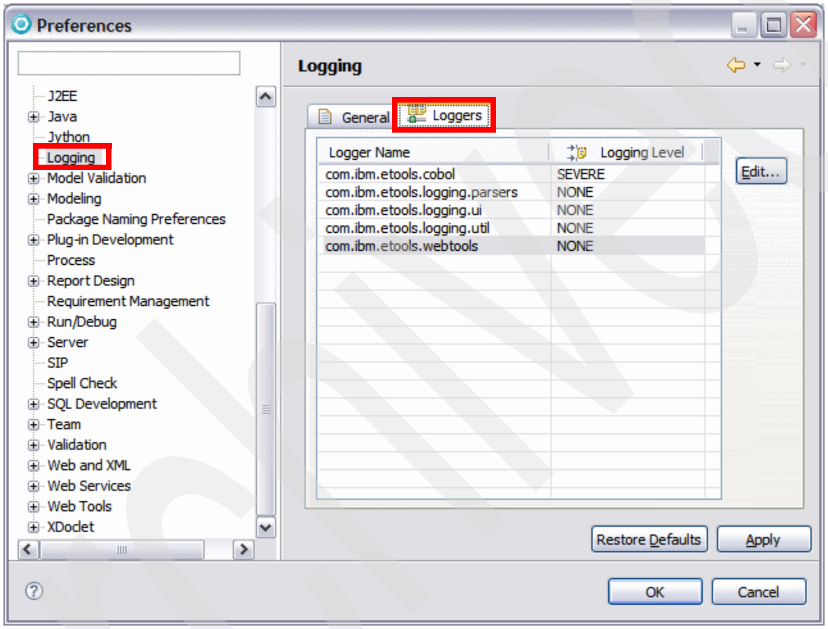
Set the number of days after which archived log file will be de

Archived file days: 7

Figure 3-6 Logging preferences: General tab

> **Note:** Logging is turned off by default unless plug-ins explicitly define a logging level in their `plugin.xml` file. Changing the message level from `NONE` to any other level will enable logging.

- Select the **Loggers** tab.
- For each plug-in, you can select the logging level. Only messages with the same or higher logging level than the logging level selected will be logged (Figure 3-7).



Logger Name	Logging Level
com.ibm.etools.cobol	SEVERE
com.ibm.etools.logging.parsers	NONE
com.ibm.etools.logging.ui	NONE
com.ibm.etools.logging.util	NONE
com.ibm.etools.webtools	NONE

Figure 3-7 Logging preferences: Loggers tab

Log files

There are two main log files in the **`.metadata`** directory of the workspace folder:

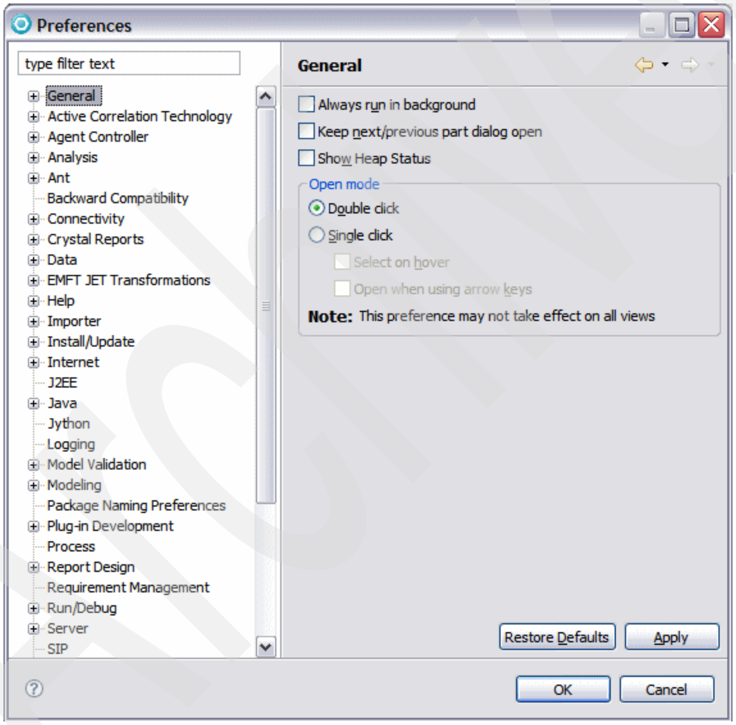
- **CommonBaseEvents.xml**: This log file is usually of interest to plug-in developers. A new `CommonBaseEvents.xml` file is created every time you start the workbench and the previous `CommonBaseEvents.xml` file is archived. Archived file names have the form `CommonBaseEvents`*`timestamp`*`.xml` where timestamp is the standard Java timestamp.

  This log file can be imported into Log and Trace Analyzer for viewing, analysis, sorting, filtering, and correlation.

- **.log**: The `.log` file is used by the Application Developer to capture errors and any uncaught exceptions from plug-ins. The `.log` file is cumulative, as each new session of Application Developer appends its messages to the end of the `.log` file without deleting any previous messages. This enables you to see a history of past messages over multiple Application Developer sessions, each one starting with the `!SESSION` string. This file is an ASCII file and can be viewed with a text editor.

Preferences

The Application Developer Workbench preferences can be modified by selecting **Window** → **Preferences** (Figure 3-8).



General

□ Always run in background
□ Keep next/previous part dialog open
□ Show Heap Status

Open mode
● Double click
○ Single click
□ Select on hover
□ Open when using arrow keys

Note: This preference may not take effect on all views

Figure 3-8 Workbench preferences

In the left pane you can navigate through category of preferences. Each preference has its own page, where you can change the initial options. The Preferences dialog pages can be searched using the filter function.

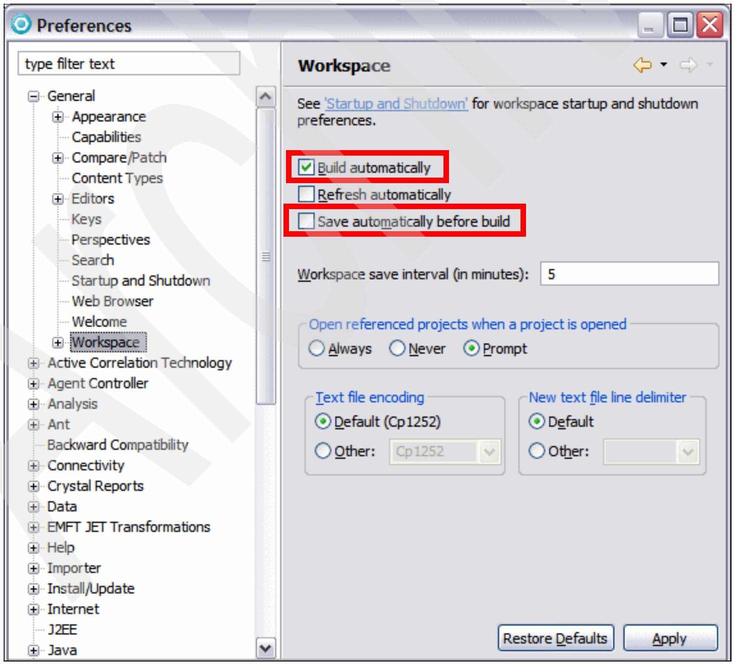
This section describes the most important workbench preferences. Application Developer contains a complete description of all the options available in the preferences dialogs and Application Developer's help.

> **Tip:** Each page of Application Developer's preferences dialog contains a **Restore Defaults** button (see Figure 3-8 on page 84). If the button is clicked, Application Developer restores the settings of the current dialog to their initial values.

Automatic builds

Builds, or a compilation of Java code in Application Developer, are done automatically whenever a resource has been modified and saved. If you require more control regarding builds, you can disable the automatic build feature, then to perform a build you have to explicitly start it. This might be desirable in cases where you know that building is of no value until you finish a large set of changes.

If you want to turn off the automatic build feature, select **Windows** → **Preferences** → **General** → **Workspace** and clear **Build automatically** (Figure 3-9).



Workspace

See 'Startup and Shutdown' for workspace startup and shutdown preferences.

☑ Build automatically
□ Refresh automatically
□ Save automatically before build

Workspace save interval (in minutes): 5

Open referenced projects when a project is opened
○ Always ○ Never ● Prompt

Text file encoding
● Default (Cp1252)
○ Other: Cp1252

New text file line delimiter
● Default
○ Other:

Figure 3-9 Workbench Preferences: Automatic builds

In this same dialog you can specify whether you want unsaved resources to be saved before performing a manual build. Select **Save automatically before build** to enable this feature.

Manual builds

Although the automatic build feature might be adequate for many developers, there are a couple of scenarios in which a developer might want to perform a build manually. First, some developers do not want to build automatically because it can slow down development. In this case the developer needs a method of building at the time of their choosing. Second, there are cases when you want to force a build of a project or all projects to resolve build errors and dependency issues. To address these types of issues, Application Developer provides the ability to perform a manual build, known as a *clean* build.

To perform a manual build, do these steps:

- Select the desired project in the Project Explorer view.
- Select **Project** → **Build Automatically** to clear the check associated with that selection. Manual build option is only available when the automatic build is disabled.
- Select **Project** → **Build Project**. Alternatively, select **Project** → **Build All** to build all projects in the workspace. Both of these commands search through the projects and only build the resources that have changed since the last build.

To build all resources, even those that have not changed since the last build, do these steps:

- Select the desired project in the Project Explorer.
- Select **Project** → **Clean**.
- In the Clean dialog select one of the following options and click **OK**:
  - Clean all projects: This performs a build of all projects.
  - Clean selected projects: `<project>` (The project selected in the previous step is selected by default or you can select it from the projects list.)

Capabilities

Application Developer has the ability to enable and disable capabilities in the tooling. The default setup does not enable some of the capabilities, such as team support for Rational ClearCase, Web services development, or profiling and logging.

Capabilities can be enabled in a number of ways in Application Developer. We describe how to enable capabilities through the following mechanisms:

- Welcome page
- Windows preferences
- Opening a perspective

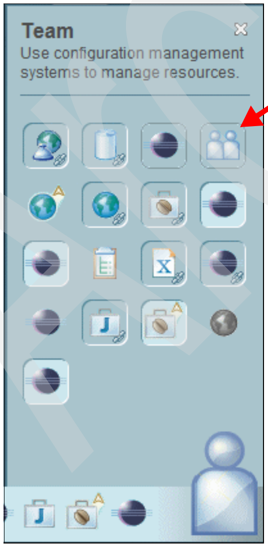
Enable capability through the Welcome page

> **Tip:** You can display the Welcome page by selecting **Help** → **Welcome**.

The Welcome page provides an icon in the shape of a human figure in the bottom right-hand corner used to enable roles (Figure 3-10). These assist in setting up available capabilities for the user of the tool through the following process.

The scenario that we will attempt is enable the team capability or role so that the developer can save their resources in a repository.

- In the Welcome page move the mouse to the bottom right corner over the human figure. Click the 👤 icon.
- Move the mouse until it is over the desired capability or role, and click the icon. For example, move the mouse over the **Team** capability or role 👥 so that it is highlighted and a matching text is shown at the top (Figure 3-10), and click the icon; this enables the Team capability.



Team

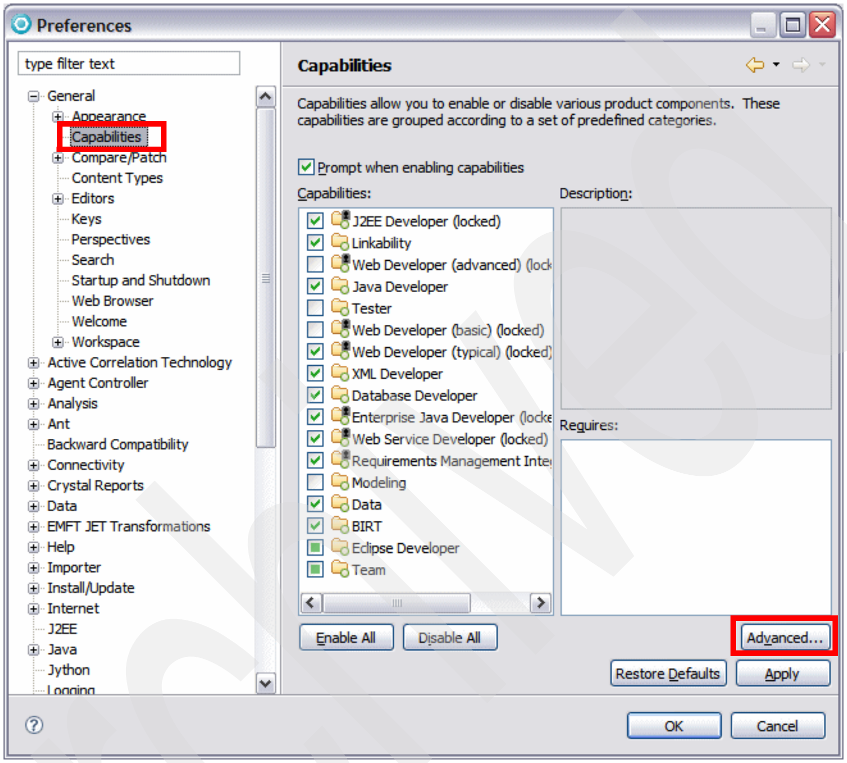
Use configuration management systems to manage resources.

Figure 3-10 Enable Team capability or role in the Welcome page

Enable capability through Windows Preferences

To enable the Team capability using the preferences dialog, do these steps:

- Select **Window** → **Preferences** → **General** → **Capabilities** and click **Advanced** (Figure 3-11).



Capabilities allow you to enable or disable various product components. These capabilities are grouped according to a set of predefined categories.

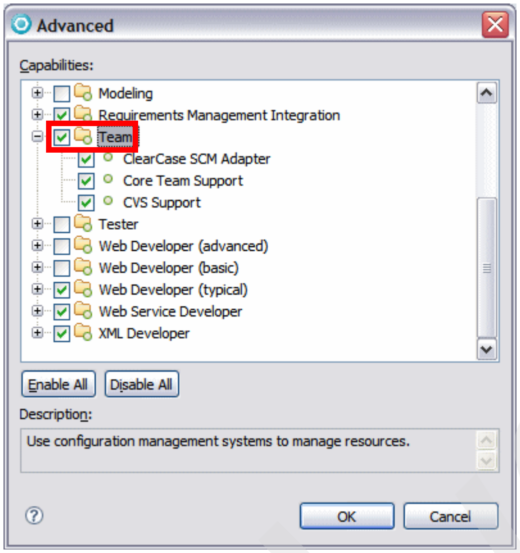
☑ Prompt when enabling capabilities

Capabilities:

☑ J2EE Developer (locked)
☑ Linkability
□ Web Developer (advanced) (lock
☑ Java Developer
□ Tester
□ Web Developer (basic) (locked)
☑ Web Developer (typical) (locked)
☑ XML Developer
☑ Database Developer
☑ Enterprise Java Developer (locke
☑ Web Service Developer (locked)
☑ Requirements Management Inte
□ Modeling
☑ Data
☑ BIRT
▣ Eclipse Developer
▣ Team

Figure 3-11 Setting the Team capability using Windows preferences: Part 1

- In the Advanced dialog expand **Team**.
- Select the **Team** check box, and this selects the check boxes for all components under this tree (Figure 3-12).
- Click **Apply** and then click **OK**. Now all Team capability is enabled.



Capabilities:

- ☐ Modeling
- ☑ Requirements Management Integration
- ☑ Team
  - ☑ ClearCase SCM Adapter
  - ☑ Core Team Support
  - ☑ CVS Support
- ☐ Tester
- ☐ Web Developer (advanced)
- ☐ Web Developer (basic)
- ☑ Web Developer (typical)
- ☑ Web Service Developer
- ☑ XML Developer

Description:

Use configuration management systems to manage resources.

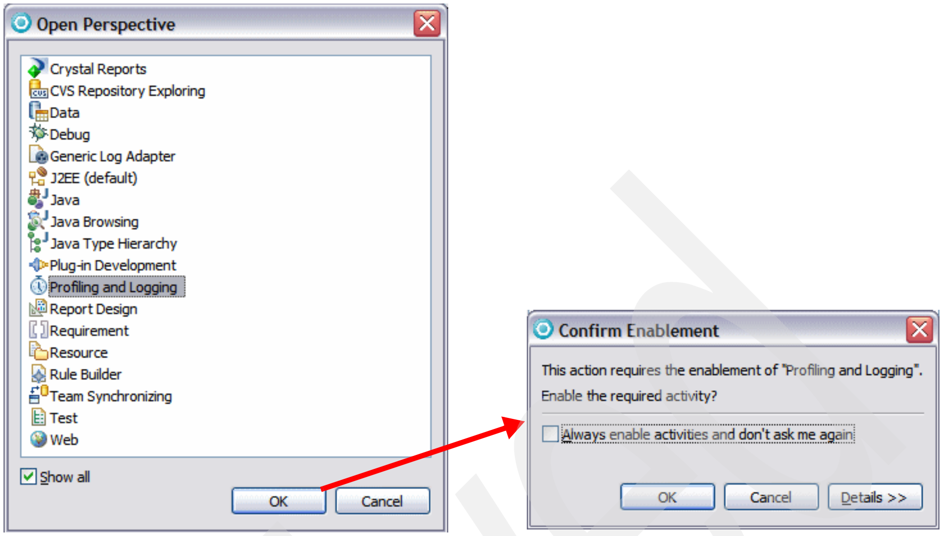
Figure 3-12 Setting the Team capability using Windows preferences: Part 2

Enable a capability by opening a perspective

A capability can be enabled by opening the particular perspective required by the capability. To enable the Profiling and Logging capability by opening a perspective, do these steps:

- Select **Window** → **Open Perspective** → **Other**.
- Select **Show all** and select **Profiling and Logging** (Figure 3-13).
- Click **OK**.
- In the Confirm Enablement dialog click **OK**, and optionally select **Always enable capabilities and don't ask me again**.

This enables the Profiling and Logging capability and open the Profiling and Logging perspective.



This action requires the enablement of "Profiling and Logging". Enable the required activity?

Figure 3-13 Enable capability by opening a perspective

File associations

The File Associations preferences page enables you to add or remove file types recognized by the Workbench. You can also associate editors or external programs with file types in the file types list. To open the preferences page, do these steps:

- Select **Window** → **Preferences** → **General** → **Editors** → **File Associations** (Figure 3-14).

The top right pane allows you to add and remove the file types. The bottom right pane allows you to add or remove the associated editors.

To add a file association, do these steps:

- We add the Internet Explorer® as an additional program to open `.ddl` (database definition language) files. Select ***.ddl** from the file types list and click **Add** next to the associated editors pane.
- In the Editor Selection dialog select **External Programs** and click **Browse**.
- Locate **`iexplore.exe`** in the folder where Internet Explorer is installed (for example, `C:\Program Files\Internet Explorer`), and click **Open**.
- Click **OK** in the Editor Selection dialog and the program is added to the editors list.

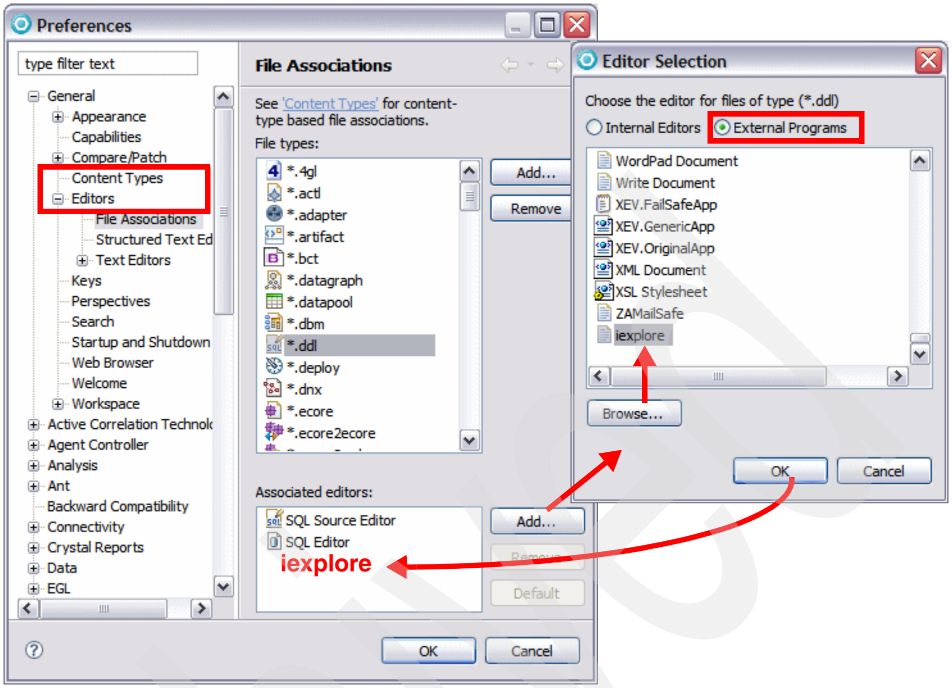


Figure 3-14 File associations preferences

> **Note:** Optionally, you can set this program as the default program for this file type by clicking **Default**.

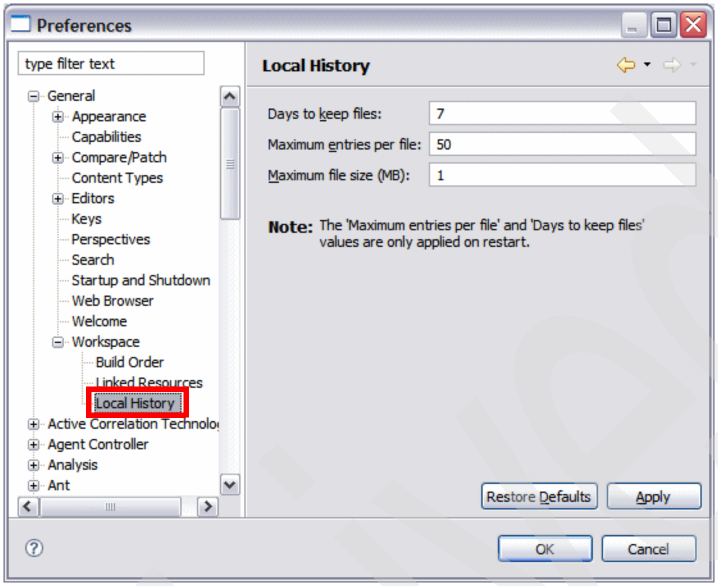
Now you can open a `.ddl` file by using the context menu on the file and selecting **Open With**, and selecting the appropriate program.

Local history

A local history of a file is maintained when you create or modify a file. A copy is saved each time you edit and save the file. This allows you to replace the current file with a previous edition or even restore a deleted file. You can also compare the content of all the local editions. Each edition in the local history is uniquely represented by the data and time the file has been saved.

> **Note:** Only files have local history. Projects and folders do not have a local history.

To configure local history settings, select **Window** → **Preferences** → **General** → **Workspace** → **Local History** to open its preferences page (Figure 3-15).



Days to keep files: 7

Maximum entries per file: 50

Maximum file size (MB): 1

Note: The 'Maximum entries per file' and 'Days to keep files' values are only applied on restart.

Figure 3-15 Local history preferences

Table 3-3 explains the options for the local history preferences.

Table 3-3 Local history settings

Option	Description
Days to keep files	Indicates for how many days you want to maintain changes in the local history. History states older than this value are lost.
Maximum entries per file	Indicates how many history states per resource you want to maintain in the local history. If you exceed this value, you will lose older history to make room for new history.
Maximum file size (MB)	Indicates the maximum size of individual states in the history store. If a resource is over this size, no local history is kept for that resource.

Compare, replace, and restore local history

To compare a file with the local history, do these steps:

- This procedure assumes that you have a Java file in your workspace. If you do not, you should add or create a file.

- Select the Java file, right-click, and select **Compare With** → **Local History**.
- In the upper pane of the Compare with Local History dialog, all available editions of the file in the local history are displayed.

  Select an edition in the upper pane to view the differences between the selected edition and the edition in the workbench.
- If you are done with the comparison, click **OK**.

To replace a file with an edition from the local history, do these steps:

- This assumes you have a Java file in your workspace. If you do not, add or create a file.
- Select the file, right-click, and select **Replace With** → **Local History**.
- Select the desired file time stamp and then click **Replace**.

To restore a deleted file from the local history, do these steps:

- Select the folder or project into which you want to restore the deleted file.
- Right-click and select **Restore from Local History**.
- Select the files that you want to restore and click **Restore**.

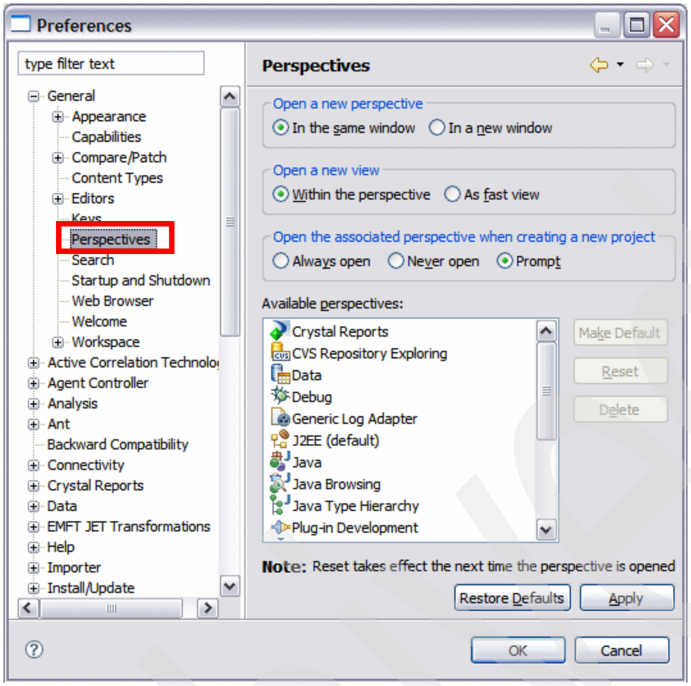
Perspectives preferences

The Perspectives preferences page enables you to manage the various perspectives defined in the Workbench. To open the page, select **Window** → **Preferences** → **General** → **Perspectives** (Figure 3-16).

You can change the following options in the Perspective preferences:

- Open a new perspective in the same or in a new window.
- Open a new view within the perspective or as a fast view (docked to the side of the current perspective).
- The option to always switch, never switch, or prompt when a particular project is created to the appropriate perspective.

There is also a list with all available perspectives where you can select the default perspective. If you have added one or more customized perspectives, you can delete them from here.



Perspectives

Open a new perspective
● In the same window ○ In a new window

Open a new view
● Within the perspective ○ As fast view

Open the associated perspective when creating a new project
○ Always open ○ Never open ● Prompt

Available perspectives:
- Crystal Reports
- CVS Repository Exploring
- Data
- Debug
- Generic Log Adapter
- J2EE (default)
- Java
- Java Browsing
- Java Type Hierarchy
- Plug-in Development

Note: Reset takes effect the next time the perspective is opened

Figure 3-16 Perspectives preferences

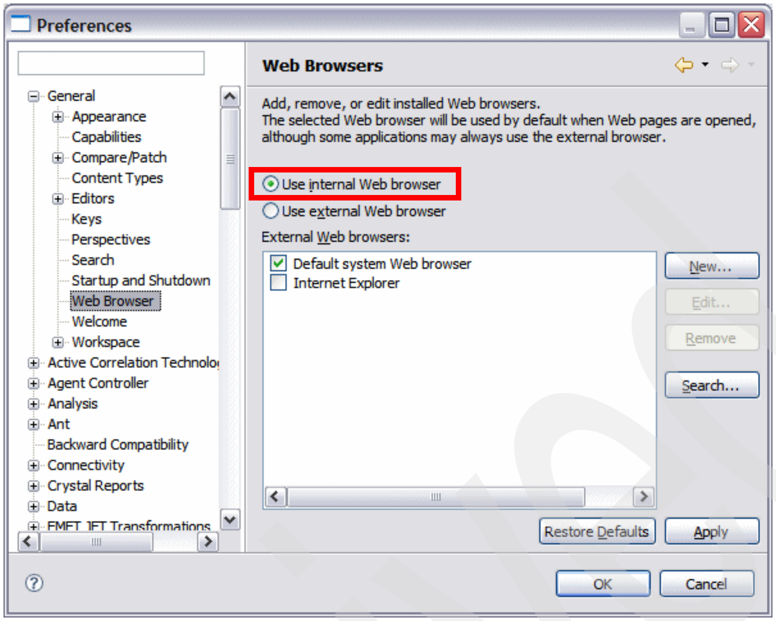
Web Browser preferences

The Web browser settings allow the user to select which Web browser is the default browser used by Application Developer for displaying Web information.

To change the Web browser settings,, do these steps:

- Select **Window** → **Preferences** → **General** → **Web Browser** (Figure 3-17).

The default option is to use the internal Web browser. To change, select **Use external Web browser** and select a browser from the available list; otherwise you can click **New** to add a new Web browser.



Add, remove, or edit installed Web browsers.
The selected Web browser will be used by default when Web pages are opened, although some applications may always use the external browser.

● Use internal Web browser
○ Use external Web browser

External Web browsers:

☑ Default system Web browser
□ Internet Explorer

Figure 3-17 Web Browser preferences

Internet preferences

The Internet preferences in Application Developer have three types of settings available to be configured:

- Cache
- FTP
- Proxy settings

Only proxy settings is covered in this section, with the other two settings you can refer to Application Developer's help.

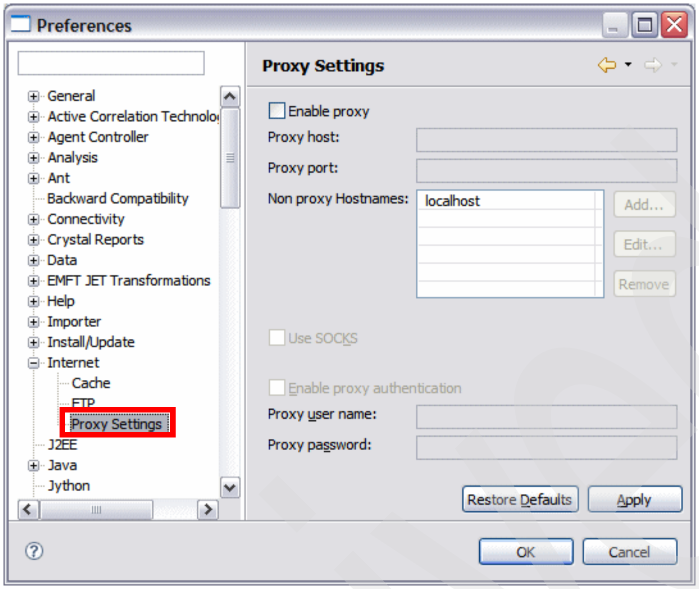
Proxy settings

When using Application Developer and working within an intranet, you might want to use a proxy server to get across a company firewall to access the Internet.

To set the preferences for the HTTP proxy server within the Workbench to allow Internet access from Application Developer, do these steps:

- Select **Window** → **Preferences** → **Internet** → **Proxy Settings** (Figure 3-18).
- Select **Enable Proxy** and enter the proxy host and port. There are additional optional settings for the use of SOCKS and enabling proxy authentication.

- Click **Apply** and then **OK**.



Proxy Settings

□ Enable proxy

Proxy host:

Proxy port:

Non proxy Hostnames: localhost

□ Use SOCKS

□ Enable proxy authentication

Proxy user name:

Proxy password:

Figure 3-18 Internet proxy settings preferences

Java development preferences

Application Developer provides a number of preferences in different categories such as Java, Web tools, XML, SQL development, and plug-in development. This section only covers the most commonly used Java development preferences. More information about the other preferences are provided in relevant chapters of this book.

Java classpath variables

Application Developer provides a number of default classpath variables that can be used in a Java build path to avoid a direct reference to the local file system in a project. This method ensures that the project only references classpaths using the variable names and not specific local file system directories or paths. This is a good programming methodology when developing within a team and using multiple projects using the same variables. This means that all team members have to set the variables required for a project, and this data is maintained in the workspace.

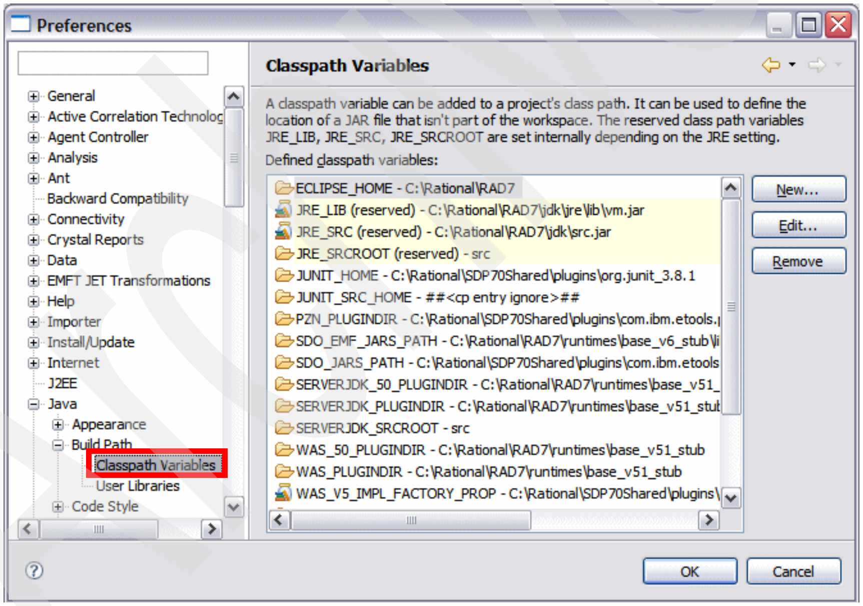
Tip: We recommend that you standardize the Application Developer installation path for your development team. Many files within the projects have absolute paths based on the Application Developer installation path, thus when you import projects from a team repository such as CVS or ClearCase, you will get errors even when using classpath variables.

Depending on the type of Java coding you plan to do, you might have to add variables pointing to other code libraries. For example, this can be driver classes to access relational databases or locally developed code that you want to reuse in other projects. Once you have created a Java project, you can add any of these variables to the project's classpath.

To configure the default classpath variables, do these steps:

- Select **Window** → **Preferences** → **Java** → **Build Path** → **Classpath Variables**.

  A list of the existing classpath variables is displayed (Figure 3-19).



Classpath Variables

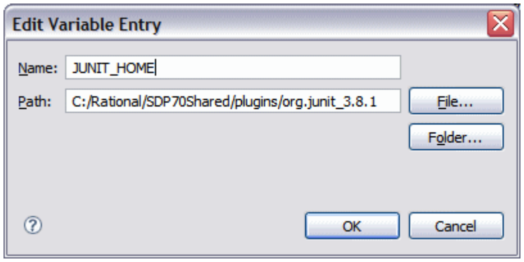
A classpath variable can be added to a project's class path. It can be used to define the location of a JAR file that isn't part of the workspace. The reserved class path variables JRE_LIB, JRE_SRC, JRE_SRCROOT are set internally depending on the JRE setting.

Defined classpath variables:

```
ECLIPSE_HOME - C:\Rational\RAD7
JRE_LIB (reserved) - C:\Rational\RAD7\jdk\jre\lib\vm.jar
JRE_SRC (reserved) - C:\Rational\RAD7\jdk\src.jar
JRE_SRCROOT (reserved) - src
JUNIT_HOME - C:\Rational\SDP70Shared\plugins\org.junit_3.8.1
JUNIT_SRC_HOME - ##<cp entry ignore>##
PZN_PLUGINDIR - C:\Rational\SDP70Shared\plugins\com.ibm.etools.
SDO_EMF_JARS_PATH - C:\Rational\RAD7\runtimes\base_v6_stub\li
SDO_JARS_PATH - C:\Rational\SDP70Shared\plugins\com.ibm.etools
SERVERJDK_50_PLUGINDIR - C:\Rational\RAD7\runtimes\base_v51_
SERVERJDK_PLUGINDIR - C:\Rational\RAD7\runtimes\base_v51_stul
SERVERJDK_SRCROOT - src
WAS_50_PLUGINDIR - C:\Rational\RAD7\runtimes\base_v51_stub
WAS_PLUGINDIR - C:\Rational\RAD7\runtimes\base_v51_stub
WAS_V5_IMPL_FACTORY_PROP - C:\Rational\SDP70Shared\plugins\
```

Figure 3-19 Classpath variables preferences

- Creation, editing, or removing of variables can be performed in this screen. Click **New** to add a new variable.
- In the New Variable Entry dialog, enter the name of the variable and browse for the path and click **OK** (Figure 3-20).



Edit Variable Entry

Name: JUNIT_HOME

Path: C:/Rational/SDP70Shared/plugins/org.junit_3.8.1

Figure 3-20 New Variable Entry dialog

Certain class path variables are set internally and cannot be changed in the Classpath variables preferences:

- `JRE_LIB`: The archive with the runtime JAR file for the currently used JRE.
- `JRE_SRC`: The source archive for the currently used JRE.
- `JRE_SRCROOT`: The root path in the source archive for the currently used JRE.

Appearance of Java elements

The appearance and the settings of associated Java elements in views, such as methods, members, and their access types, can be configured:

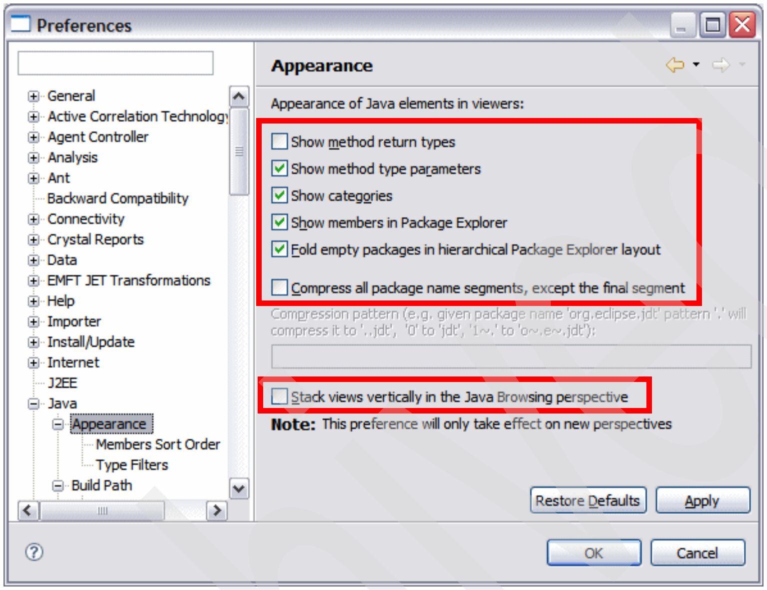
- Select **Window** → **Preferences** → **Java** → **Appearance**.

  The Appearance preferences page is displayed (Figure 3-21) with appearance check boxes as described in Table 3-4.

Table 3-4 Description of appearance settings for Java views

Appearance setting	Description
Show method return types	If selected, methods displayed in views show the return type.
Show method type parameters	If selected, methods displayed in views show their type parameters.
Show categories	If selected, method, field, and type labels contain the categories specified in their Javadoc™ comment.
Show members in Package Explorer	If selected, displays the members of the class and their scope such as private, private or protected, including others.
Fold empty packages in hierarchical layout	If selected, folds the empty packages that do not contain resources or other child elements.
Compress package name segments	If selected, compresses the name of the package based on a pattern supplied in the dialog below the check box.

Appearance setting	Description
Stack views vertically in the Java Browsing perspective	If selected, displays the views in the Java Browsing perspective vertically rather than horizontally.



Appearance

Appearance of Java elements in viewers:

☐ Show method return types
☑ Show method type parameters
☑ Show categories
☑ Show members in Package Explorer
☑ Fold empty packages in hierarchical Package Explorer layout
☐ Compress all package name segments, except the final segment

Compression pattern (e.g. given package name 'org.eclipse.jdt' pattern '.' will compress it to '..jdt', '0' to 'jdt', '1~.' to 'o~.e~.jdt'):

☐ Stack views vertically in the Java Browsing perspective

Note: This preference will only take effect on new perspectives

Figure 3-21 Java appearance settings

To change the appearance of the order of the members to be displayed in the Java viewers, do these steps:

- In the Preferences dialog select **Java** → **Appearance** → **Members Sort Order**. This preference allows you to display the members in the order you prefer, as well as the order of scoping within that type of member.
- Select the order in which members will be displayed using the **Up** and **Down** buttons. You can also sort in same category by visibility.
- Click **Apply** and then **OK**.

To specify types and packages to hide in the Open Type dialog and content assist or quick fix proposals, do these steps:

- In the Preferences dialog select **Java** → **Appearance** → **Type Filters**.
- You can either create a new filter or add packages to the type filter list. The default is to hide nothing.

Code style and formatting

The Java editor in the Workbench can be configured to format code and coding style in conformance to personal preferences or team-defined standards. When setting up the Workbench, you can decide what formatting style should be applied to the Java files created using the wizards, as well as how the Java editors operate to assist what has been defined.

> **Important:** Working in a team environment requires a common understanding between all the members of the team regarding the style of coding and conventions such as class, member, and method name definitions. The coding standards have to be documented and agreed upon to ensure a consistent and standardized method of operation.

Code style

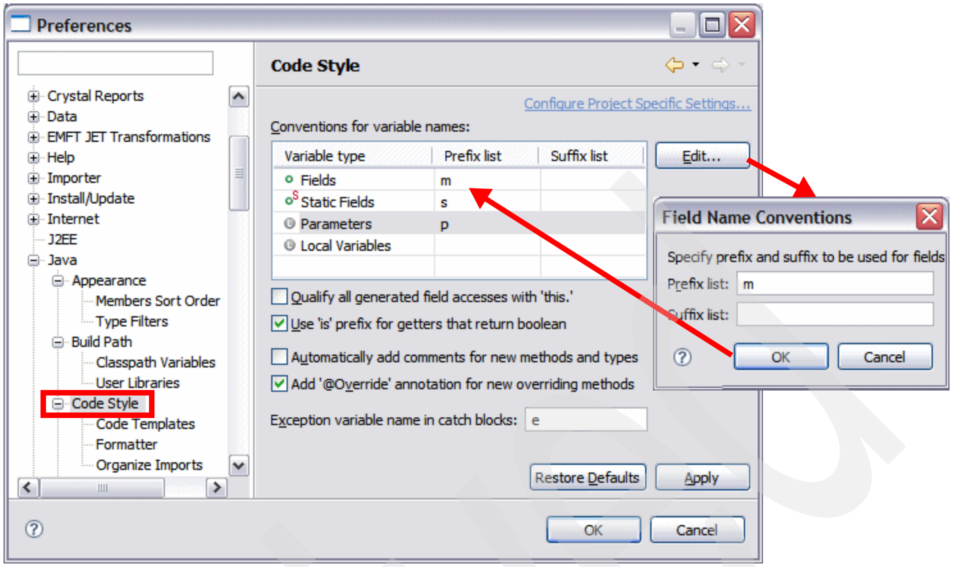
The Java code style preferences allow you to configure naming conventions, style rules and comment settings.

To demonstrate setting up a style, we define a sample style in which the following conventions are defined:

- Member attributes or fields will be prefixed by an `m`.
- Static attributes or fields will be prefixed by an `s`.
- Parameters of methods will be prefixed by a `p`.
- Local variables can have any name.
- Boolean getter methods will have a prefix of `is`.

To configure the customized style, do these steps:

- Select **Windows** → **Preferences** → **Java** → **Code Style** (Figure 3-22).
- Select the **Fields** row and click **Edit**.
- In the Field Name Conventions dialog enter **`m`** in the **Prefix list** field and click **OK**.
- Repeat the same steps for all the conventions defined above.



Code Style

Configure Project Specific Settings...

Conventions for variable names:

Variable type	Prefix list	Suffix list
Fields	m	
Static Fields	s	
Parameters	p	
Local Variables		

☐ Qualify all generated field accesses with 'this.'

☑ Use 'is' prefix for getters that return boolean

☐ Automatically add comments for new methods and types

☑ Add '@Override' annotation for new overriding methods

Exception variable name in catch blocks: e

Field Name Conventions

Specify prefix and suffix to be used for fields

Prefix list: m

Suffix list:

Figure 3-22 Code style preferences

These coding conventions are used in the generation of setters and getters for Java classes:

- Whenever a prefix of `m` followed by a capital letter is found on an attribute, this would ignore the prefix and generate a getter and setter without the prefix.
- If the prefix is not found followed by a capitalized letter, then the setter and getter would be generated with the first letter capitalized followed by the rest of the name of the attribute.

An example of the outcome of performing code generation of a getter is shown in Example 3-1 for some common examples of attributes.

> **Note:** The capitalization of getters in Application Developer is based on the way the attributes are named.

Example 3-1 Example snippet of code generation output for getters

```
 private long mCounter;
 private String maddress;
 private float m_salary;
 private int zipcode;
/**
 * @return the m_salary
 */
public float getM_salary() {
   return m_salary;
}
/**
 * @return the maddress
 */
public String getMaddress() {
   return maddress;
}
/**
 * @return the counter
 */
public long getCounter() {
   return mCounter;
}
/**
 * @return the zipcode
 */
public int getZipcode() {
   return zipcode;
}
```

The settings described in Table 3-5 specify how newly generated code should look like. These settings are configured in the Code Style preferences page (Figure 3-22 on page 101).

Table 3-5 Description of code style settings

Action	Description
Qualify all generated field accesses with 'this.'	If selected, field accesses are always prefixed with **this.**, regardless whether the name of the field is unique in the scope of the field access or not.
Use 'is' prefix for getters that return boolean	If selected, the names of getter methods of boolean type are prefixed with **is** rather than get.
Automatically add comments for new methods and types	If selected, newly generated methods and types are automatically generated with comments where appropriate.
Add '@Override' annotation for overriding methods	If selected, methods which override an already implemented method are annotated with an **@Override** annotation.
Exception variable name in catch blocks	Specify the name of the exception variable declared in catch blocks.

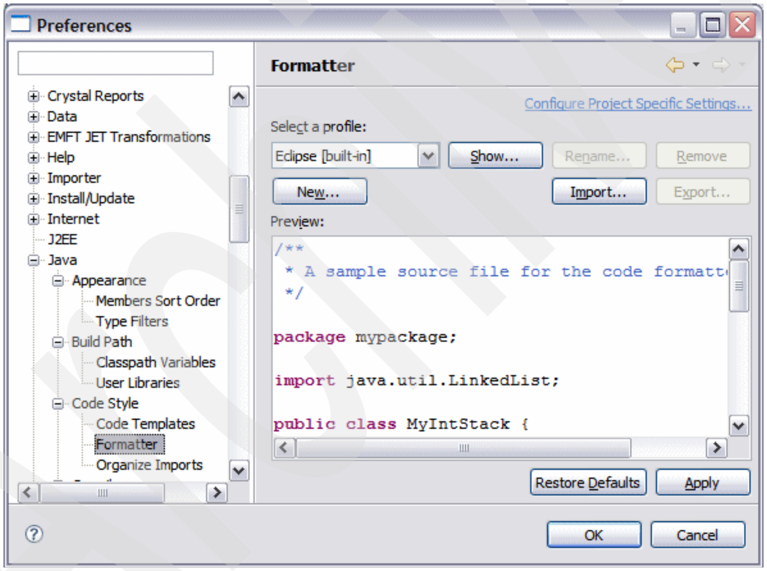
Formatter

The formatter preferences in Application Developer are used to ensure that the format of the Java code meets the standard defined for a team of developers working on code. There are three profiles built-in with Application Developer with the option of creating new profiles specific for your project:

- Eclipse (default profile on startup for Application Developer)
- Eclipse 2.1 (similar to WebSphere Studio V5)
- Java Conventions

To configure the formatter, do these steps:

- Select **Window** → **Preferences** → **Java** → **Code Style** → **Formatter** (Figure 3-23).
- To display the information about a profile, click **Show**, or to create a new profile click **New**.
- An existing profile can also be loaded by clicking **Import**.



Select a profile: Eclipse [built-in]

```
/**
 * A sample source file for the code formatt
 */

package mypackage;

import java.util.LinkedList;

public class MyIntStack {
```

Figure 3-23 Formatter preferences

Enforcing coding standards

The code formatting rules are enforced on code when a developer has entered code using their own style, and which does not conform to the team-defined standard. When this is the case, after the code has been written and tested and preferably before adding the code to the team repository, we recommend that the code is formatted.

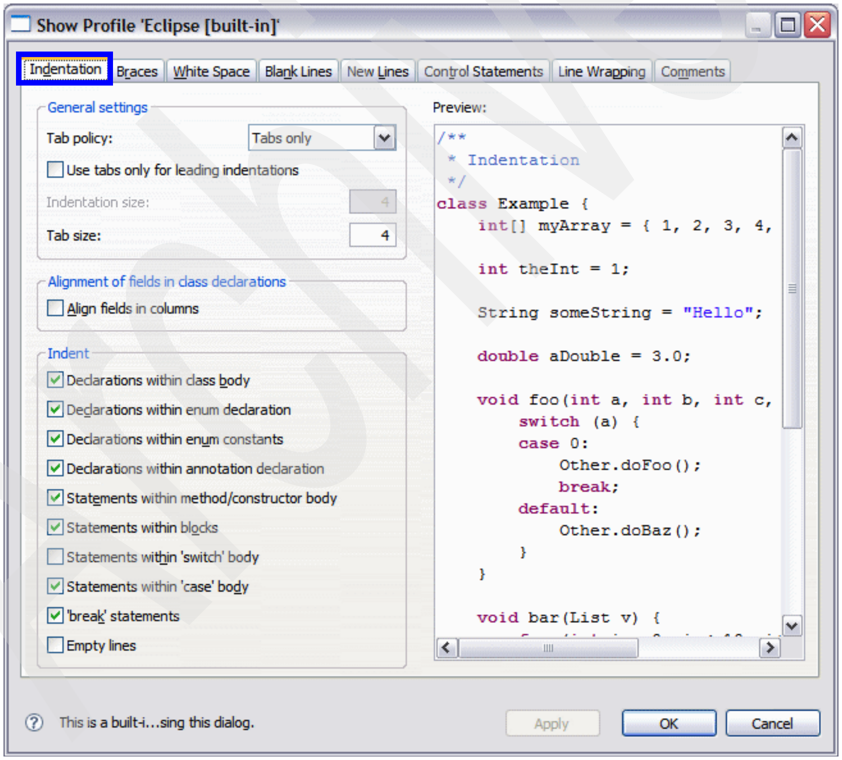
The procedure to perform this operation is described here:

- In the Java editor, right-click, and select **Source → Format**. This formatting uses the rules that have been defined in the formatter preferences.

Creating a user-defined profile

User-defined profiles are established from one of the existing built-in profiles, which can be exported to the file system to share with team members. If the existing profile is modified then you are prompted to create a new profile. Any profile that is created can be exported as an XML file that contains the standardized definitions required for your project. This can then be stored in a team repository and imported by each member of the team.

A profile consists of a number of sections that are provided as tab sections to standardize on the format and style that the Java code is written (Figure 3-24). Each of these tab sections are self-explanatory and provide a preview of the code after selection of the required format.



General settings

Tab policy: Tabs only

☐ Use tabs only for leading indentations

Indentation size: 4

Tab size: 4

Alignment of fields in class declarations

☐ Align fields in columns

Indent

☑ Declarations within class body
☑ Declarations within enum declaration
☑ Declarations within enum constants
☑ Declarations within annotation declaration
☑ Statements within method/constructor body
☑ Statements within blocks
☐ Statements within 'switch' body
☑ Statements within 'case' body
☑ 'break' statements
☐ Empty lines

Preview:

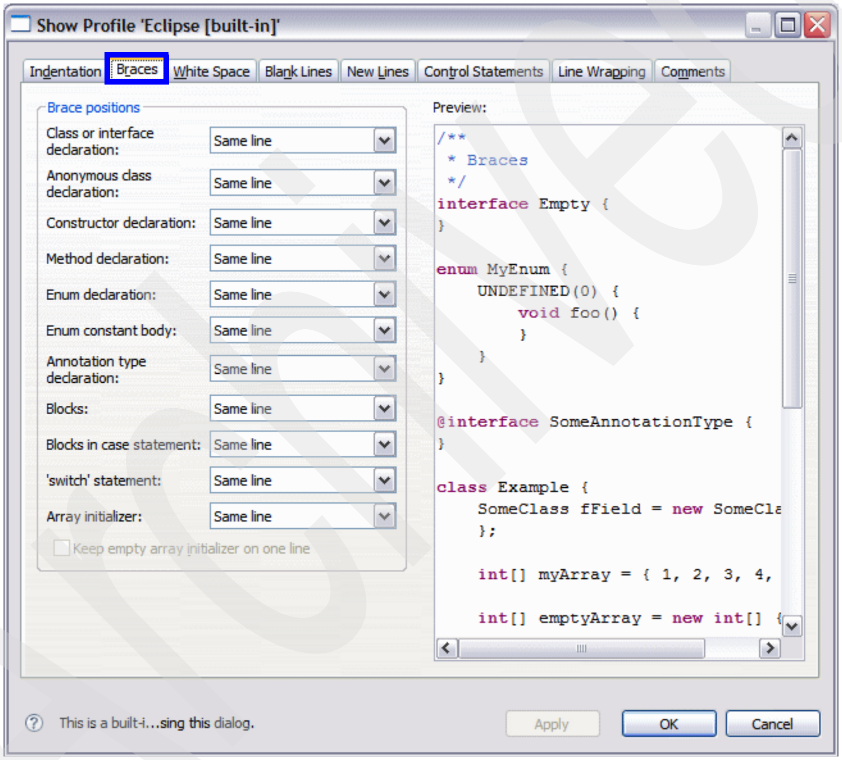
```
/**
 * Indentation
 */
class Example {
    int[] myArray = { 1, 2, 3, 4,

    int theInt = 1;

    String someString = "Hello";

    double aDouble = 3.0;

    void foo(int a, int b, int c,
        switch (a) {
        case 0:
            Other.doFoo();
            break;
        default:
            Other.doBaz();
        }
    }

    void bar(List v) {
```

This is a built-i...sing this dialog.

Figure 3-24 Formatter preferences: Eclipse profile

The definition of these tabs are described as follows:

- **Indentation**: Specifies the indentations that you wish on your Java code in the Workbench (Figure 3-24). The area it covers includes:
  - Tab spacing
  - Alignment of fields
  - Indentation of code
- **Braces**: Formats the Java style of where braces are placed for a number of Java language concepts. A preview is provided as you select the check boxes to ensure that it fits in with the guidelines established in your team (Figure 3-25).



Brace positions

Class or interface declaration: Same line
Anonymous class declaration: Same line
Constructor declaration: Same line
Method declaration: Same line
Enum declaration: Same line
Enum constant body: Same line
Annotation type declaration: Same line
Blocks: Same line
Blocks in case statement: Same line
'switch' statement: Same line
Array initializer: Same line
□ Keep empty array initializer on one line

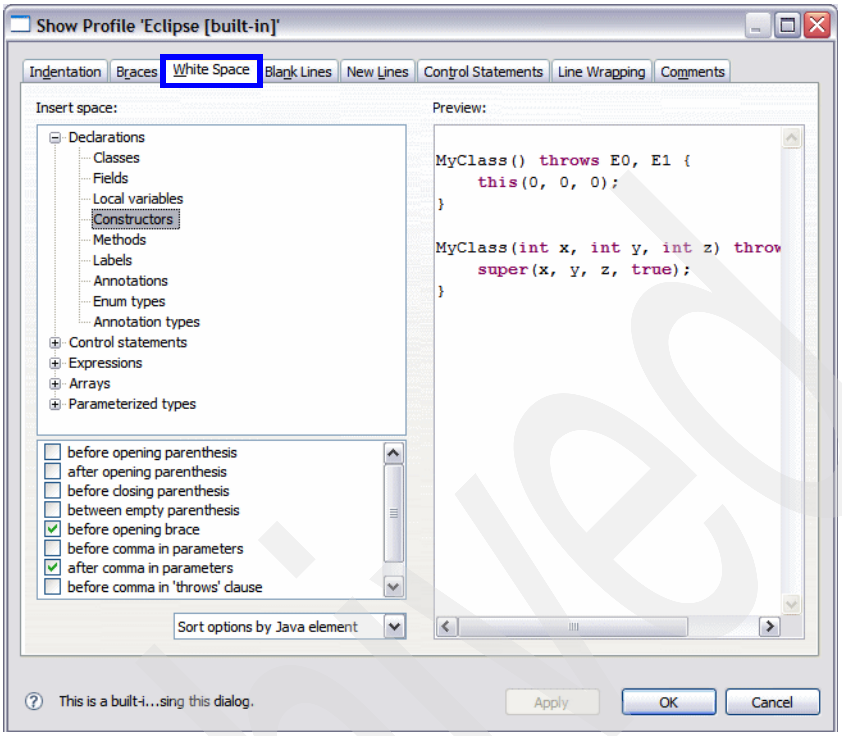
Preview:

```
/**
 * Braces
 */
interface Empty {
}

enum MyEnum {
    UNDEFINED(0) {
        void foo() {
        }
    }
}

@interface SomeAnnotationType {
}

class Example {
    SomeClass fField = new SomeCla
    };

    int[] myArray = { 1, 2, 3, 4,

    int[] emptyArray = new int[] {
```

This is a built-i...sing this dialog.

Figure 3-25 Formatter: Braces

- **White Space**: Format where the spaces are placed in the code based on a number of Java constructs (Figure 3-26).



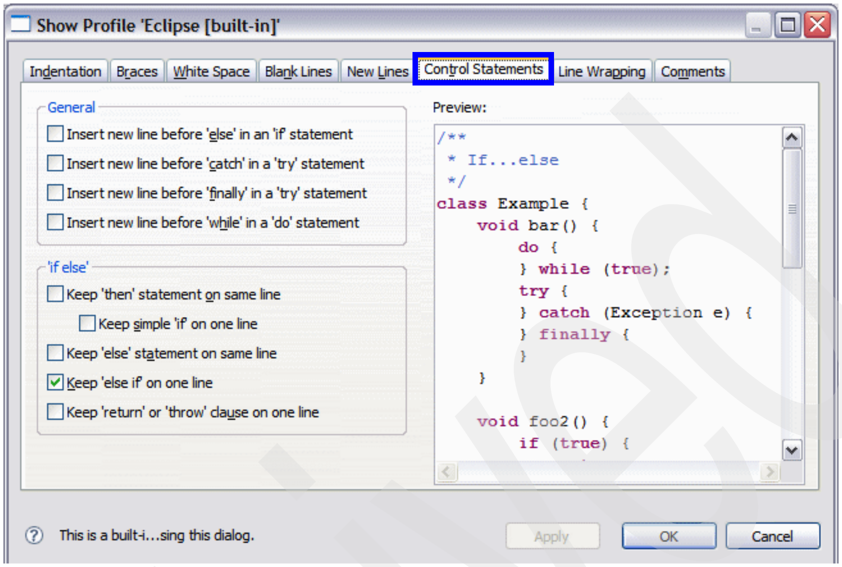
- ☐ before opening parenthesis
- ☐ after opening parenthesis
- ☐ before closing parenthesis
- ☐ between empty parenthesis
- ☑ before opening brace
- ☐ before comma in parameters
- ☑ after comma in parameters
- ☐ before comma in 'throws' clause

```
MyClass() throws E0, E1 {
    this(0, 0, 0);
}

MyClass(int x, int y, int z) throw
    super(x, y, z, true);
}
```

Figure 3-26 Formatter: White Space

- **Blank Lines**: Specify where you want to place blank lines in the code for readability or style guidelines, for example:
  - Before and after package declaration
  - Before and after import declaration
  - Within a class:
    - Before first declaration
    - Before field declarations
    - Before method declarations
    - At the beginning of a method body
- **New Lines**: Specifies the option of where you want to insert a new line, for example:
  - In empty class body
  - In empty method body
  - In empty annotation body
  - At end of file

- **Control Statements**: Control the insertion of lines in control statements, as well as the appearance of *if else* statements of the Java code (Figure 3-27).



General

- ☐ Insert new line before 'else' in an 'if' statement
- ☐ Insert new line before 'catch' in a 'try' statement
- ☐ Insert new line before 'finally' in a 'try' statement
- ☐ Insert new line before 'while' in a 'do' statement

'if else'

- ☐ Keep 'then' statement on same line
    - ☐ Keep simple 'if' on one line
- ☐ Keep 'else' statement on same line
- ☑ Keep 'else if' on one line
- ☐ Keep 'return' or 'throw' clause on one line

```
/**
 * If...else
 */
class Example {
    void bar() {
        do {
        } while (true);
        try {
        } catch (Exception e) {
        } finally {
        }
    }

    void foo2() {
        if (true) {
```

This is a built-i...sing this dialog.

Figure 3-27   Formatter: Control Statements

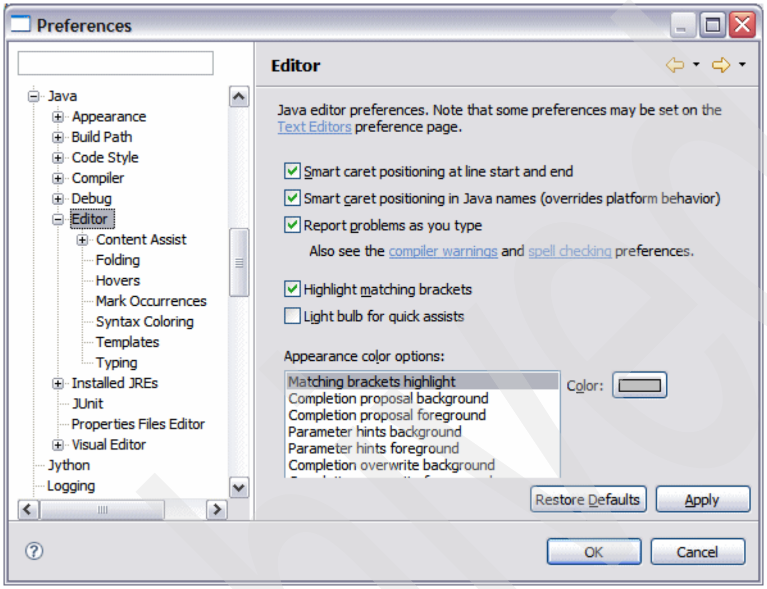
- **Line Wrapping**: Provides the style rule on what should be performed with the wrapping of code, for example:
    - – Maximum line width (default is 80)
    - – Default indentation (2)
    - – How declarations, constructors, function calls, and expressions are wrapped
- **Comments**: Determine the rules of the look of general comments and of Javadoc that are in the Java code.

Java editor settings

The Java editor has a number of settings that assist in the productivity of the user in Application Developer. Most of these options relate to the look and feel of the Java editor in the Workbench. To configure Java editor preferences, do these steps:

- Select **Windows** → **Preferences** → **Java** → **Editor** (Figure 3-28).

> **Note:** Some options that are generally applicable to text editors can be configured on the text editor preference page under **General → Editors → Text Editors**.



Preferences

Java
- Appearance
- Build Path
- Code Style
- Compiler
- Debug
- Editor
  - Content Assist
  - Folding
  - Hovers
  - Mark Occurrences
  - Syntax Coloring
  - Templates
  - Typing
- Installed JREs
- JUnit
- Properties Files Editor
- Visual Editor

Jython

Logging

Editor

Java editor preferences. Note that some preferences may be set on the Text Editors preference page.

☑ Smart caret positioning at line start and end

☑ Smart caret positioning in Java names (overrides platform behavior)

☑ Report problems as you type

Also see the compiler warnings and spell checking preferences.

☑ Highlight matching brackets

□ Light bulb for quick assists

Appearance color options:

- Matching brackets highlight
- Completion proposal background
- Completion proposal foreground
- Parameter hints background
- Parameter hints foreground
- Completion overwrite background

Figure 3-28 Java editor preferences

The main Java editor settings are described in Table 3-6.

Table 3-6 Description of Java editor settings

Option	Description
Smart caret positioning at line start and end	If selected, the Home and End commands jump to the first and last non white space character on a line.
Smart caret positioning in Java names (overrides platform behavior)	If selected, there are additional word boundaries inside `|Camel|Case|` Java names.
Report problems as you type	If selected, the editor marks errors and warnings as you type, even if you do not save the editor contents. The problems are updated after a short delay.
Highlight matching brackets	If selected, whenever the cursor is next to a parenthesis, bracket or curly braces, its opening or closing counter part is highlighted. The color of the bracket highlight is specified with Appearance color options.

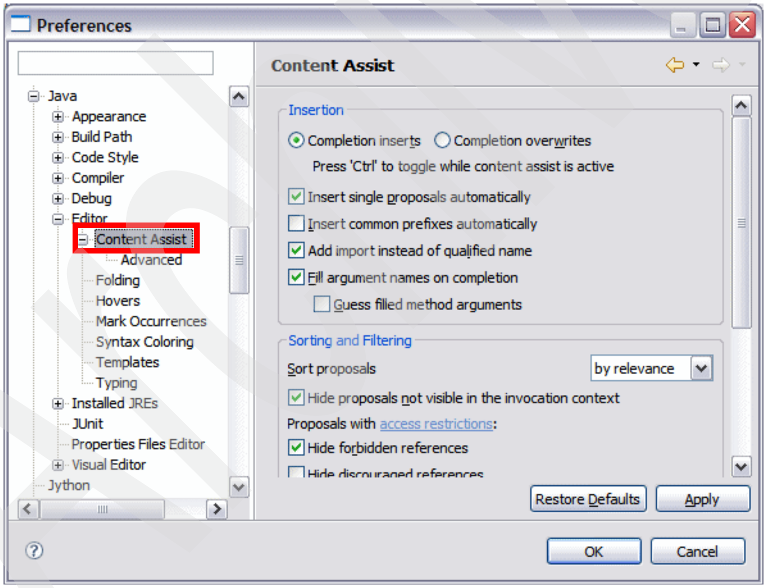
Option	Description
Light bulb for quick assists	If selected, a light bulb shows up in the vertical ruler whenever a quick assist is available.
Appearance color options	The colors of various Java editor appearance features are specified here.

Some of Java editor preferences sub-pages are described as follows. A detailed description of each sub-page can be found in the Application Developer help.

Content assist

The content assist feature in Application Developer is used in assisting a developer in writing their code rapidly and reducing the errors in what they are writing. It is a feature that is triggered by pressing **Ctrl+Spacebar**, and assists the developer in completion of a variable or method name when coding.

To configure content assist, select **Windows** → **Preferences** → **Java** → **Editor** → **Content Assist** (Figure 3-29).



Content Assist

Insertion

● Completion inserts ○ Completion overwrites

Press 'Ctrl' to toggle while content assist is active

☑ Insert single proposals automatically

☐ Insert common prefixes automatically

☑ Add import instead of qualified name

☑ Fill argument names on completion

☐ Guess filled method arguments

Sorting and Filtering

Sort proposals: by relevance

☑ Hide proposals not visible in the invocation context

Proposals with access restrictions:

☑ Hide forbidden references

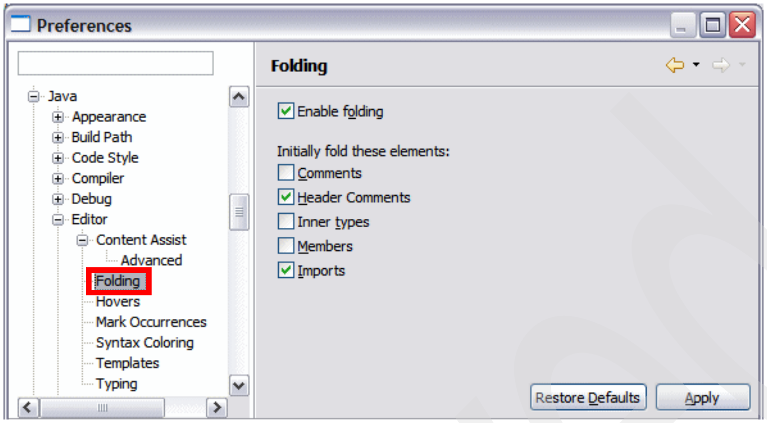
☐ Hide discouraged references

Figure 3-29 Java Editor: Content Assist preferences

Folding

When enabled, a user of the Workbench can collapse a method, comments, or class into a concise single line view.

To configure folding preferences, select **Windows** → **Preferences** → **Java** → **Editor** → **Folding** (Figure 3-30).



Folding

☑ Enable folding

Initially fold these elements:

☐ Comments
☑ Header Comments
☐ Inner types
☐ Members
☑ Imports

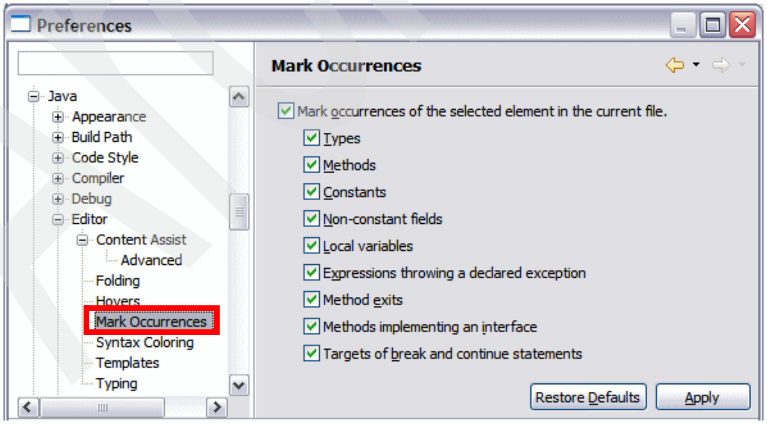
Figure 3-30   Java Editor: Folding preferences

Mark occurrences

When enabled, this highlights all occurrences of the types of entities described in the screen (Figure 3-31). In the Java editor, by selecting an attribute (for example), the editor displays in the far right-hand context bar all occurrences in the resource of that attribute. This can be navigated to by selecting the highlight bar in that context bar.

To configure mark occurrences preferences, select **Windows** → **Preferences** → **Java** → **Editor** → **Mark Occurrences** (Figure 3-31).

.



Mark Occurrences

☑ Mark occurrences of the selected element in the current file.

☑ Types
☑ Methods
☑ Constants
☑ Non-constant fields
☑ Local variables
☑ Expressions throwing a declared exception
☑ Method exits
☑ Methods implementing an interface
☑ Targets of break and continue statements

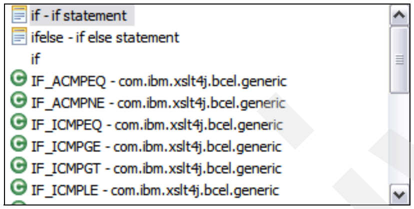
Figure 3-31   Java Editor: Mark Occurrences preferences

Templates

A template is a convenience that allows the programmer to quickly insert often reoccurring source code patterns. Application Developer has pre-defined templates and allows you to create new and edit existing templates.

The templates can be used by typing a part of the statement you want to add; and then by pressing Ctrl+Spacebar in the Java editor, a list of templates matching the key will appear in the presented list. Note that the list is filtered as you type, so typing the few first characters of a template name will reveal it.

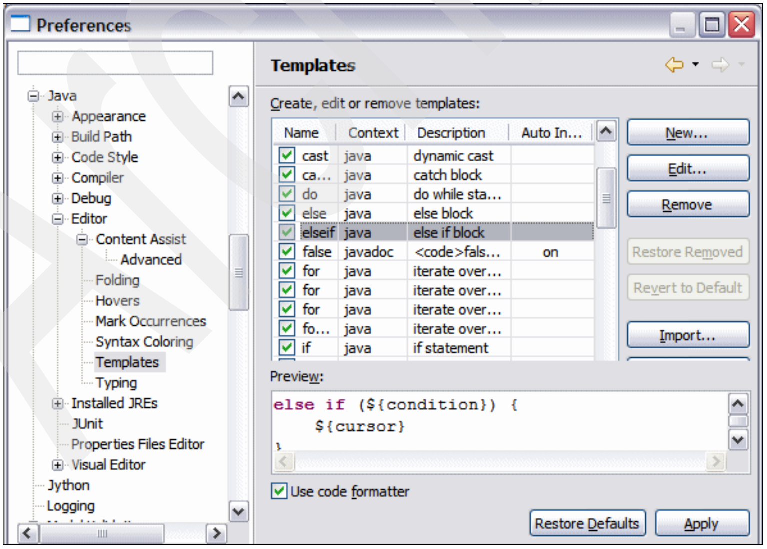
The symbol in front of each template (Figure 3-32), in the code assist list is colored yellow, so you can distinguish between a template and a Java statement entry.



```
if - if statement
ifelse - if else statement
if
IF_ACMPEQ - com.ibm.xslt4j.bcel.generic
IF_ACMPNE - com.ibm.xslt4j.bcel.generic
IF_ICMPEQ - com.ibm.xslt4j.bcel.generic
IF_ICMPGE - com.ibm.xslt4j.bcel.generic
IF_ICMPGT - com.ibm.xslt4j.bcel.generic
IF_ICMPLE - com.ibm.xslt4j.bcel.generic
```

Figure 3-32 Using templates for content assist

To configure templates, select **Windows** → **Preferences** → **Java** → **Editor** → **Templates** (Figure 3-33).



Create, edit or remove templates:

	Name	Context	Description	Auto In...
☑	cast	java	dynamic cast	
☑	ca...	java	catch block	
☑	do	java	do while sta...	
☑	else	java	else block	
☑	elseif	java	else if block	
☑	false	javadoc	<code>fals...	on
☑	for	java	iterate over...	
☑	for	java	iterate over...	
☑	for	java	iterate over...	
☑	fo...	java	iterate over...	
☑	if	java	if statement	

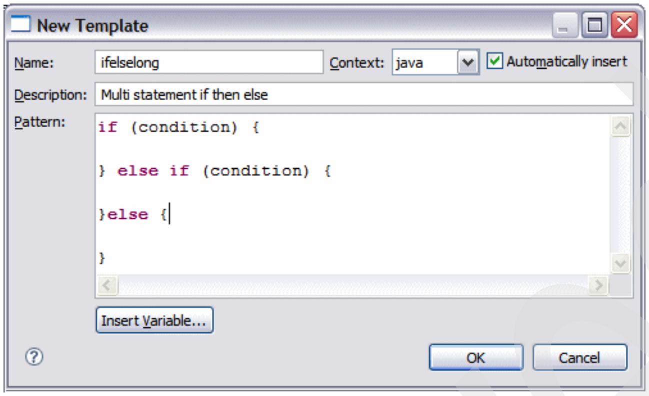
Preview:

```
else if (${condition}) {
    ${cursor}
}
```

☑ Use code formatter

Figure 3-33 Java Editor: Templates preferences

- Click **New** to add a new template and, for example, enter the information to create a new template for a multi-line if-then-else statement, and click **OK** (Figure 3-34).



Name: ifelselong
Context: java
☑ Automatically insert
Description: Multi statement if then else
Pattern:

```
if (condition) {

} else if (condition) {

}else {

}
```

Figure 3-34 Creating a new template

The template is now available for use in the Java editor.

There are also some predefined variables available that can be added in the template. These variables can be inserted by clicking **Insert Variable**. This brings up a list and a brief description of the variable.

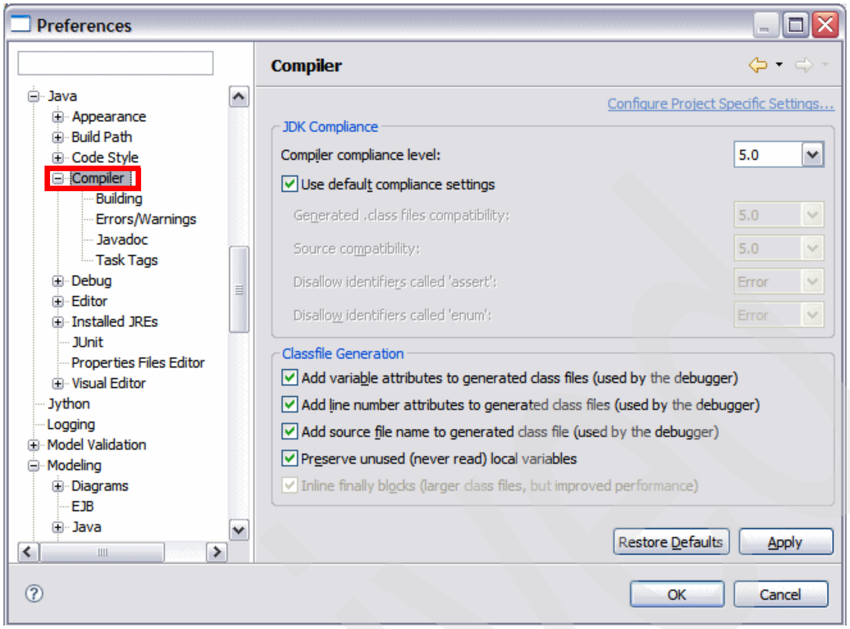
Templates that have been defined can be exported and later imported into Application Developer to ensure that a common environment can be set up among a team of developers.

Compiler options

Problems detected by the compiler are classified as either warnings or errors. The existence of a warning does not affect the execution of the program. The code executes as if it had been written correctly. Compile-time errors (as specified by the Java Language Specification) are always reported as errors by the Java compiler.

For some other types of problems you can, however, specify if you want the Java compiler to report them as warnings, errors, or to ignore them.

To configure the compiler options, select **Window** → **Preferences** → **Java** → **Compiler** (Figure 3-35).



Compiler

Configure Project Specific Settings...

JDK Compliance

Compiler compliance level: 5.0

☑ Use default compliance settings

Generated .class files compatibility: 5.0

Source compatibility: 5.0

Disallow identifiers called 'assert': Error

Disallow identifiers called 'enum': Error

Classfile Generation

☑ Add variable attributes to generated class files (used by the debugger)

☑ Add line number attributes to generated class files (used by the debugger)

☑ Add source file name to generated class file (used by the debugger)

☑ Preserve unused (never read) local variables

☑ Inline finally blocks (larger class files, but improved performance)

Figure 3-35 Java: Compiler preferences

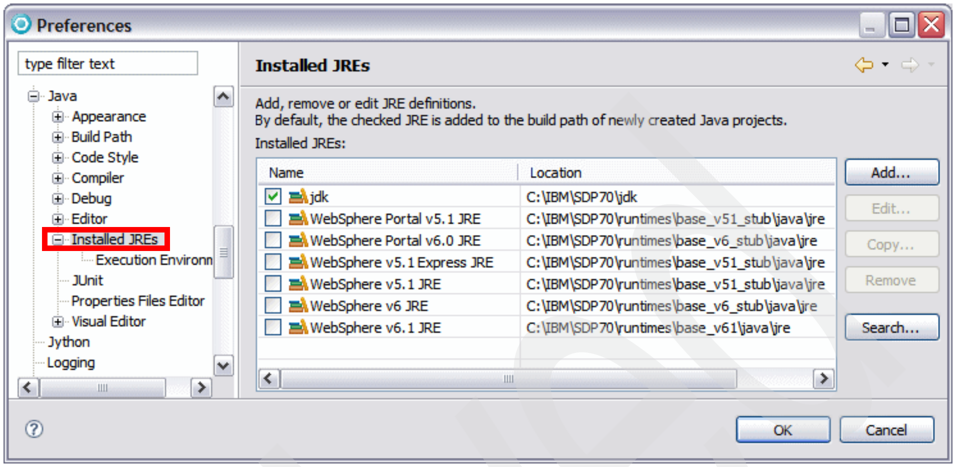
The compiler preferences page includes four sub-pages that allow you to set the appropriate behavior required to ensure that you obtain the required information from the compiler:

- Building: Indicates your preferences for the Building settings, such as build path problems, output folder, and so on.
- Errors/Warnings: Defines the level the errors and warnings in several categories such as code style, potential programming problems, unnecessary code, annotations, and so on.
- Javadoc: Provides configuration settings on how to deal with Javadoc problems that might arise and what to display as errors.
- Task Tags: Enables you to create, edit and remove Java task tags.

Installed JREs

Application Developer allows you to specify which Java Runtime Environment (JRE) should be used by the Java builder. By default, the standard Java VM that comes with the product is used; however, to ensure that your application is targeted for the correct platform, the same JRE or at least the same version of the JRE should be used to compile the code. If the application is targeted for a WebSphere Application Server V6.1, then the JRE should be set to use the JRE associated with this environment in Application Developer.

To configure the installed JREs, select **Windows** → **Preferences** → **Java** → **Installed JREs** (Figure 3-36).



Installed JREs

Add, remove or edit JRE definitions.
By default, the checked JRE is added to the build path of newly created Java projects.
Installed JREs:

Name	Location
☑ jdk	C:\IBM\SDP70\jdk
□ WebSphere Portal v5.1 JRE	C:\IBM\SDP70\runtimes\base_v51_stub\java\jre
□ WebSphere Portal v6.0 JRE	C:\IBM\SDP70\runtimes\base_v6_stub\java\jre
□ WebSphere v5.1 Express JRE	C:\IBM\SDP70\runtimes\base_v51_stub\java\jre
□ WebSphere v5.1 JRE	C:\IBM\SDP70\runtimes\base_v51_stub\java\jre
□ WebSphere v6 JRE	C:\IBM\SDP70\runtimes\base_v6_stub\java\jre
□ WebSphere v6.1 JRE	C:\IBM\SDP70\runtimes\base_v61\java\jre

Figure 3-36 Java: Installed JRE preferences

> **Note:** Changing the JRE used for running does not affect the way Java source is compiled. You can adjust the build path to compile against custom libraries.

By default, the JRE used to run the Workbench will be used to build and run Java programs. It appears selected in the list of installed JREs. If the target JRE that the application will run under is not in the list of JREs, then this can be installed on the machine and added onto the list. You can add, edit, or remove a JRE.

Let us assume that the application you are writing requires the latest JRE 1.6 located in the directory `C:\Program Files\Java\jre1.6\`. The procedure to add a new JRE is as follows:

- Click **Add**.
- In the Add JRE dialog, enter the following items:
  - JRE type: A drop-down box indicating whether a Standard VM or Standard 1.1.x VM. In most circumstances this will be set to **Standard VM**.
  - JRE name: Any name for the JRE to identify it.
  - JRE home directory: The location of the root directory of the install for the JRE: `C:\Program Files\Java\jre1.6\`
  - Default VM arguments: Arguments that are required to be passed to the JRE.

  - JRE system libraries: List of Jar files required for the JRE. Add Jar files under the `C:\Program Files\Java\jre1.6\lib` and `C:\Program Files\Java\jre1.6\lib\ext` directories.

- Click **OK** to add.
- Select the JRE in the list of installed JREs to set it as the default JRE, click **OK**, and rebuild all the projects in the workspace.

Summary

In this chapter we described how to configure the Workbench preferences in regard to logging, automatic builds, development capabilities, and Java development.

4

Perspectives, views, and editors

Rational Application Developer supports a role-based development model, which means that the development environment provides different tools, depending on the role of the user. It does this by providing several different perspectives that contain different editors and views necessary to work on tasks associated with each role.

This chapter starts with an introduction to the common structures and features applicable to all perspectives in Application Developer and then describes its help facility. Following this we provide a brief overview of the main features of each perspective available in IBM Rational Application Developer V7.0. Most of these perspectives are described in detail in the chapters within this book.

The chapter is organized into the following sections:

- Integrated development environment (IDE)
- Application Developer Help
- Available perspectives
- Summary

Integrated development environment (IDE)

An integrated development environment is a set of software development tools such as source editors, compilers and debuggers, that are accessible from a single user interface.

In Rational Application Developer, the IDE is called the **Workbench**. Rational Application Developer's Workbench provides customizable perspectives that support role-based development. It provides a common way for all members of a project team to create, manage, and navigate resources easily.

Views provide different ways of looking at the resource you are working on and editors allow you to create and modify the resource. Perspectives are a combination of views and editors that show various aspects of the project resource, and are organized by developer role or task. For example, a Java developer would work most often in the Java perspective, while a Web designer would work in the Web perspective.

Several default perspectives are provided in Rational Application Developer and team members also can customize them according to their current role and personal preference. More than one perspective can be opened at a time and users can switch perspectives while working with Application Developer. If you find that a particular perspective does not contain the views or editors you require, you can add them to the perspective and position them to suit your requirements.

Perspectives

Perspectives provide a convenient grouping of views and editors that match a particular way of using Rational Application Developer. A different perspective can be used to work on a given workspace depending on the role of the developer or the task that has to be done.

For each perspective, Application Developer defines an initial set and layout of views and editors for performing a particular set of development activities. For example, the J2EE perspective contains views and editors applicable for EJB development. The layout and the preferences in each perspective can be changed and saved as a customized perspective and used again later. This is described in “Organizing and customizing perspectives” on page 122.

Views

Views provide different presentations of resources or ways of navigating through the information in your workspace. For example, the **Project Explorer** view displays projects and other resources that you are working in a folder type hierarchy, while the **Links** view (in the Web perspective) shows just the Web pages in a given project and the URL links between them. Rational Application Developer provides synchronization between views and editors, so that changing the focus or a value in an editor or view can automatically update another. In addition, some views display information obtained from other software products, such as database systems or software configuration management (SCM) systems.

A view can appear by itself or stacked with other views in a tabbed notebook arrangement. To quickly move between views in a given perspective, you can select **Ctrl-F7** (and hold down Ctrl), which will show all the open views and let the user move quickly to the desired view. Press **F7** until the required view is selected, then release to move to that view.

Editors

When you open a file, Rational Application Developer automatically opens the editor that is associated with that file type. For example, the Page Designer is opened for `.html`, `.htm`, and `.jsp` files, while the Java editor is opened for `.java` and `.jpage` files.

Editors that have been associated with specific file types will open in the editor area of the Workbench. By default, editors are stacked in a notebook arrangement inside the editor area. If there is no associated editor for a resource, Rational Application Developer will open the file in the default editor, which is a text editor. It is also possible to open a resource in another editor by using the **Open With** option from the context menu.

To quickly move between editors open on the workspace, you can select **Ctrl-F6** (and hold down Ctrl) which will show all the open editors and let the user move quickly to the desired one. Press **F6** until the required editor is selected, then release.

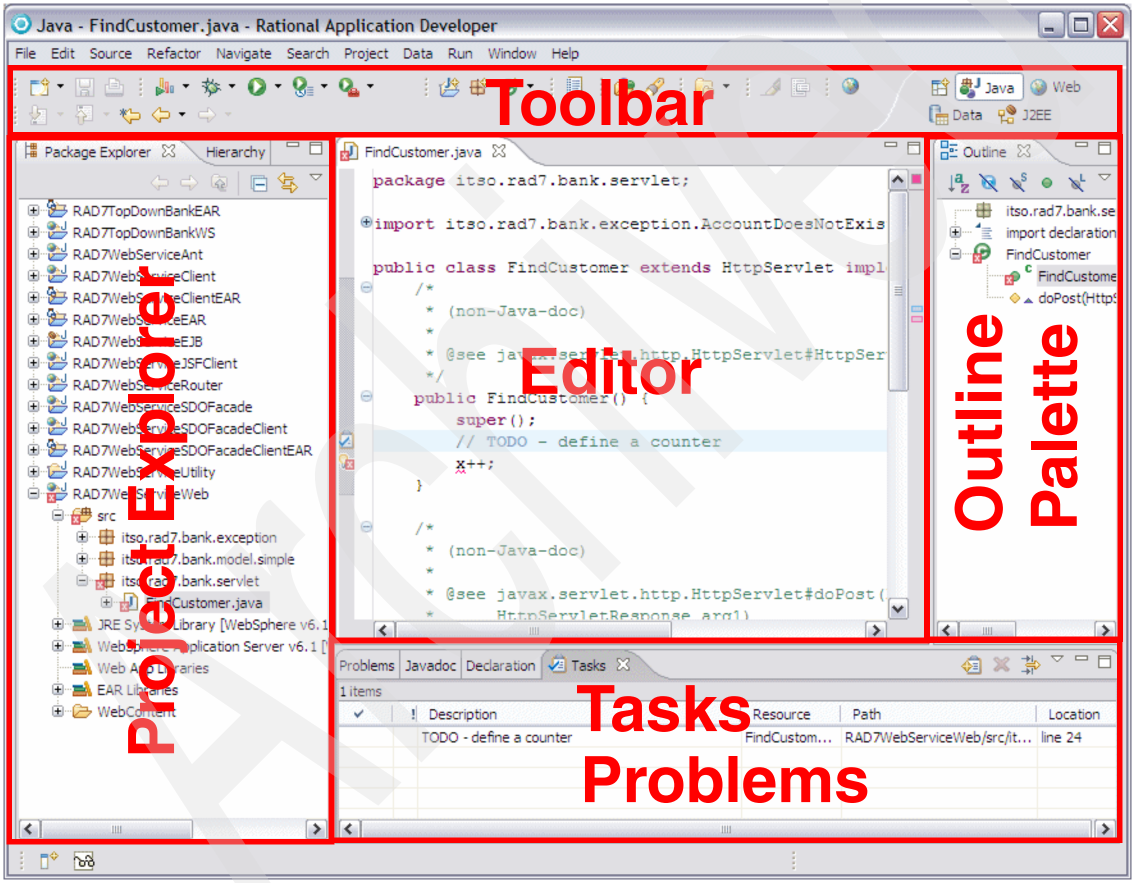
The following icons often appear in the Toolbar of a perspective to speed up navigation between editors:

- **Next and Previous cursor location** (⇦ ▾ and ⇨ ▾)—Moves the focus around recent cursor positions.
- **Last Edit Location** (⇦)—Returns the focus to the last location where something was changed.

- **Next and Previous Annotation** (and)—Depending on the options selected in the associated drop-down menu, this moves the cursor to the next and previous items in the associated list. For example, if errors are chosen, these buttons will move to the next or previous source code error in the workspace.

Perspective layout

Many of Rational Application Developer's perspectives use a similar layout. Figure 4-1 shows the general layout that is used for most default perspectives.



```
package itso.rad7.bank.servlet;

import itso.rad7.bank.exception.AccountDoesNotExis

public class FindCustomer extends HttpServlet impl
    /*
     * (non-Java-doc)
     *
     * @see javax.servlet.http.HttpServlet#HttpSer
     */
    public FindCustomer() {
        super();
        // TODO - define a counter
        x++;
    }

    /*
     * (non-Java-doc)
     *
     * @see javax.servlet.http.HttpServlet#doPost(
     *      HttpServletResponse arg1)
```

1 items

✓	!	Description	Resource	Path	Location
		TODO - define a counter	FindCustom...	RAD7WebServiceWeb/src/it...	line 24

Figure 4-1 Perspective layout

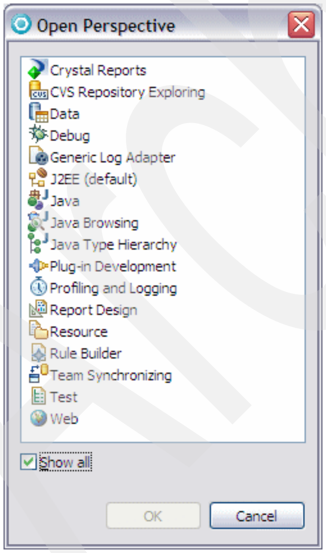
On the left side are views for navigating through the workspace. In the middle of the Workbench is larger pane, which is where the main editors are shown. The right pane usually contains Outline or Palette views of the file in the main editor. In some perspectives the editor pane is larger and the outline view is located at the bottom left corner of the perspective. At the bottom right is a tabbed series of views including the Tasks view, the Problems view, and the Properties view. This is where smaller miscellaneous views not associated with navigation, editing, or outline information are shown.

Switching perspectives

There are two ways to open another perspective:

- Click the **Open a perspective** icon (⊞) in the top right corner of the Workbench working area and select the appropriate perspective from the list.
- Select **Window** → **Open Perspective** and select one from the drop-down list shown.

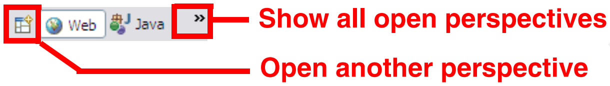
In both cases there is also an **Other** option, which when selected displays the **Open Perspective** dialog that shows the complete list of perspectives (see Figure 4-2). Here the user can select the required perspective and click **OK**.



☑ Show all

Figure 4-2 Open Perspective dialog

In all perspectives, a group of buttons appears in the top right corner of the Workbench (an area known as the shortcut bar). Each button corresponds to an open perspective and the >> icon will show a list if there are too many (see Figure 4-2). Clicking on one of these will move to the associated perspective.



Show all open perspectives

Open another perspective

Figure 4-3 Buttons to switch between perspectives

Tips:

- The name of the perspective is shown in the window title area along with the name of the file open in the editor, which is currently at the front.
- To close a perspective, right-click the perspective's button on the shortcut bar (top right) and select **Close**.
- To display only the icons for the perspectives, right-click somewhere in the shortcut bar and clear the **Show Text** option.
- Each perspective requires memory, so it is good practice to close perspectives that are not used to improve performance.

Specifying the default perspective

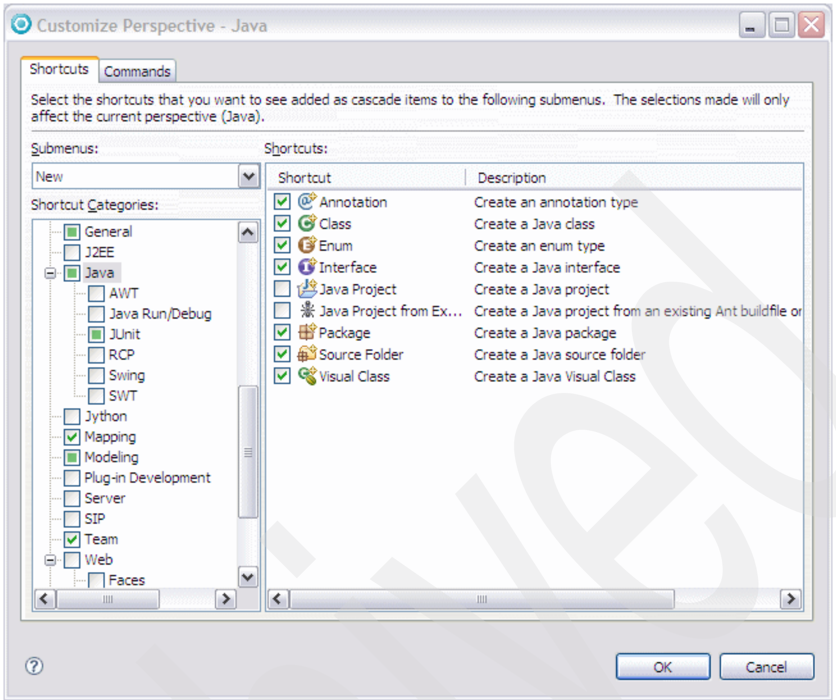
The J2EE perspective is Rational Application Developer's default perspective, but this can be changed using the Preferences dialog:

- From the Workbench, select **Window** → **Preferences**.
- Expand **General** and select **Perspectives**. Note that the J2EE perspective has **default** after it.
- Select the perspective that you want to define as the default, and click **Make Default**.
- Click **OK**.

Organizing and customizing perspectives

Rational Application Developer allows you to open, customize, reset, save, and close perspectives. These actions can be found in the **Window** menu.

To customize the commands and shortcuts available within a perspective, select **Window** → **Customize Perspective**. The Customize Perspective dialog opens (Figure 4-4).



Select the shortcuts that you want to see added as cascade items to the following submenus. The selections made will only affect the current perspective (Java).

Submenus: New

Shortcut Categories:

- ▪ General
- □ J2EE
- ▪ Java
  - □ AWT
  - □ Java Run/Debug
  - ▪ JUnit
  - □ RCP
  - □ Swing
  - □ SWT
- □ Jython
- ☑ Mapping
- ▪ Modeling
- □ Plug-in Development
- □ Server
- □ SIP
- ☑ Team
- □ Web
  - □ Faces

Shortcuts:

	Shortcut	Description
☑	Annotation	Create an annotation type
☑	Class	Create a Java class
☑	Enum	Create an enum type
☑	Interface	Create a Java interface
□	Java Project	Create a Java project
□	Java Project from Ex...	Create a Java project from an existing Ant buildfile or
☑	Package	Create a Java package
☑	Source Folder	Create a Java source folder
☑	Visual Class	Create a Java Visual Class

Figure 4-4 Customize Perspective dialog

The **Shortcuts** tab provides the facility to specify which options are shown on the **New**, **Open Perspective**, and **Show View** menu options within the current perspective. Select the menu you want to customize from the **Submenus** drop-down window and check the boxes for whichever options you want to appear. Note that items you do not select are still accessible by clicking the **Other** menu option, which is always present for these options.

The **Commands** tab of the dialog allows you to select command groups that will be added to the menu bar or tool bar for Rational Application Developer in this perspective.

In addition to customizing the Command and options available as shortcuts, it is also possible to reposition any of the views and editors and to add or remove other editors as desired. The following features are available to do this:

- **Adding and Removing Views**—To add a view to the perspective, select **Window** → **Show View** and select the view you would like to add to the currently open perspective. To remove a view, simply close it from its title menu.
- **Move**—You can move a view to another pane by using drag and drop. To do this, select its title bar and drag the view to another place on the workspace.

While you drag the view, the mouse cursor changes into a drop cursor, which is an arrow or other icon indicting where the view will appear when it is released. In each case, the area that is filled with the dragged view is highlighted with a rectangular outline.

The drop cursor will look like one of the following:

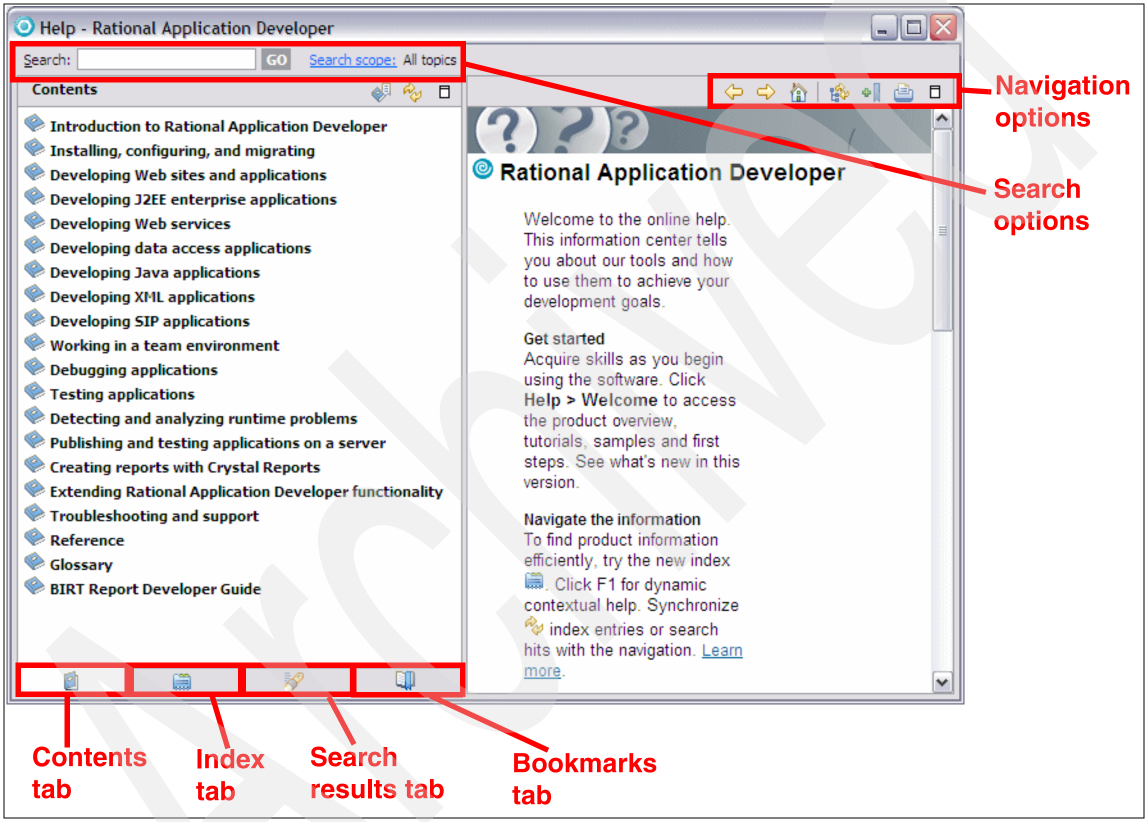
⬇	The view will dock below the view under the cursor.
⬅	The view will dock to the left of the view under the cursor.
➡	The view will dock to the right of the view under the cursor.
⬆	The view will dock above the view under the cursor.
⧉	The view will appear as a tab in the same pane as the view under the cursor.
⇤	The view will dock in the status bar (at the bottom of the Rational Application Developer window) and become a **Fast View** (see below). This icon will appear when a view is dragged to the bottom left corner of a work-space.
⊞	The view becomes a separate child window of the main Rational Application Developer window. This icon will appear when you drag a view to an area outside the work-space.

- **Fast View**—A Fast View appears as a button in the status bar of Rational Application Developer in the bottom left corner of the workspace. Clicking the button will toggle whether or not the view is displayed on top of the other views in the perspective.
- **Maximize and minimize a view**—To maximize a view to fill the whole working area of the Workbench, you can double-click the title bar of the view, press Ctrl+M, or click the Maximize icon (□)in the view's toolbar. To restore the view double-click the title bar, select the restore button (⧉) or press Ctrl+M again. The Minimize button in the toolbar of a view minimizes the tab group so that only the tabs are visible; click the **Restore** button or one of the view tabs to restore the tab group.
- **Save**—Once you have configured the perspective to your preferences, you can save it as your own perspective by selecting **Window** → **Save Perspective As** and type a new name. The new perspective now appears as an option on the **Open Perspective** window.
- **Restore**—To restore the currently open perspective to its original layout, select **Window** → **Reset Perspective**.

Application Developer Help

The Rational Application Developer Help system provides access to the documentation and lets users browse and print help content. It also has a full-text search engine and context-sensitive help.

The Help contents can be displayed in a separate window, by selecting **Help** → **Help Contents** from the menu bar (see Figure 4-5).



- Introduction to Rational Application Developer
- Installing, configuring, and migrating
- Developing Web sites and applications
- Developing J2EE enterprise applications
- Developing Web services
- Developing data access applications
- Developing Java applications
- Developing XML applications
- Developing SIP applications
- Working in a team environment
- Debugging applications
- Testing applications
- Detecting and analyzing runtime problems
- Publishing and testing applications on a server
- Creating reports with Crystal Reports
- Extending Rational Application Developer functionality
- Troubleshooting and support
- Reference
- Glossary
- BIRT Report Developer Guide

Rational Application Developer

Welcome to the online help. This information center tells you about our tools and how to use them to achieve your development goals.

Get started
Acquire skills as you begin using the software. Click **Help > Welcome** to access the product overview, tutorials, samples and first steps. See what's new in this version.

Navigate the information
To find product information efficiently, try the new index. Click F1 for dynamic contextual help. Synchronize index entries or search hits with the navigation. Learn more.

Figure 4-5 Help window

In the Help window you see the available books in the left pane and the content in the right pane. When you select a book (📘) in the left pane, the appropriate table of contents opens up and you can select a topic (📖) within the book. When a page (📄) is selected, the page content is displayed in the right pane.

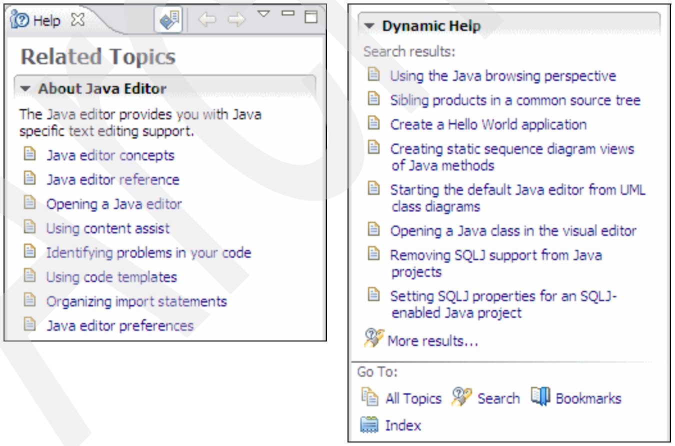
You can navigate through the help documents by clicking **Go Back** ⇦ and **Go Forward** (⇨) in the toolbar for the right pane.

The following buttons are also available in the toolbar:

- **Show in Table of Contents** ()—Synchronizes the navigation frame with the current topic, which is helpful if you have followed several links to related topics in several files, and want to see where the current topic fits into the navigation path.
- **Bookmark Document** ()—Adds a bookmark to the Bookmarks view, which is one of the tabs on the left pane.
- **Print Page** (🖨)—Provides the option to print the page current displayed in the right-hand window.
- **Maximize** (□)—Maximizes the right hand pane to fill the whole help window. Note that when this pane is maximized, the icon changes to **Restore**(🗗) which allows the user to return the page back to normal.

Also, the left hand pane of the help window can be tabbed between the **Contents, Index, Search Results**, and **Bookmarks** views, which provide different methods of accessing information the help contents.

Rational Application Developer's help system contains a lot of useful information about the tools and technologies available from the workbench, and is loosely arranged around different types of development possible (for example Java, Web, XML and many others). While performing any task within Application Developer, you can press **F1** at any time and the **Help** view displays the context help showing a list of relevant topics for the current view or editor. For example, Figure 4-6 shows the context help when editing normal Java code.



Related Topics

About Java Editor

The Java editor provides you with Java specific text editing support.

- Java editor concepts
- Java editor reference
- Opening a Java editor
- Using content assist
- Identifying problems in your code
- Using code templates
- Organizing import statements
- Java editor preferences

Dynamic Help

Search results:

- Using the Java browsing perspective
- Sibling products in a common source tree
- Create a Hello World application
- Creating static sequence diagram views of Java methods
- Starting the default Java editor from UML class diagrams
- Opening a Java class in the visual editor
- Removing SQLJ support from Java projects
- Setting SQLJ properties for an SQLJ-enabled Java project

More results...

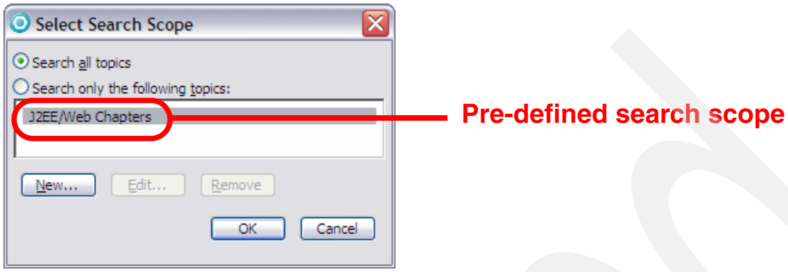
Go To:

All Topics Search Bookmarks Index

Figure 4-6 Context sensitive help when Java editing

Also, at any time it is possible to display the Help contents as a view within the workspace, by selecting **Window** → **Show View** → **Other** → **Help** → **Help**.

The Search field allows you to do a search which will be through all the Help contents by default. Clicking the **Search Scope** link opens a dialog box where you can select a scope for your search (Figure 4-7). This will show any previously defined search scopes and give the user the opportunity to create a new one.


Select Search Scope

● Search all topics

○ Search only the following topics:

J2EE/Web Chapters

Pre-defined search scope

Figure 4-7 Select Search Scope dialog for help

Once the search list has been saved, it can be selected as a search scope from the main search page.

Clicking **Go** performs the search across the selected scope and display the results in the **Search Results** view, and from there the user can click on links to pages within the Help facility.

Available perspectives

In this section all the perspectives that are available in Rational Application Developer are briefly described. The perspectives are covered in alphabetical order which is how they appear in the **Select Perspective** dialog.

Rational Application Developer allows a developer to disable or enable capabilities to simplify the interface or make it more capable for specific types of development work. This is described in “Capabilities” on page 86. For this section, all capabilities have been enabled in the product. If this is not done, certain perspectives, associated with specific capabilities, are not available.

IBM Rational Application Developer V7.0 includes the following perspectives:

- Crystal Reports perspective
- CVS Repository Exploring perspective
- Data perspective
- Debug perspective
- Generic Log Adapter perspective
- J2EE perspective
- Java perspective

- Java Browsing perspective
- Java Type Hierarchy perspective
- Plug-in Development perspective
- Profiling and Logging perspective
- Report design perspective
- Resource perspective
- Rule Builder perspective
- Team Synchronizing perspective
- Test perspective
- Web perspective

Crystal Reports perspective

The Crystal Reports features are an optional extra that can be chosen when installing Application Developer. If this is installed, then the Crystal Reports perspective provides the tools to work with a database and produce simple or complex reports (using Crystal reports) which can be published to the Web or incorporated into an application. The main editor uses a Layout page which allows report elements to be placed on a report template from a Palette view and a Preview page to show what a report will look like. These work with the Database Explorer View and the Field Explorer View to position various fields from results of queries and database tables onto the report.

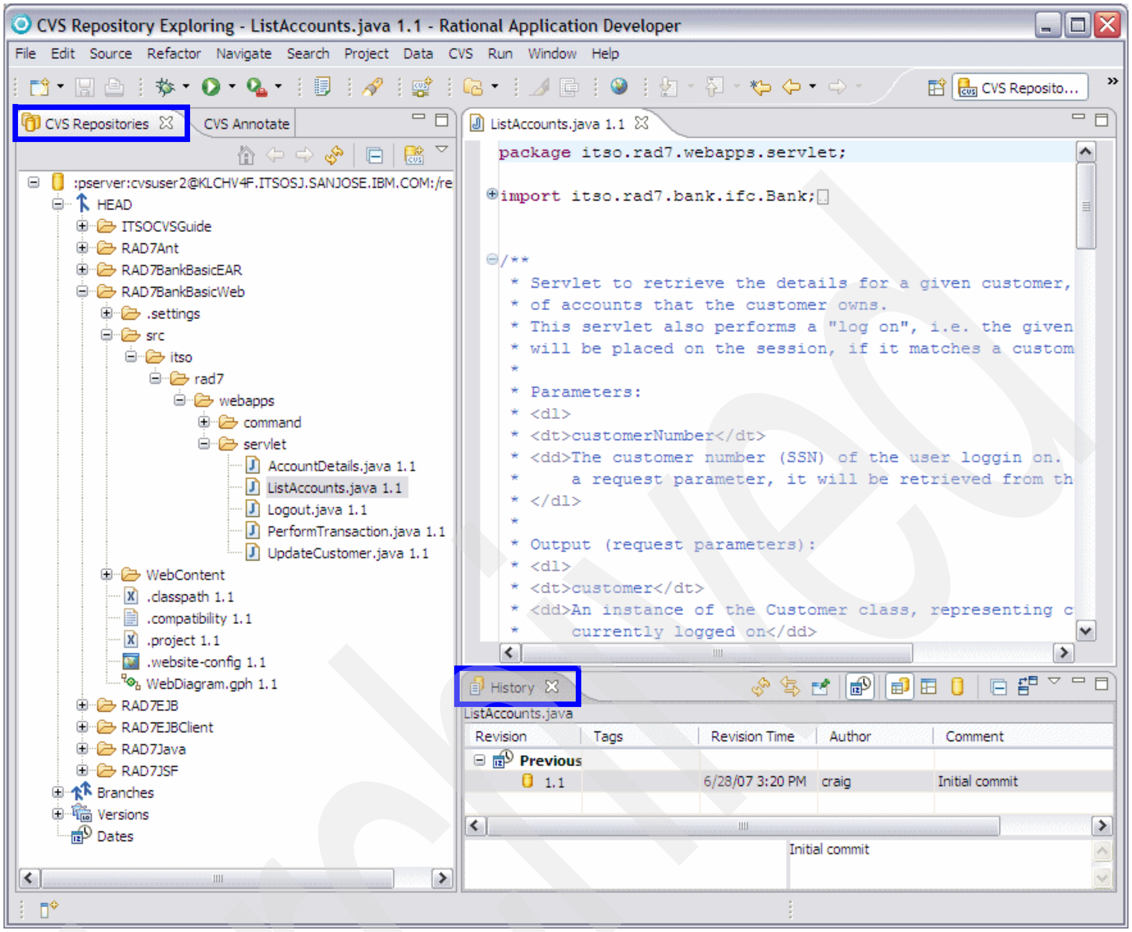
Application Developer Help has a large section on the details of using this perspective.

CVS Repository Exploring perspective

The CVS Repository Exploring perspective (Figure 4-8) lets you connect to Concurrent Versions System (CVS) repositories and to inspect the revision history of resources in those repositories.

- **CVS Repositories view**—Shows the CVS repository locations that have been added to the Workbench. Expanding a location reveals the main trunk (HEAD), project versions, and branches in that repository. You can further expand the project versions and branches to reveal the folders and files contained within them.

  The context menu for this view also allows you to specify new repository locations. The CVS Repositories view can be used to check out resources from the repository to the Workbench, configure the branches and versions shown by the view, view a resource’s history and compare resource versions.



```
package itso.rad7.webapps.servlet;

import itso.rad7.bank.ifc.Bank;

/**
 * Servlet to retrieve the details for a given customer,
 * of accounts that the customer owns.
 * This servlet also performs a "log on", i.e. the given
 * will be placed on the session, if it matches a custom
 *
 * Parameters:
 * <dl>
 * <dt>customerNumber</dt>
 * <dd>The customer number (SSN) of the user loggin on.
 *     a request parameter, it will be retrieved from th
 * </dl>
 *
 * Output (request parameters):
 * <dl>
 * <dt>customer</dt>
 * <dd>An instance of the Customer class, representing c
 *     currently logged on</dd>
```

Revision	Tags	Revision Time	Author	Comment
Previous				
1.1		6/28/07 3:20 PM	craig	Initial commit

Initial commit

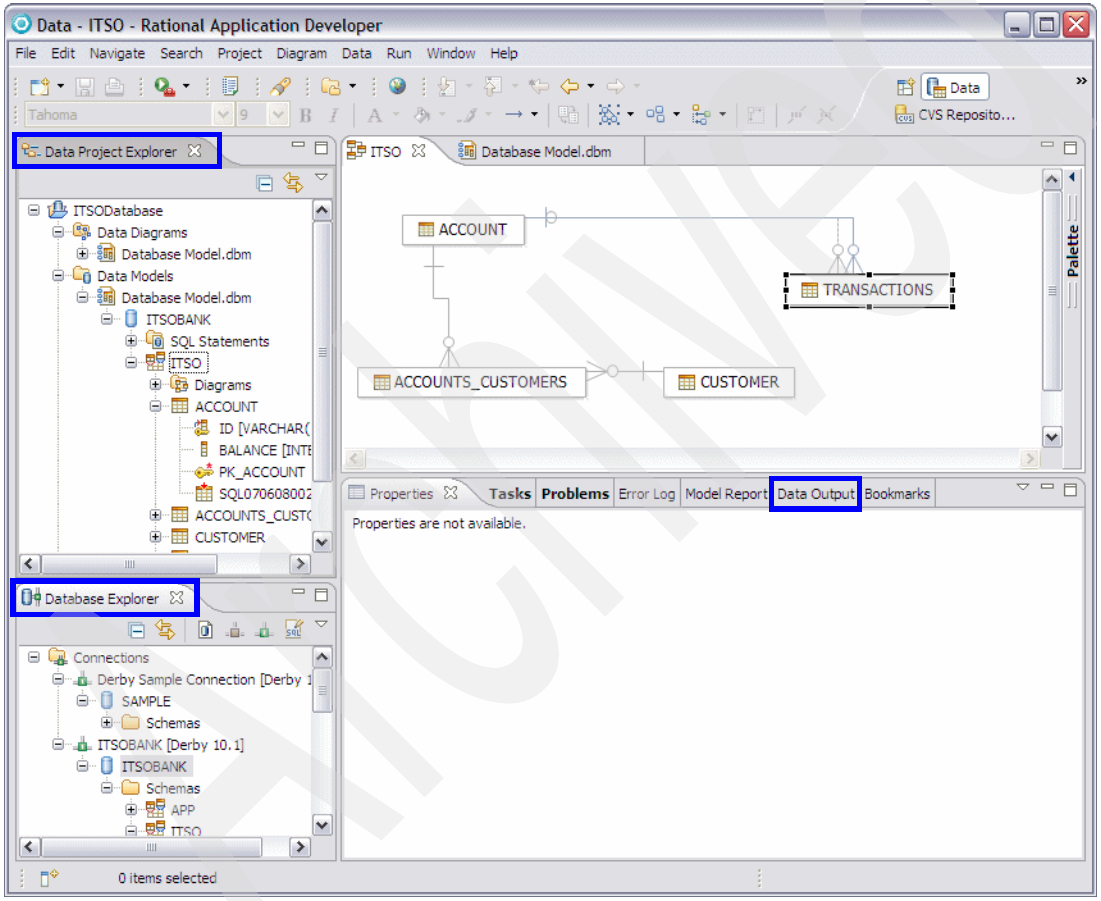
Figure 4-8 CVS Repository Exploring perspective

- **Editor**—Files that exist in the repositories can be viewed by double-clicking them in a branch or version. This opens the version of the file specified in the editor pane. Note that the contents of the editor are read-only.
- **CVS Resource History view**—Displays a detailed history of each file providing a list of all the revisions of it in the repository. From this view you can also compare two revisions or open an editor on a revision.
- **CVS Annotation view**—To show this view, select a resource in the CVS Repositories view, right-click and select **Show Annotation**. The CVS Annotate view will come to the front and will display a summary of all the changes made to the resource since it came under the control of the CVS server. The CVS Annotate view will link with the main editor, showing which CVS revisions apply to which source code lines.

More details about using the CVS Repository Exploring perspective, and other aspects of CVS functionality in Rational Application Developer, can be found in Chapter 27, “CVS integration” on page 1213.

Data perspective

The Data perspective (Figure 4-9) lets you access a set of relational database tools, where you can create and manipulate the database definitions for your projects.



Properties are not available.

Figure 4-9 Data perspective

The important views are as follows:

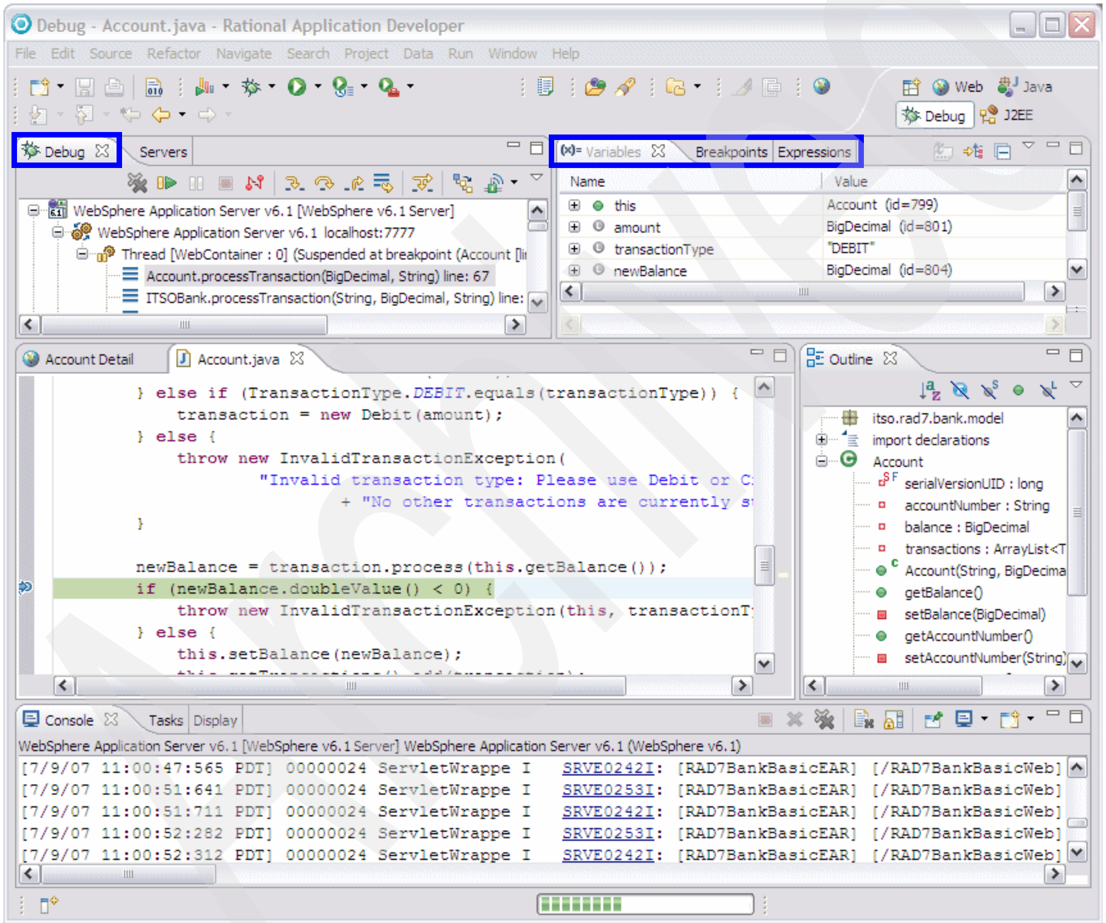
- **Data Project Explorer**—The main navigator view in the Data perspective showing only the data projects in the workspace. This view lets you work directly with data definitions and define relational data objects. It can hold local copies of existing data definitions imported from the DB Servers view, designs created by running DDL scripts, or new designs that you have created directly in the Workbench.
- **Database Explorer view**—Using this view, you can work with database connections to view the design of existing databases, and import the designs to another folder in the Data Project Explorer view, where you can extend or modify them.
- **Tasks view**—The Tasks view displays system-generated errors, warnings, or information associated with a resource, typically produced by builders. Tasks can also be added manually and optionally associated with a resource in the Workbench.
- **Navigator view**—The Navigator view provides a hierarchical view of all the resources in the Workbench. By using this view you can open files for editing or select resources for operations such as exporting. The Navigator view is essentially a file system view, showing the contents of the workspace and the directory structures used by any projects that have been created outside the workspace.
- **Console view**—The Console view shows the output of a process and allows you to provide keyboard input to a process. The console shows three different kinds of text, each in a different color: Standard output, standard error, and standard input.
- **Data Output view**—The Data Output view shows the messages, parameters, and results that are related to the database objects that you work with, such as SQL statements, stored procedures, and user-defined functions.
- **SQL Builder/Editor**—This view shows specialized wizards for creating and editing of SQL statements.
- **Data Diagram Editor**—This view shows an Entity Relationship diagram of the selected database.

More details about using the Data perspective can be found in Chapter 9, “Develop database applications” on page 355.

Debug perspective

By default, the Debug perspective (Figure 4-10) contains five panes with the following views:

- **Top left**—Shows Debug and Servers views
- **Top right**—Shows Breakpoints, Variables, and Expressions views.
- **Middle left**—Shows the editor for the resource being debugged.
- **Middle right**—Shows the Outline view of the resource being debugged.
- **Bottom**—Shows the Console and the Tasks view


Debug - Account.java - Rational Application Developer
File Edit Source Refactor Navigate Search Project Data Run Window Help
Web
Java
Debug
J2EE
Debug
Servers
(x)= Variables
Breakpoints
Expressions
Account Detail
Account.java
Outline
Console
Tasks
Display

WebSphere Application Server v6.1 [WebSphere v6.1 Server]
WebSphere Application Server v6.1 localhost:7777
Thread [WebContainer : 0] (Suspended at breakpoint (Account [li
Account.processTransaction(BigDecimal, String) line: 67
ITSOBank.processTransaction(String, BigDecimal, String) line:

Name	Value
this	Account (id=799)
amount	BigDecimal (id=801)
transactionType	"DEBIT"
newBalance	BigDecimal (id=804)

```
} else if (TransactionType.DEBIT.equals(transactionType)) {
   transaction = new Debit(amount);
} else {
   throw new InvalidTransactionException(
            "Invalid transaction type: Please use Debit or C
                  + "No other transactions are currently s
}

newBalance = transaction.process(this.getBalance());
if (newBalance.doubleValue() < 0) {
   throw new InvalidTransactionException(this, transactionT
} else {
   this.setBalance(newBalance);
```

itso.rad7.bank.model
import declarations
Account
serialVersionUID : long
accountNumber : String
balance : BigDecimal
transactions : ArrayList<T
Account(String, BigDecima
getBalance()
setBalance(BigDecimal)
getAccountNumber()
setAccountNumber(String)

WebSphere Application Server v6.1 [WebSphere v6.1 Server] WebSphere Application Server v6.1 (WebSphere v6.1)

```
[7/9/07 11:00:47:565 PDT] 00000024 ServletWrappe I   SRVE0242I: [RAD7BankBasicEAR] [/RAD7BankBasicWeb]
[7/9/07 11:00:51:641 PDT] 00000024 ServletWrappe I   SRVE0253I: [RAD7BankBasicEAR] [/RAD7BankBasicWeb]
[7/9/07 11:00:51:711 PDT] 00000024 ServletWrappe I   SRVE0242I: [RAD7BankBasicEAR] [/RAD7BankBasicWeb]
[7/9/07 11:00:52:282 PDT] 00000024 ServletWrappe I   SRVE0253I: [RAD7BankBasicEAR] [/RAD7BankBasicWeb]
[7/9/07 11:00:52:312 PDT] 00000024 ServletWrappe I   SRVE0242I: [RAD7BankBasicEAR] [/RAD7BankBasicWeb]
```

Figure 4-10 Debug perspective

The important views while debugging are as follows:

- **Debug view**—The Debug view displays the stack frame for the suspended threads for each target you are debugging. Each thread in your program appears as a node in the tree. If the thread is suspended, its stack frames are shown as child elements.

  If the resource containing a selected thread is not open and/or active, the file opens in the editor and becomes active, focusing on the point in the source where the thread is currently positioned.

  The Debug view contains a number of command buttons which enable users to perform actions such as start, terminate, and step-by-step debug actions.
- **Variables view**—The Variables view displays information about the variables in the currently selected stack frame.
- **Breakpoints view**—The Breakpoints view lists all the breakpoints you have set in the Workbench projects. You can double-click a breakpoint to display its location in the editor. In this view, you can also enable or disable breakpoints, delete them, change their properties, or add new ones. This view also lists Java exception breakpoints, which suspend execution at the point where the exception is thrown.
- **Servers view**—The Servers view lists all the defined servers and their status. Right-clicking a server displays the server context menu, which allows the server to be started, stopped, and to republish the current applications.
- **Outline view**—The Outline view shows the elements (imports, class, fields, and methods) that exist in the source file in the front editor. Clicking an item in the outline will position you in the editor view at the line where that structure element is defined.
- **Problems view**—This view shows all errors, warnings, and information messages relating to resources in the workspace. The items listed here can be used to navigate to the line of code containing the error, warning, or information point.

The Console and Tasks views have already been discussed in earlier sections of this chapter.

More information about the Debug perspective can be found in Chapter 22, “Debug local and remote applications” on page 1041.

Generic Log Adapter perspective

This perspective allows developers to create applications to convert application logs into the Common Base Event (CBE) format. This is a defined format for logging system events in a standardized format, so that problems from different applications can be caught and managed centrally. Because most applications do not log their information in Common Base Event format, this perspective provides tools to create and debug adapters from non-standard application log formats to Common Base Events.

For more information about the Generic Log Adapter perspective, refer to the Eclipse documentation at:

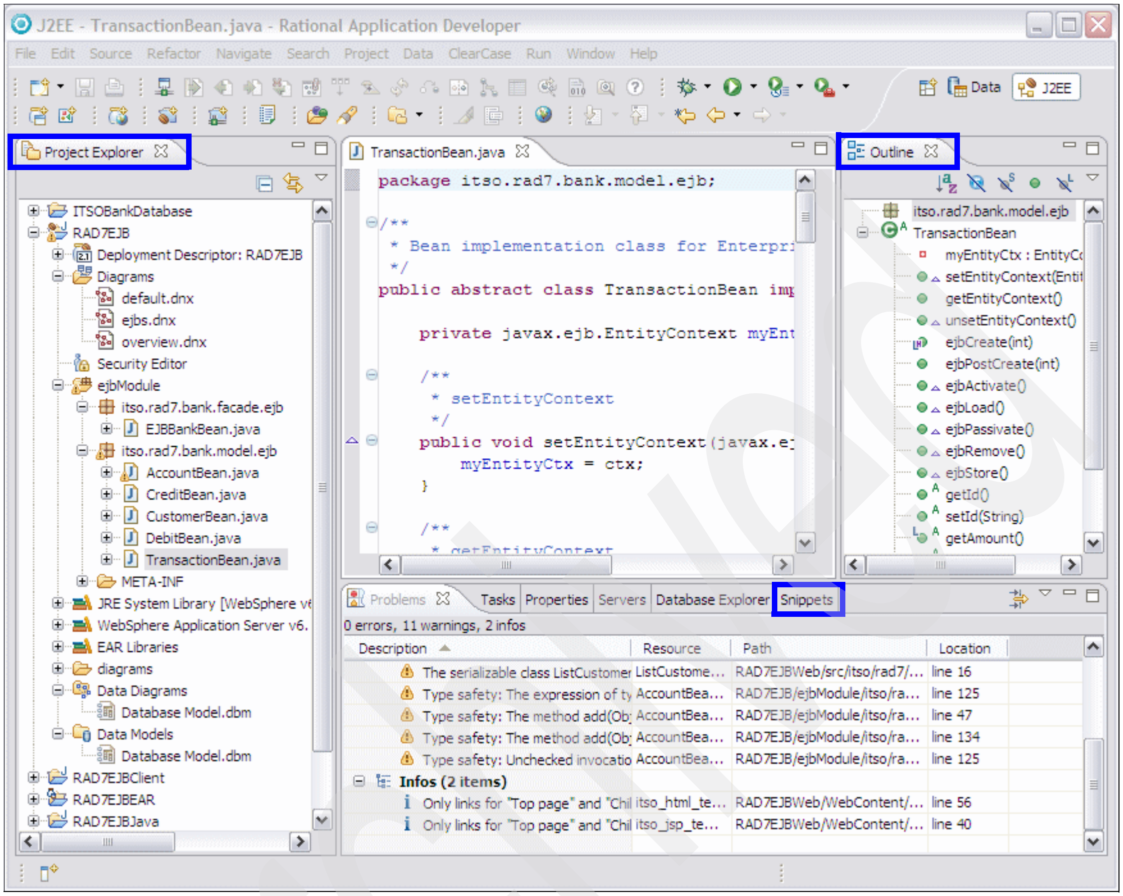
```
http://www.eclipse.org/tptp/home/documents/tutorials/gla/getting_started_gl
a.html
```

J2EE perspective

The p perspective (Figure 4-11) is used for working with Enterprise Application projects and EJB projects. It contains the following views typically used when developing J2EE applications:

- **Project Explorer view**—This view shows information about your J2EE and other projects in the workspace. It is similar to the Navigator view in that it shows the files and folders that make up a project but includes extra icons specific for a project type such as J2EE deployment descriptors and the Security Editor (both for J2EE projects). For Web projects the Project Explorer view includes icons for the Web deployment descriptor, the Web Navigation view, the Web Diagram view and the Security Editor.
- **Snippets view**—The Snippets view lets you catalog and organize reusable programming objects, such as HTML tagging, JavaScript, and JSP code, along with files and custom JSP tags. The view can be extended based on additional objects that you define and include. The available snippets are arranged in *drawers*, such as JSP or EJB, and the drawers can be customized by right-clicking a drawer and selecting **Customize**.
- **Properties view**—This view provides a tabular view of the properties and associated values of objects in files you have open in an editor. The format of this view differs depending on what is selected in the editor, and by default it shows the file properties (last modification date, file path and so on).

The Outline, Servers, Problems, Tasks, and Database Explorer views are also relevant to the Web perspective and have already been discussed in earlier sections of this chapter.



```
package itso.rad7.bank.model.ejb;

/**
 * Bean implementation class for Enterpri
 */
public abstract class TransactionBean imp

    private javax.ejb.EntityContext myEnt

    /**
     * setEntityContext
     */
    public void setEntityContext(javax.ej
        myEntityCtx = ctx;
    }

    /**
     * getEntityContext
```

0 errors, 11 warnings, 2 infos

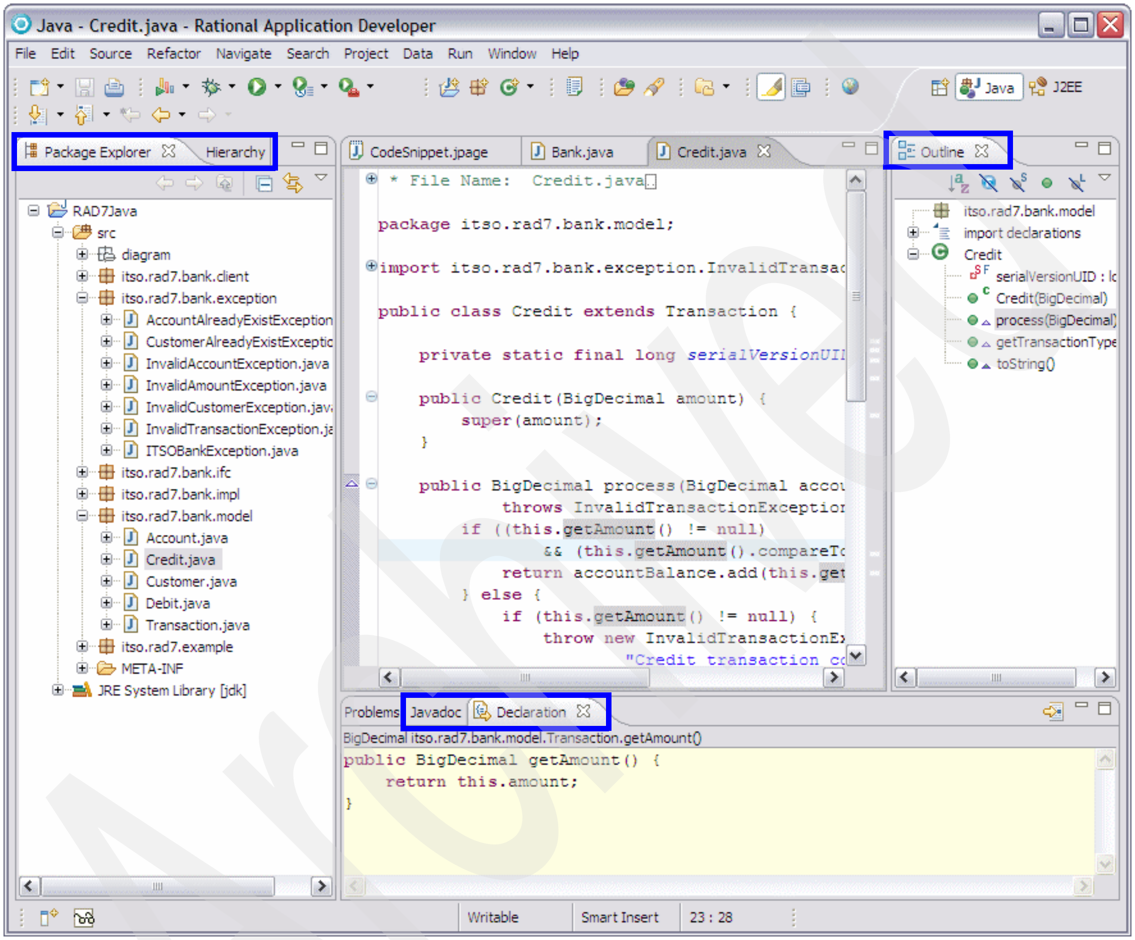
Description	Resource	Path	Location
⚠ The serializable class ListCustomer	ListCustome...	RAD7EJBWeb/src/itso/rad7/...	line 16
⚠ Type safety: The expression of ty	AccountBea...	RAD7EJB/ejbModule/itso/ra...	line 125
⚠ Type safety: The method add(Ob;	AccountBea...	RAD7EJB/ejbModule/itso/ra...	line 47
⚠ Type safety: The method add(Ob;	AccountBea...	RAD7EJB/ejbModule/itso/ra...	line 134
⚠ Type safety: Unchecked invocatio	AccountBea...	RAD7EJB/ejbModule/itso/ra...	line 125
Infos (2 items)			
ℹ Only links for "Top page" and "Chil	itso_html_te...	RAD7EJBWeb/WebContent/...	line 56
ℹ Only links for "Top page" and "Chil	itso_jsp_te...	RAD7EJBWeb/WebContent/...	line 40

Figure 4-11 J2EE perspective

More details about using the J2EE perspective can be found in Chapter 16, “Develop Web applications using EJBs” on page 719.

Java perspective

The Java perspective (Figure 4-12) supports developers with the tasks of creating, editing, and compiling Java code.



```
 * File Name:  Credit.java

package itso.rad7.bank.model;

import itso.rad7.bank.exception.InvalidTransac

public class Credit extends Transaction {

    private static final long serialVersionUII

    public Credit(BigDecimal amount) {
        super(amount);
    }

    public BigDecimal process(BigDecimal accou
            throws InvalidTransactionExceptior
        if ((this.getAmount() != null)
                && (this.getAmount().compareTc
            return accountBalance.add(this.get
        } else {
            if (this.getAmount() != null) {
                throw new InvalidTransactionEx
                        "Credit transaction cc
```

BigDecimal itso.rad7.bank.model.Transaction.getAmount()

```
public BigDecimal getAmount() {
    return this.amount;
}
```

Figure 4-12   Java perspective

It consists of a main editor area and displays by default, the following views:

- **Package Explorer view**—Shows the Java element hierarchy of all the Java projects in your Workbench. This is a Java-specific view of the resources shown in the Navigator view (which is not shown by default in the Java perspective). For each project, its source folders and referenced libraries are shown in the tree view and from here it is possible to open and browse the contents of both internal and external JAR files.

- **Hierarchy view**—Can be opened for a selected type to show its super-classes and subclasses. It offers three different ways to look at a class hierarchy, by selecting the icons buttons at the top of the view:
  - The **Type Hierarchy icon** ( ) displays the type hierarchy of the selected type. This includes its position in the hierarchy along with all its superclass and subclasses.
  - The **Supertype Hierarchy icon** ( ) displays the supertype hierarchy of the selected type and any interfaces the type implements.
  - The **Subtype Hierarchy icon** ( ) displays the subtype hierarchy of the selected type or, for interfaces, displays classes that implement the type.

  More information about the Hierarchy view is provided in , “Java Type Hierarchy perspective” on page 138.
- **Javadoc view**—This view shows the Javadoc comments associated with the element selected in the editor or outline view.
- **Declarations view**—Shows the source code declaration of the element selected in the editor or in outline view.

The Outline and Problems views are also applicable to the Java perspective and have already been discussed in earlier sections of this chapter.

Refer to Chapter 7, “Develop Java applications” on page 227 for more information about how to work with the Java perspective and the following two perspectives.

Java Browsing perspective

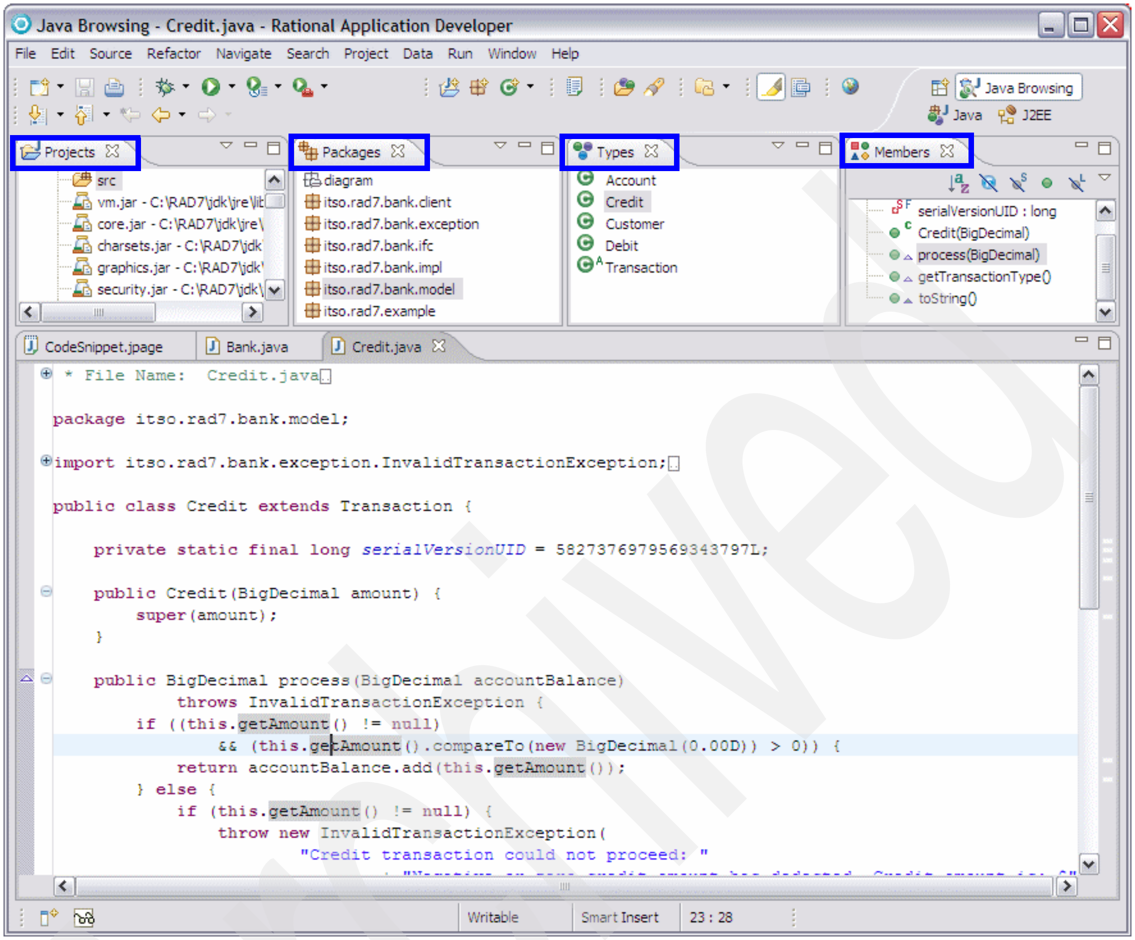
The Java Browsing perspective is also for Java development (Figure 4-13), but it provides different views from the Java perspective.

The Java Browsing perspective has a larger editor area and several views to select the programming element you want to edit:

- **Projects view**—Lists all Java projects
- **Packages view**—Shows the Java packages within the selected project
- **Types view**—Shows the types defined within the selected package
- **Members view**—Shows the members of the selected type

The Show Selected Element Only button ( ) toggles between showing all the content of the selected type and showing only the code (and comments) for the element selected in the Members view.

The Toggle Mark occurrences button ( ) when toggled on highlights all occurrences of the previously search text within the current editor.



```
 * File Name:  Credit.java

package itso.rad7.bank.model;

import itso.rad7.bank.exception.InvalidTransactionException;

public class Credit extends Transaction {

    private static final long serialVersionUID = 5827376979569343797L;

    public Credit(BigDecimal amount) {
        super(amount);
    }

    public BigDecimal process(BigDecimal accountBalance)
            throws InvalidTransactionException {
        if ((this.getAmount() != null)
                && (this.getAmount().compareTo(new BigDecimal(0.00D)) > 0)) {
            return accountBalance.add(this.getAmount());
        } else {
            if (this.getAmount() != null) {
                throw new InvalidTransactionException(
                        "Credit transaction could not proceed: "
```

Figure 4-13   Java Browsing perspective

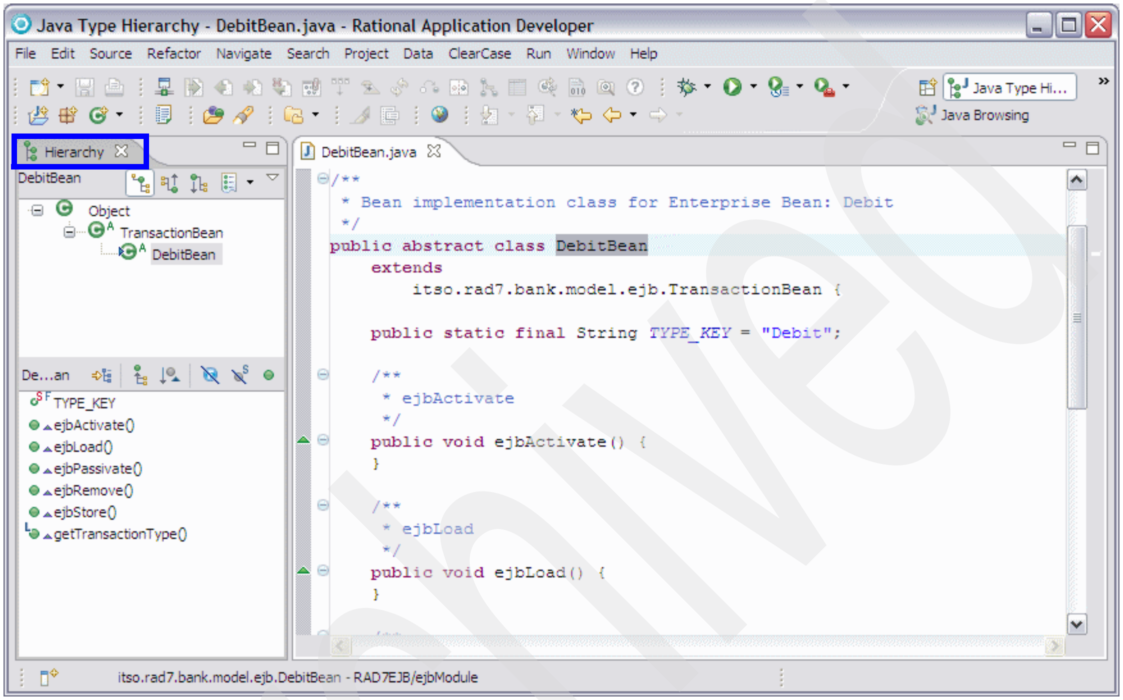
Java Type Hierarchy perspective

This perspective is also for Java developers and allows users to explore the type hierarchy. It can be opened on types, compilation units, packages, projects, or source folders and consists of the **Hierarchy** view and an editor.

The Hierarchy view shows only an information message until you select a type:

> *To display the type hierarchy, select a type (for example, in the outline view or in the editor), and select the 'Open Type Hierarchy' menu option. Alternatively, you can drag and drop an element (for example, project, package, type) onto this view.*

To open a type in the Hierarchy view, open the context menu for a Java class in any view or editor (for example, the main source code editor) and select **Open Type Hierarchy**. The type hierarchy is displayed in the Hierarchy view. Figure 4-14 shows the Hierarchy view of the `DebitBean` class from Chapter 16, “Develop Web applications using EJBs” on page 719.


Java Type Hierarchy - DebitBean.java - Rational Application Developer
File Edit Source Refactor Navigate Search Project Data ClearCase Run Window Help
Java Type Hi...
Java Browsing
Hierarchy
DebitBean
Object
TransactionBean
DebitBean
De...an
TYPE_KEY
ejbActivate()
ejbLoad()
ejbPassivate()
ejbRemove()
ejbStore()
getTransactionType()
DebitBean.java
itso.rad7.bank.model.ejb.DebitBean - RAD7EJB/ejbModule

```
/**
 * Bean implementation class for Enterprise Bean: Debit
 */
public abstract class DebitBean
    extends
        itso.rad7.bank.model.ejb.TransactionBean {

    public static final String TYPE_KEY = "Debit";

    /**
     * ejbActivate
     */
    public void ejbActivate() {
    }

    /**
     * ejbLoad
     */
    public void ejbLoad() {
    }
```

Figure 4-14 Java Type Hierarchy perspective with Hierarchy view

Although the Hierarchy view is also present in the Java perspective and the Java Type perspective only contains two views, it is useful as it provides a way for developers to explore and understand complex object hierarchies without the clutter of other information.

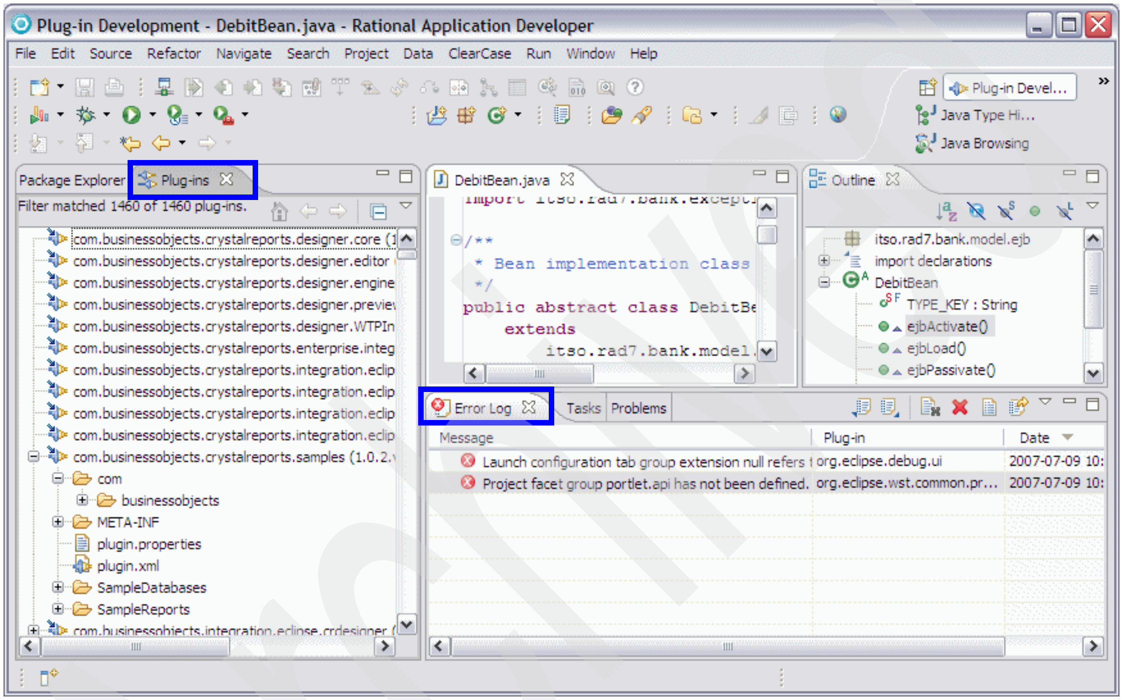
Plug-in Development perspective

The ability to write extra features and plug-ins is an important part of the philosophy of the Eclipse framework. Using this perspective, you can develop your own Application Developer or Eclipse tools.

The Plug-in Development perspective (Figure 4-15) includes:

- **Plug-ins view**—Shows the combined list of workspace and external plug-ins.
- **Error Log view**—Shows the error log for the software development platform, allowing a plug-in developer to diagnose problems with plug-in code.

The perspective also includes Package Explorer, Outline, Tasks, and Problems views, which have already been described earlier in this chapter.


Plug-in Development - DebitBean.java - Rational Application Developer
File Edit Source Refactor Navigate Search Project Data ClearCase Run Window Help
Plug-in Devel...
Java Type Hi...
Java Browsing
Package Explorer
Plug-ins
Filter matched 1460 of 1460 plug-ins.
com.businessobjects.crystalreports.designer.core (1
com.businessobjects.crystalreports.designer.editor
com.businessobjects.crystalreports.designer.engine
com.businessobjects.crystalreports.designer.previe
com.businessobjects.crystalreports.designer.WTPIn
com.businessobjects.crystalreports.enterprise.integ
com.businessobjects.crystalreports.integration.eclip
com.businessobjects.crystalreports.integration.eclip
com.businessobjects.crystalreports.integration.eclip
com.businessobjects.crystalreports.integration.eclip
com.businessobjects.crystalreports.samples (1.0.2.
com
businessobjects
META-INF
plugin.properties
plugin.xml
SampleDatabases
SampleReports
com.businessobjects.integration.eclipse.crdesigner (
DebitBean.java
Outline
itso.rad7.bank.model.ejb
import declarations
DebitBean
TYPE_KEY : String
ejbActivate()
ejbLoad()
ejbPassivate()
Error Log
Tasks
Problems

```
import itso.rad7.bank.excepti
/**
 * Bean implementation class
 */
public abstract class DebitBe
    extends
        itso.rad7.bank.model.
```

Message	Plug-in	Date
Launch configuration tab group extension null refers t	org.eclipse.debug.ui	2007-07-09 10:
Project facet group portlet.api has not been defined.	org.eclipse.wst.common.pr...	2007-07-09 10:

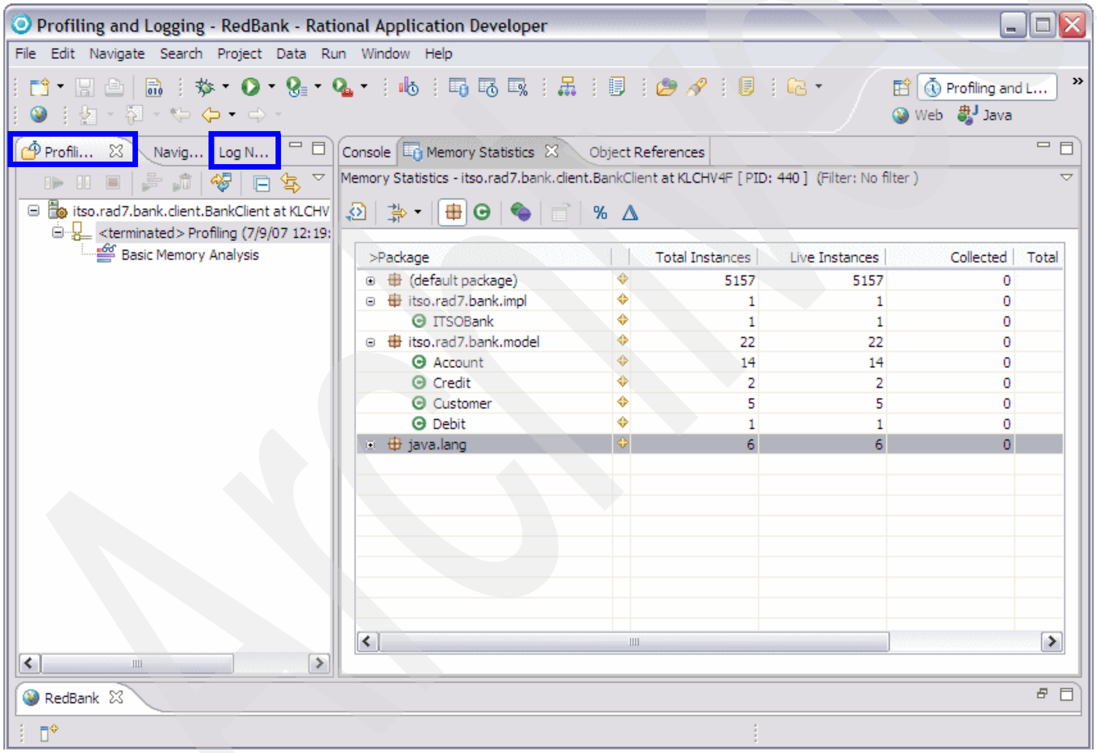
Figure 4-15 Plug-in Development perspective

This book does not cover how to develop plug-ins for Rational Application Developer or Eclipse. To learn more about plug-in development, refer to the IBM Redbooks publication, *Eclipse Development using the Graphical Editing Framework and the Eclipse Modeling Framework*, SG24-6302, or *The Java Developer's Guide to Eclipse-Second Edition,* by D'Anjou et al (refer to http://jdg2e.com).

Profiling and Logging perspective

The Profiling and Logging perspective (Figure 4-16) provides a number of views for working with logs and for profiling applications:

- **Log Navigator view**—This view is used for working with various types of log files, including the service logs for WebSphere Application Server. Logs and symptom databases can be loaded using this view and opened in an editor or viewer.
- **Profiling Monitor view**—This view shows the process that can be controlled by the profiling features of Rational Application Developer. Performance and statistical data can be collected from processes using this feature and displayed in various specialized views and editors.


Profiling and Logging - RedBank - Rational Application Developer
File Edit Navigate Search Project Data Run Window Help
Profiling and L...
Web
Java
Profili...
Navig...
Log N...
Console
Memory Statistics
Object References
RedBank

itso.rad7.bank.client.BankClient at KLCHV
<terminated> Profiling (7/9/07 12:19:
Basic Memory Analysis

Memory Statistics - itso.rad7.bank.client.BankClient at KLCHV4F [PID: 440] (Filter: No filter)

>Package		Total Instances	Live Instances	Collected	Total
(default package)		5157	5157	0	
itso.rad7.bank.impl		1	1	0	
ITSOBank		1	1	0	
itso.rad7.bank.model		22	22	0	
Account		14	14	0	
Credit		2	2	0	
Customer		5	5	0	
Debit		1	1	0	
java.lang		6	6	0	

Figure 4-16 Profiling perspective

In addition to this, there are several editors for viewing the results of profiling, for example the **Memory Statistics** view and the **Object References** view. More details about these views and the techniques required to use them can be found in Chapter 25, “Profile applications” on page 1155.

Report Design perspective

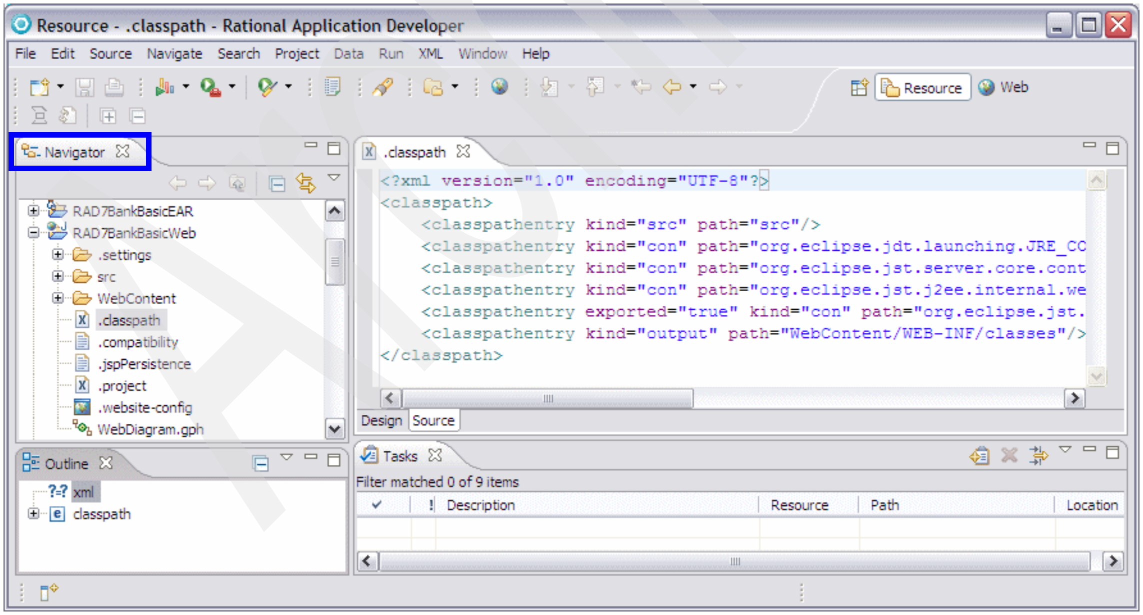
This perspective allows developers to design and develop report templates using the Business Intelligence and Reporting Tools (BIRT) framework, which is an Eclipse-based open source reporting system for Web applications, especially those based on Java and J2EE. The BIRT system also includes a runtime component to process these reports which can be added to an Application Server. The Reports Designer editor provides the facility to layout the fields on the report template and to map these fields with data from XML schemas or database definitions and to test them within Application Developer.

The following link is a reference to the Eclipse project for providing the functionality of this perspective:

`http://eclipse.org/birt/phoenix/project`

Resource perspective

The Resource perspective is a very simple perspective (Figure 4-17). By default it contains only **Navigator**, **Outline**, and **Tasks** views and an editor area. It can be useful when it is necessary to view the underlying files and folders present for a project without any extra information added. All the views in this perspective are available in other perspectives and have been described previously.



```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
    <classpathentry kind="src" path="src"/>
    <classpathentry kind="con" path="org.eclipse.jdt.launching.JRE_CC
    <classpathentry kind="con" path="org.eclipse.jst.server.core.cont
    <classpathentry kind="con" path="org.eclipse.jst.j2ee.internal.we
    <classpathentry exported="true" kind="con" path="org.eclipse.jst.
    <classpathentry kind="output" path="WebContent/WEB-INF/classes"/>
</classpath>
```

Filter matched 0 of 9 items

Figure 4-17 Resource perspective

Rule Builder perspective

The Rule Builder perspective allows the user to work with Active Correlation Technology, which is a system that processes logs and other application outputs according to a set of rules to help diagnose when certain high level system events occur. In particular, this can help diagnose when an error or some other condition worth rasing an alert has occurred. The rules are expressed in a defined XML language and can be configured to run within a running application server. Application Developer provides a Rules Editing wizard in this perspective to aid the rule development process.

Application Developer Help has a section on writing the correlation rules, and for an introduction to Active Correlation Technology refer to:

```
http://www.ibm.com/developerworks/autonomic/library/ac-acact/index.html
```

Team Synchronizing perspective

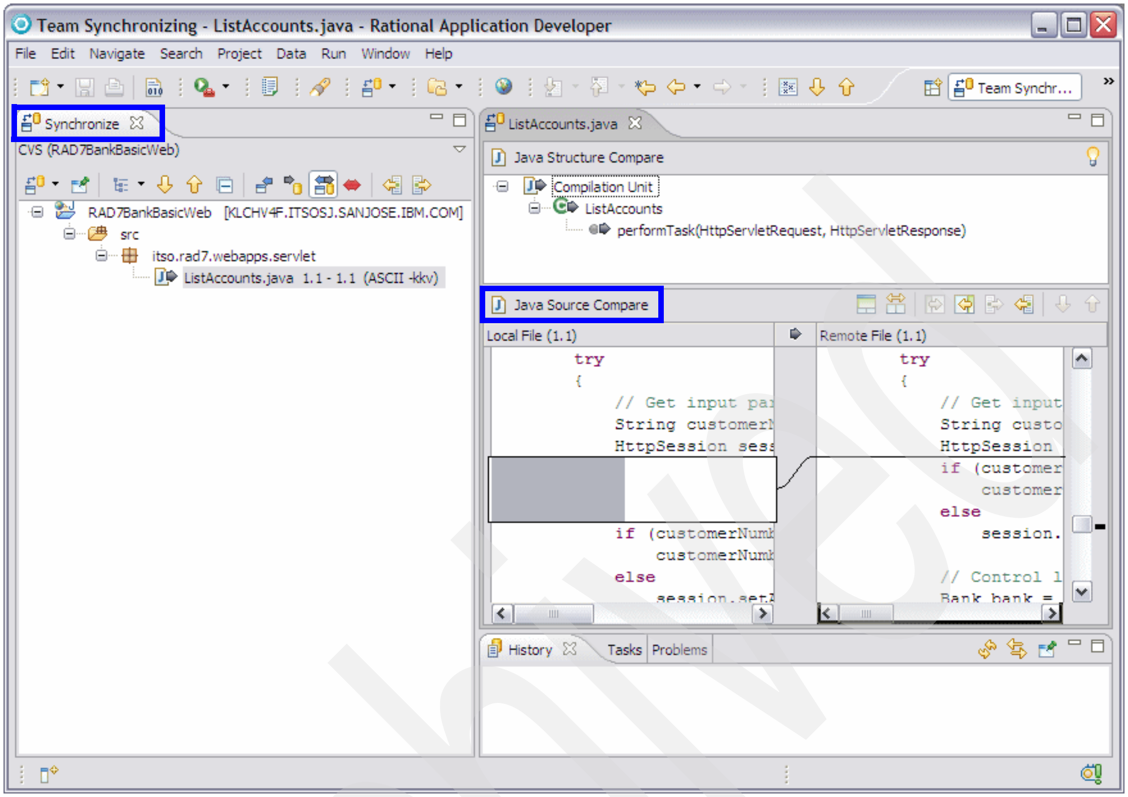
The Team Synchronizing perspective provides the features necessary to synchronize the resources in the workspace with resources held on an SCM repository system. This perspective is used with CVS and ClearCase repositories, plus any other source code repository which might run as an additional plug-in to Application Developer.

Figure 4-17 shows a typical layout while working in the Team Synchronizing perspective.

The following views are important when working in this perspective:

- **Synchronize view**—For any resource with is under source control, the user can select **Team** → **Synchronize**, which prompts the user to move to the Team Synchronizing perspective and show the Synchronize view. It displays the list of synchronization items that result from the analysis of the differences between the local and repository versions of your projects. Double-clicking an item will open the comparison view to help you in completing the synchronization.
- **Source Code Comparison editor**—This editor appears in the main editor area and shows a line by line comparison of two revisions of the same source code.

Also present in the Team Synchronizing perspective is the History view to show the revision history of a given resource file and the Tasks and Problems view. More details about these views on this perspective and how to use them can be found in Chapter 26, “ClearCase integration” on page 1185 and Chapter 27, “CVS integration” on page 1213.


Team Synchronizing - ListAccounts.java - Rational Application Developer
File Edit Navigate Search Project Data Run Window Help
Team Synchr...
Synchronize
CVS (RAD7BankBasicWeb)
RAD7BankBasicWeb [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
src
itso.rad7.webapps.servlet
ListAccounts.java 1.1 - 1.1 (ASCII -kkv)
ListAccounts.java
Java Structure Compare
Compilation Unit
ListAccounts
performTask(HttpServletRequest, HttpServletResponse)
Java Source Compare
Local File (1.1)
Remote File (1.1)
History
Tasks
Problems

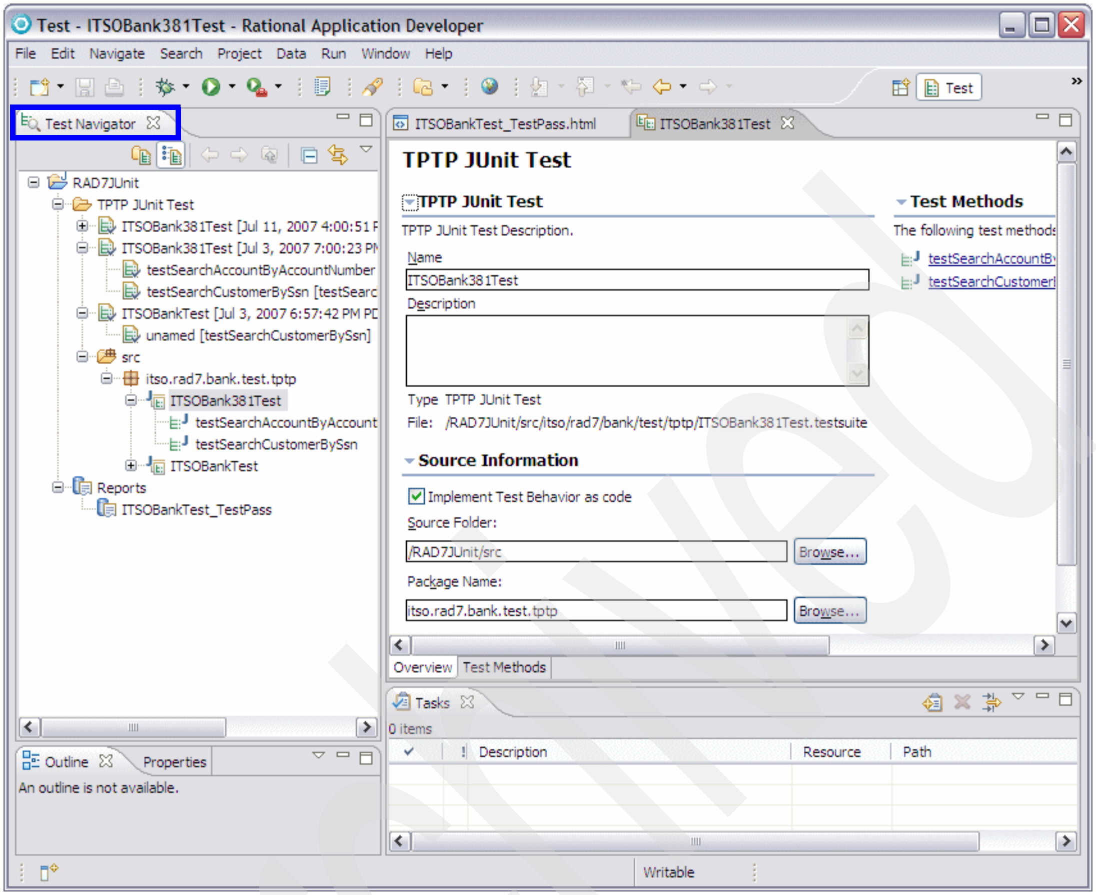
```
try
{
    // Get input pa
    String customerN
    HttpSession sess

    if (customerNumb
        customerNumb
    else
        session.setA
```

```
try
{
    // Get input
    String custo
    HttpSession
    if (customer
        customer
    else
        session.

    // Control l
    Bank bank =
```

Figure 4-18 Synchronizing resources using the Team Synchronizing perspective

Test perspective

The Test perspective (Figure 4-19) provides a framework for defining and executing test cases and test suites. Note that the focus here is on running the tests and examining the results rather than building the code contained in JUnit tests. Building JUnit tests involves writing sometimes complex Java code and is best done in the Java perspective.


Test - ITSOBank381Test - Rational Application Developer
File Edit Navigate Search Project Data Run Window Help
Test
Test Navigator
RAD7JUnit
TPTP JUnit Test
ITSOBank381Test [Jul 11, 2007 4:00:51 F
ITSOBank381Test [Jul 3, 2007 7:00:23 PM
testSearchAccountByAccountNumber
testSearchCustomerBySsn [testSearc
ITSOBankTest [Jul 3, 2007 6:57:42 PM PD
unamed [testSearchCustomerBySsn]
src
itso.rad7.bank.test.tptp
ITSOBank381Test
testSearchAccountByAccount
testSearchCustomerBySsn
ITSOBankTest
Reports
ITSOBankTest_TestPass
Outline
Properties
ITSOBankTest_TestPass.html
ITSOBank381Test
TPTP JUnit Test
Test Methods
Source Information
Browse...
Overview
Test Methods
Tasks
Description
Resource
Path
Writable

An outline is not available.

TPTP JUnit Test Description.

Name: ITSOBank381Test

Description:

Type TPTP JUnit Test

File: /RAD7JUnit/src/itso/rad7/bank/test/tptp/ITSOBank381Test.testsuite

The following test methods

testSearchAccountB

testSearchCustomerI

☑ Implement Test Behavior as code

Source Folder: /RAD7JUnit/src

Package Name: itso.rad7.bank.test.tptp

0 items

Figure 4-19 Test perspective

The following views are important when working in this perspective:

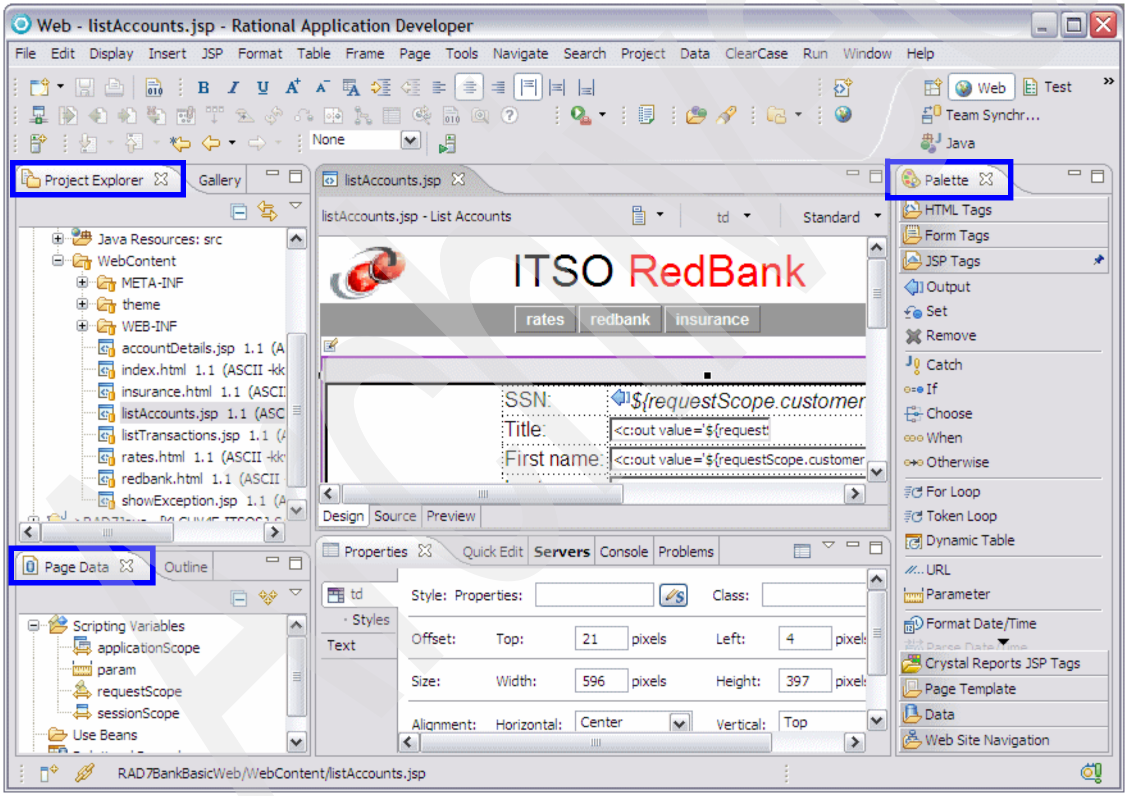
- **Test Navigator**—The main navigator view for browsing and editing test suites and reviewing test results. It has two main options to structure the display of these resources. The **Show the Resource test navigator** ( ) option shows the resources based around the file system but with the test suites displayed at the bottom can containing their associated test cases. The **Show the Logical test navigator** ( ) option shows the resources arranged by Test suites, source code and test results.
- **Test Log view**—If the user clicks on a test result, this view is shown in the main editor area showing the date/time and result of the test.
- **Test editor**—Shows a summary of a test suite and its contained tests.

The Tasks, Properties, and Outline views are also present and useful when working in the Test perspective. These have already been covered in this chapter.

More information about Component Testing is located in Chapter 21, “Test using JUnit” on page 1001.

Web perspective

Web developers can use the Web perspective to build and edit Web resources, such as servlets, JSPs, HTML pages, style sheets and images as well as the deployment descriptor `web.xml`. Figure 4-20 shows a typical layout while developing in this perspective.



Style: Properties: Class:

Offset: Top: 21 pixels Left: 4 pixels

Size: Width: 596 pixels Height: 397 pixels

Alignment: Horizontal: Center Vertical: Top

Figure 4-20 Web perspective

The Web perspective contains the following views and editors:

- **Page Designer**—Page Designer allows you to work with HTML files, JSP files, and embedded JavaScript. Within the Page Designer, you can move among three tabs that provide different ways for you to work with the file that you are editing. This editor is synchronized with the Outline and Properties views, so that the selected HTML or JSP element always appears in these views. The main tabs provided by Page Designer are as follows:
  - **Design**—The Design page of Page Designer is the WYSIWYG mode for editing HTML and JSP files. As you edit in the Design page, your work reflects the layout and style of the Web pages you build without the added complexity of source tagging syntax, navigation, and debugging. Although all tasks can also be done in the Source page, the Design view should allow most operations to be done more efficiently and without requiring a detailed knowledge of HTML syntax.
  - **Source**—The Source page enables you to view and work with a file's source code directly.
  - **Preview**—Shows how the current page is likely to look when viewed in a Web browser. JSPs shown in this view will contain only static HTML output.
- **Gallery view**—Contains a variety of catalogs of reusable files that can be applied to Web pages. The file types available include images, wallpaper, Web art, sound files and style sheet files.
- **Page Data view**—Allows you manage data from a variety of sources, such as session EJBs, JavaBeans, and Web services which can be configured and dropped onto a JSP.
- **Styles view**—Provides guided editing for cascading style sheets and individual style definitions for HTML elements.
- **Thumbnails view**—Shows thumbnails of the images in the selected project, folder, or file. This view is especially valuable when used with the Gallery view to add images from the artwork libraries supplied by Application Developer to your page designs. When used with the Gallery view, thumbnail also displays the contents of a selected folder. You can drag and drop from this view into the Project Explorer view or to the Design or Source page of Page Designer.
- **Quick Edit view**—Allows you to edit small bits of code, including adding and editing actions assigned to tags. This view is synchronized with what element is selected in the Page Designer. You can drag and drop items from the Snippets view into the Quick Edit view.
- **Palette view**—Contains expandable drawers of drag and drop objects. Allows you to drag objects, such as tables or form buttons, onto the Design or Source page of the Page Designer.

- **Links view**—This view (Figure 4-21) is available as a fast view, by selecting the link icon ( ) on the bottom left corner of the Web perspective. It shows the resources used by or linked to from the selected file and files that use or link to the selected resource. This view provides you with a way to navigate through the various referenced parts of a Web application. Clicking the button toggles the display of this view.

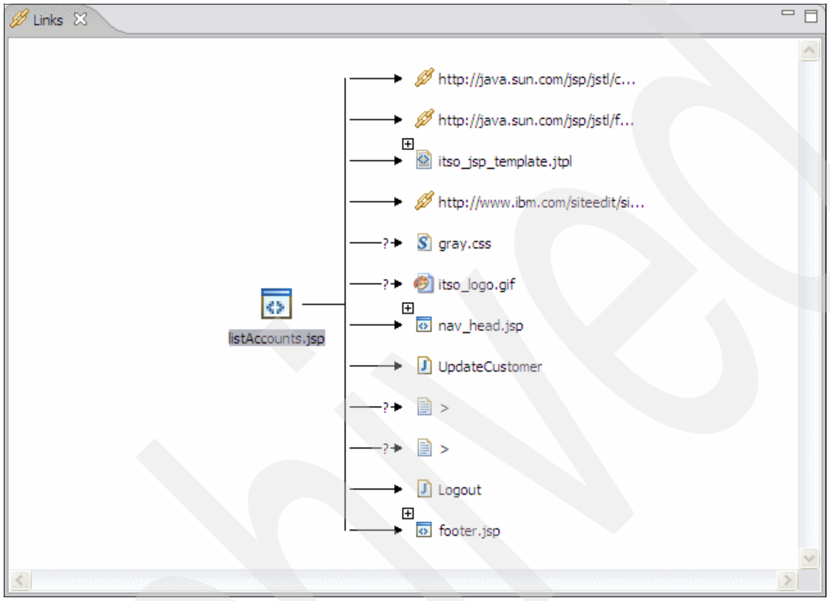


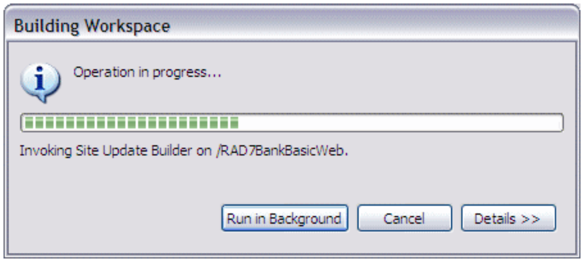
Figure 4-21 Links view in the Web perspective

The Project Explorer, Outline, Properties, Servers, Console, Problems, and Snippets views are also present in the Web perspective and have already been discussed in this chapter.

More information about developing JSPs and other Web application components in the Web persecutive can be found in Chapter 12, “Develop Web applications using JSPs and servlets” on page 465.

Progress view

The Progress view is not part of any perspective by default, but is a very useful tool when using Rational Application Developer. When Rational Application Developer is carrying out a task that takes a substantial amount of time, a prompt might appear with two options available (Figure 4-22).



Building Workspace

Operation in progress...

Invoking Site Update Builder on /RAD7BankBasicWeb.

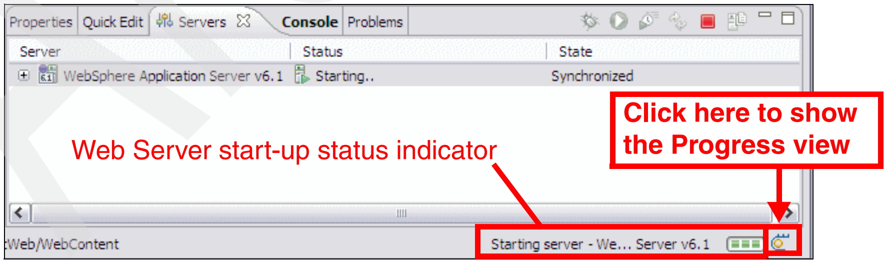
Figure 4-22 Progress view

The user can either watch the dialog until the operation completes, or the user can click **Run in Background** and the task continues in the background. If the second option is selected, Rational Application Developer runs more slowly, but the developer can carry out other tasks while waiting. Examples of tasks that might be worth running in the background would be publishing and running an enterprise application, checking a large project into CVS, or rebuilding a complex set of projects.

If **Run in Background** is clicked, then the **Progress** view can be shown again to review the status of the running task by clicking the icon in the bottom right of the workspace. Note that this icon only shows if there are processes running in the background.

Some processes do not prompt the user with a dialog and run in the background when they are initiated. In these cases, the Progress view can be accessed in the same way.

For example, when a Web application is published to the test server and the server has to be started, this process might take some time. By default this condition shows as a flashing status bar in the bottom left of the workspace and the icon to show the Progress view appears (Figure 4-23).



Server	Status	State
WebSphere Application Server v6.1	Starting..	Synchronized

:Web/WebContent

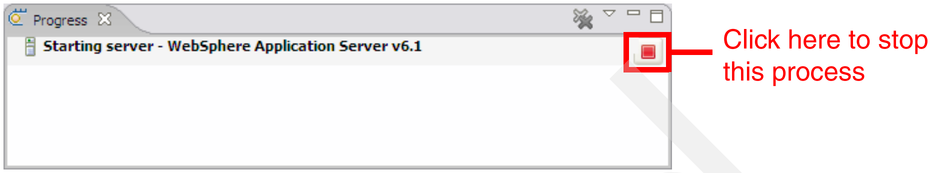
Starting server - We... Server v6.1

Web Server start-up status indicator

Click here to show the Progress view

Figure 4-23 Process information in status bar

If the user becomes concerned about the time the deployment process is taking, then the Progress view can be opened, the current process reviewed, and if necessary, stopped (Figure 4-24).



Starting server - WebSphere Application Server v6.1

Figure 4-24 Progress view

Summary

In this chapter, the perspectives available within Application Developer and the main views associated were described. Parts 2, 3, 4, and 5 of this book demonstrate in detail the use of these perspectives for various development scenarios.

5

Projects

This chapter provides a high level summary of the types projects within Application Developer and their main features, in particular, those used for building J2EE applications. We start with a review of the basic features of the J2EE framework and then describe the fundamental structures and mechanisms used by Application Developer for handling projects and the many types of projects available.

Following this, we focus on the project types relating to J2EE applications, including project interrelationships and how they are packaged for deployment. We then provide a description of how to create a project and set some simple project properties. Finally, we introduce the project samples gallery.

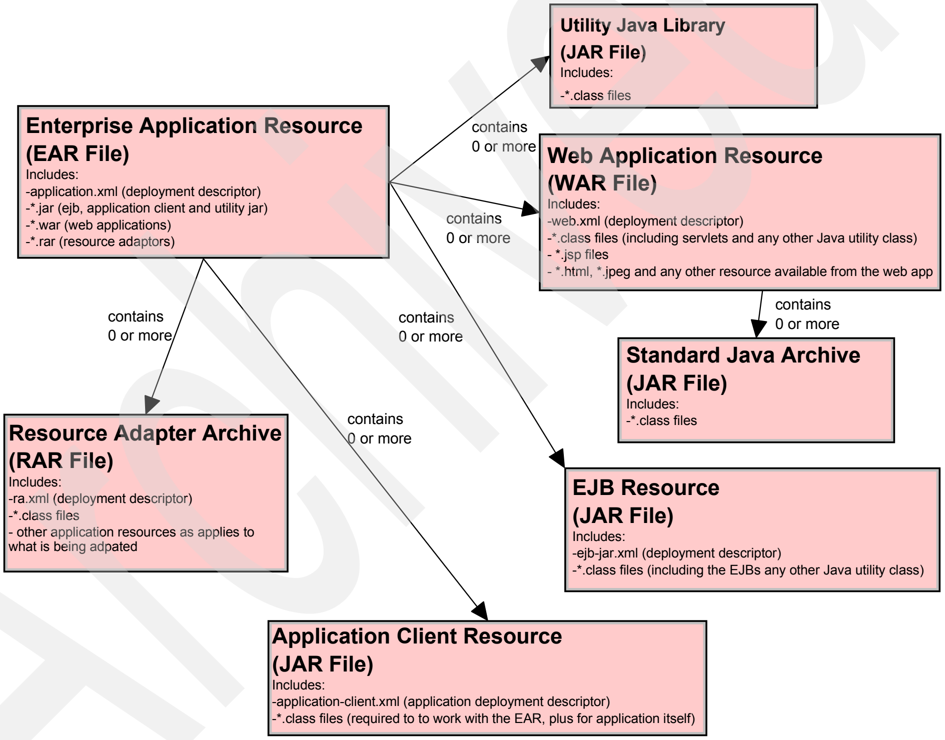
The chapter is organized into the following sections:

- J2EE architecture
- Project basics
- Rational Application Developer projects
- Creating a new project
- Project properties
- Rational Application Developer samples
- Summary

J2EE architecture

The Java 2 Platform Enterprise Edition (J2EE) is a highly available, reliable, scalable, and secure platform for developing Web and client-server applications in Java. The examples in this book use J2EE V1.4, which is the latest version of J2EE to be supported by Rational Application Developer V7.0.

The J2EE specification, along with many other resources relating to J2EE, is available at `http://java.sun.com/j2ee/`. This specification includes a description of the component model for developing J2EE applications and the runtime architecture. A high level view of a J2EE enterprise application structure is shown in Figure 5-1.



Utility Java Library
(JAR File)
Includes:
-*.class files
Enterprise Application Resource
(EAR File)
Includes:
-application.xml (deployment descriptor)
-*.jar (ejb, application client and utility jar)
-*.war (web applications)
-*.rar (resource adaptors)
contains
0 or more
Web Application Resource
(WAR File)
Includes:
-web.xml (deployment descriptor)
-*.class files (including servlets and any other Java utility class)
- *.jsp files
- *.html, *.jpeg and any other resource available from the web app
contains
0 or more
contains
0 or more
Standard Java Archive
(JAR File)
Includes:
-*.class files
contains
0 or more
contains
0 or more
contains
0 or more
Resource Adapter Archive
(RAR File)
Includes:
-ra.xml (deployment descriptor)
-*.class files
- other application resources as applies to what is being adpated
EJB Resource
(JAR File)
Includes:
-ejb-jar.xml (deployment descriptor)
-*.class files (including the EJBs any other Java utility class)
Application Client Resource
(JAR File)
Includes:
-application-client.xml (application deployment descriptor)
-*.class files (required to to work with the EAR, plus for application itself)

Figure 5-1 J2EE modules

The following section details the main components of a J2EE Enterprise Application. Subsequent sections describe the types of projects within Application Developer that can be used to create these components.

EAR files

Enterprise archive (EAR) files represent a J2EE application that can be deployed to WebSphere Application Server. EAR files are standard Java archive files based on zip and conform to a defined directory and file structure. An EAR (`.ear`) file contains a deployment descriptor (`application.xml`) which describes the contents of the application and contains instructions for running and deploying the application. These instructions include things such as security settings to be used in the runtime environment and references to Web applications contained within the application resource.

An EAR file can include zero or more of the following modules:

- Web modules—Packaged in a WAR file (`.war`)
- EJB modules—Packaged in an EJB JAR file (`.jar`)
- Application client modules—Packaged in a JAR file (`.jar`)
- Resource adapter modules—Packaged in a RAR file (`.rar`)
- Utility JAR files required by other modules—Packaged in a JAR file (`.jar`)

WAR files

Web archive (WAR) files contain all the components of a Web application which can be deployed to a J2EE compliant application server. These components often include:

- HyperText Markup Language (HTML) files
- Cascading style sheets (CSS) files
- JavaServer Pages (JSP) files
- Compiled Java servlet classes
- Other compiled Java classes
- Image files
- Portlets (portal applications)

WAR files also include a deployment descriptor (`web.xml`), which describes how to deploy the various components packaged within the Web module and to map incoming Web requests to the correct resources.

EJB JAR files

An EJB JAR file is used to package all the classes and interfaces that make up the EJBs in an enterprise application. The EJB JAR file includes a deployment descriptor (`ejb-jar.xml`), which describes how the EJBs should be deployed.

J2EE Application Client JAR files

A J2EE Application Client is packaged in a JAR file. The JAR file includes the classes required for the user interface for the application, which runs on a client machine and access resources provided by a J2EE server, such as data sources, security services, and EJBs. The file also includes a deployment descriptor (`application-client.xml`).

RAR files

A resource adapter archive (RAR) file is used to package J2EE resource adapters. These provide access to back-end resources using services provided by an application server to manage the lookup of the resource and to manage connections. Resource adapters are often provided by vendors of enterprise information systems to facilitate access from J2EE applications to resources such as mainframe systems or non-Java applications.

The resource adapter can be installed as a stand-alone module within the server, allowing it to be shared among several applications, or as part of an enterprise application, in which case it is available only to the associated application and modules contained within it. The RAR file includes the Java classes, providing the Java-side of the resource adapter, and platform-specific code, providing the back-end connection functionality. The deployment descriptor is called `ra.xml`.

J2EE resource adapters are known by several names:

- Java 2 Connectors (J2C)
- J2EE Connector Architecture (JCA)
- J2EE Resource Adapter (RA)

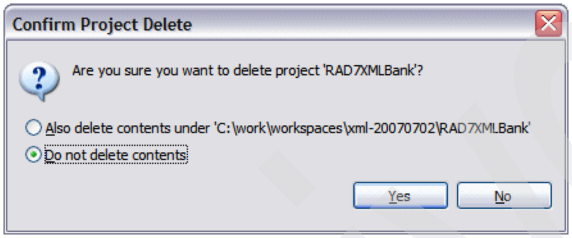
Project basics

Within any Rational Application Developer work-space all resources are organized into projects which contain the files, folders, and properties required for the application under construction. In a workspace you can create different kinds of projects with different structures as required for different applications. The full set of available projects is listed in “Rational Application Developer projects” on page 156.

Unless otherwise specified, projects are stored in the Application Developer workspace directory, which is normally chosen as the tool is started.

Deleting projects

When projects are deleted from a workspace, Rational Application Developer offers the choice of **Also delete contents under <directory>** or **Do not delete contents** (Figure 5-2). Selecting the second option (the default) removes the project from Rational Application Developer’s list of projects, but leaves the file structure of the project intact. These projects can later be imported by selecting **File** → **Import** → **General** → **Existing Projects Into Workspace**. A project that has been deleted from the workspace takes up no memory and is not examined during builds, so deleting projects in this way can improve the performance of Application Developer.



Confirm Project Delete

Are you sure you want to delete project 'RAD7XMLBank'?

○ Also delete contents under 'C:\work\workspaces\xml-20070702\RAD7XMLBank'

● Do not delete contents

Figure 5-2 Confirm Project Delete dialog

Project interchange file

Projects can be transferred between workspaces as project interchange files, which is a file format based on zip to encapsulate a project. To create a project interchange for any project simply select **File** → **Export** → **Other** → **Project Interchange** and specify which projects to export and the location. To import projects into another workspace select **File** → **Export** → **Other** → **Project Interchange**. Note that when exporting and importing a project, the project interrelationships are also transferred but not the referenced projects; it might also be necessary to export all related projects.

Closing projects

It is also possible to close projects on the workspace, which means their contents are locked and cannot be edited, and other projects cannot reference any of the resources inside them. This can be done using either the **Close Project** or **Close Unrelated Projects** from the Project Explorer context menu. Closed projects remain in the Project Explorer view, but they cannot be expanded.

Closing non-important projects can speed up compilation times as the underlying application builders only have to check for resources in open projects. Closed projects can be re-opened by selecting **Open Project** from Project Explorer.

Rational Application Developer projects

The following list describes wizards that can be used to create projects within Application Developer. To invoke the wizard, simply use **File** → **New** → **Project** and select the appropriate project wizard. The wizard will then prompt the user for the required information as appropriate for the type of project:

- **Project (General)**—This is the simplest project which just contains a collection of files and folders. It contains no builders and is useful for creating a project which has no application code, for example, a project to store XML or XSD files or to store application configuration information.
- **Report Project (Business Intelligence and Reporting tools, BIRT)**—The BIRT system (refer to `http://eclipse.org/birt/phoenix/project`) is an initiative to build an open source reporting system in Java. This wizard creates a report project which has facilities to combine database information or content from XML into report templates.
- **Crystal Reports Web project**—This creates a Web project with Crystal reports features activated. The new project includes the libraries for the Java Reporting Component and support for Crystal Reports Viewer pages, which is built on JSP technology. Note that support for Crystal reports must be selected on installation of Application Developer for this to be available.
- **User Function Library (Crystal Reports)**—This type of project allows Java code to be called from a Crystal reports formula. The wizard creates a project very similar to a Java project but with links to the Crystal Reports Java libraries.
- **Projects from CVS**—This wizard guides the user through creating a new project by checking out an existing project within CVS. It is possible to check-out a complete project from CVS, or to create a new project as part of the check-out process.
- **Data Design Project**—This wizard creates a project to store data design artifacts, including data design models and SQL statements.
- **Data Development Project**—This wizard creates a project which stores a connection to a given database. From within such a project it is possible to create resources to interrogate and manipulate the associated database. Initially the wizard creates folders to store SQL scripts and stored procedures.
- **Existing RAD6.x Data Definition Project**—The tooling which supports database definitions has changed since Application Developer V6.0.X and V5.1.2. Therefore, any data project which contain database definitions or other database objects created in the Data definition view from previous versions of Application Developer must be migrated to work with V7.0. This wizard takes a project folder in the old format and migrates it for V7.0.

- **EMF Project (Eclipse Modelling Framework)**—This wizard guides the user through creating a new Java project based on an imported model (either XML, Ecore, UM, or Rose class model). The new project will include the model documents and provides the facility to generate Java code from them.
- **EJB Project**—Guides the user through the process of creating a project suitable for containing EJBs. This procedure also creates an empty EJB deployment descriptor and associates the project with a containing enterprise application project.
- **Application Client Project (J2EE)**—Guides the user through creating an empty Application Client project. Includes prompts for the associated EAR project and a list of facets applicable for J2EE Application Client projects.
- **Connector Project (J2EE)**—Guides the user through creating a J2EE connector project, which includes specifying the associated enterprise application project and a set of facets applicable to it.
- **Enterprise Application Project (J2EE)**—Creates a new EAR project. This includes options for creating associated Web, EJB, and Application Client projects.
- **Utility Project (J2EE)**—Assists in the construction of a Java utility library project which is associated with an Enterprise Application project. The product of such a project is a JAR file that is available to the resources within the Enterprise project and is automatically wrapped into the EAR file when it is exported.
- **Java Project**—Simple wizard for a Java project. This provides an interface for specifying the class path including project dependencies.
- **Java Project from Existing Ant Buildfile**—It is possible to export the build settings of a project as an Ant file (use **Export** → **Export** → **General** → **Ant Buildfiles** from the **Project Explorer**). Given such an Ant build file, this wizard can be used to create a new project based on the instructions contained within it.
- **Jython Project**—Creates an empty project for developing Jython resources.
- **Feature Patch, Feature Project, Fragment Project, Plug-in from Existing JAR Archives, Plug-in Project and Update Site Project (Plug-in Development)**—These wizards assist in the creation of Eclipse plug-ins, features, and fragments, which can enhance existing Eclipse (or Application Developer) perspectives or create entirely new ones. Rational Application developer help has a section on how to use these wizards, and the Eclipse plugin central home page (`http://www.eclipseplugincentral.com`) has information on many plug-ins already built and tutorials on building new ones.

- **SIP Project (Session Initiated Protocol Project)**—The SIP protocol is an extension to the J2EE servlets API intended for telecommunications applications using technologies such as VOIP. This wizard creates a Web project with the appropriate facets selected to allow the construction of SIP applications.
- **Dynamic Web Project**—Builds the project for a Web application, which can include JSPs, servlets, and other dynamic content.
- **Static Web Project**—Builds a project for a Web application containing only images, HTML files, and other static Web resources.

Each wizard will create an empty project of the specified type with the structures, files, folders, supporting libraries, and references to support such a project. Once created, it is still possible to change aspects of the project through the project properties.

Application Developer J2EE projects

Rational Application Developer complies with the J2EE specifications for developing enterprise and Web applications.

The packaging rules, described above in “J2EE architecture” on page 152, are only applied by Rational Application Developer when a J2EE application or module is exported.

While these applications and modules are being worked on within Rational Application Developer, they are stored as projects within the workspace. The relationship between the enterprise application projects, and the modules they contain, is managed by Rational Application Developer and applied on export to produce a properly packaged `.ear` file.

This arrangement between projects and their associated outputs is shown in Figure 5-3. Note that this diagram relates to Figure 5-1 on page 152, where the relationships between various J2EE modules are reflected in the Application Developer project references.

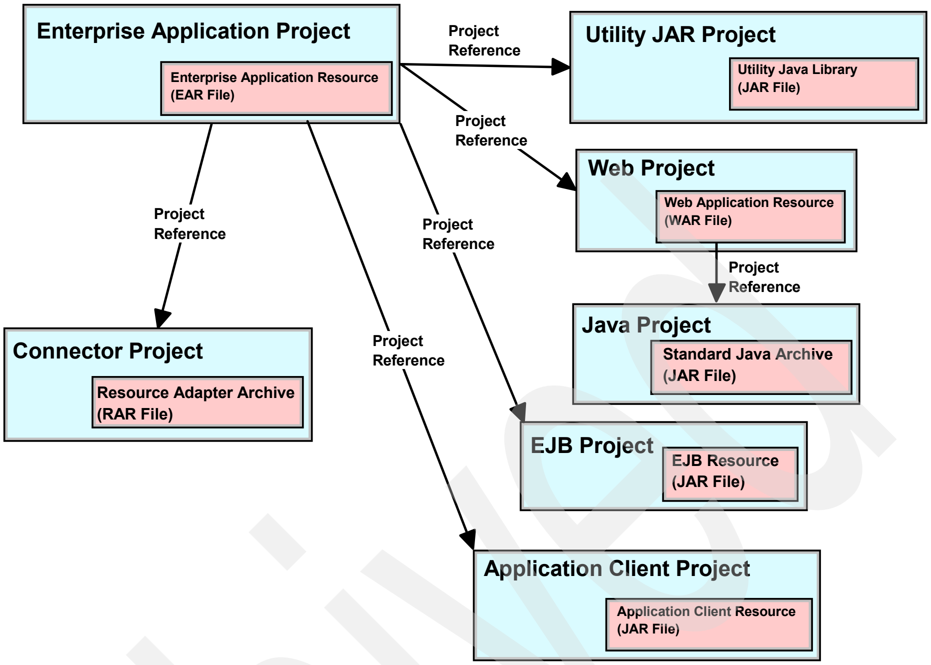

Enterprise Application Project
Enterprise Application Resource (EAR File)
Project Reference
Utility JAR Project
Utility Java Library (JAR File)
Project Reference
Web Project
Web Application Resource (WAR File)
Project Reference
Project Reference
Project Reference
Java Project
Standard Java Archive (JAR File)
Connector Project
Resource Adapter Archive (RAR File)
Project Reference
EJB Project
EJB Resource (JAR File)
Application Client Project
Application Client Resource (JAR File)

Figure 5-3 J2EE projects in Application Developer

Enterprise Application project

Enterprise Application projects contain the resources needed for enterprise applications and can contain references to a combination of Web modules, EJB modules, application client modules, resource adapter modules, utility Java projects, and JAR files.

These relationships can be specified when creating a new Enterprise Application project through the wizard or through the project properties.

For more information on developing J2EE enterprise applications, see Chapter 16, “Develop Web applications using EJBs” on page 719.

J2EE Application Client project

J2EE Application Client projects contain the resources needed for J2EE application client modules. An application client module is used to contain a fully-functioning client Java application (non-Web-based) that connects to and uses the J2EE resources in a J2EE enterprise application and an application server. By holding a reference to the associated enterprise application, it shares information such as the Java Naming and Directory Interface (JNDI) reference to EJBs and data sources.

The wizard allows the J2EE version, the target server, and the associated enterprise application to be specified. For more information on developing J2EE client applications, refer to Chapter 17, “Develop J2EE application clients” on page 793.

Dynamic Web project

A dynamic Web project contains resources needed for Web applications, such as JSPs, Java Servlets, HTML and other files. The dynamic Web project wizard provides the capability to configure the version of the Java servlet specification, target server, EAR file name, and context root. The wizard also allows various other features to be added to the dynamic Web project, including:

- A CSS file
- Struts support
- A Web diagram
- JSP tag libraries
- Web page templates
- Struts support
- JSP support

When building a dynamic Web project, the user is prompted for which facets are to be used by the new project, and then the wizard automatically adds the supporting libraries and configuration files to the new project. By selecting the appropriate facets, it is possible to create a project which uses Struts or JavaServer Faces as a framework for building a Web application.

For more information on developing Web applications, see Chapter 12, “Develop Web applications using JSPs and servlets” on page 465.

Static Web project

A static Web project contains only static Web content, such as HTML pages, images, sounds, and movie files. These resources are designed to be deployed to a Web server and do not require the services of a J2EE application server. The wizard allows the CSS file for the project to be selected, and a Web page template can be selected or constructed.

For more information on developing Web applications, see Chapter 12, “Develop Web applications using JSPs and servlets” on page 465.

EJB project

EJB projects contain the resources for EJB applications. This includes the classes and interfaces that make up the EJBs, the deployment descriptor for the EJB module, IBM extensions, and bindings files, and files describing the mapping between entity beans in the project and relational database resources.

The wizard allows the EJB version, target server, and containing EAR file to be specified and a series of facet features applicable for EJBs. An EJB Client JAR can also be created, which includes all the resources needed by client code to access the EJB module (the interfaces and stubs).

For more information on developing EJBs, see Chapter 16, “Develop Web applications using EJBs” on page 719.

Connector project

A connector project contains the resources required for a J2EE resource adapter. The wizard allows a set of facets (including the J2EE Connector Architecture (JCA) version) and containing EAR file to be specified.

Java project

A Java project contains Java packages and Java code as `.java` files and `.class` files. They have an associated Java builder that incrementally compiles Java source files as they are changed and can be exported as JAR files or into a directory structure.

Java projects are used for stand-alone applications or to build utility JAR files for an enterprise application.

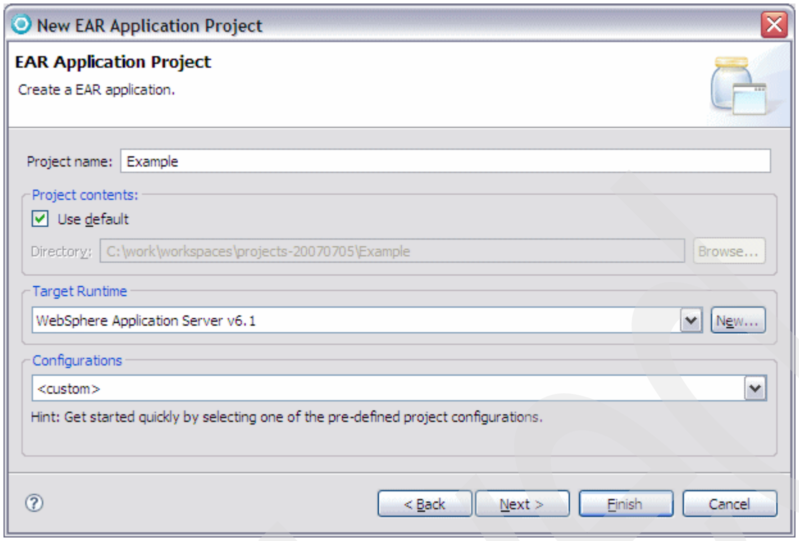
For more information on building Java projects, see Chapter 7, “Develop Java applications” on page 227.

Creating a new project

Development on a new application is usually started by building one or more projects in the Application Developer workspace. As part of a solution design, the projects required should be planned beforehand, and then the wizards described previously can be used to create a skeleton set of projects for the application under construction.

As an example of running through the process of creating a new project, the following instructions demonstrate the New Enterprise Application wizard:

- To launch this wizard, select **File** → **New** → **Project**.
- Select **J2EE** → **Enterprise Application Project**. The first page of the New Enterprise Application Project wizard is shown in Figure 5-4.



New EAR Application Project

EAR Application Project

Create a EAR application.

Project name: Example

Project contents:

☑ Use default

Directory: C:\work\workspaces\projects-20070705\Example

Target Runtime

WebSphere Application Server v6.1

Configurations

<custom>

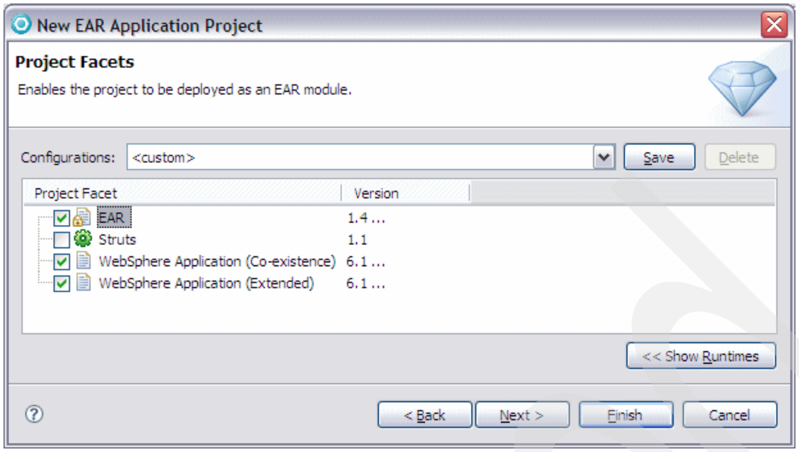
Hint: Get started quickly by selecting one of the pre-defined project configurations.

Figure 5-4 New EAR Application Project wizard (1)

On this page you can specify:

- **Name**—The project name, `Example` in this case.
- **Project contents**—By default, projects are stored in a subdirectory that is created for the project in the workspace directory. Using this option, another location can be specified.
- **Target Runtime**—An enterprise application project is intended to run on an Application Server. This option allows the user to configure the target runtime environment.
- **Configuration**—This drop-down provides a list of saved configurations which have been created for previous enterprise application projects. A configuration can include a specific set of features and available versions and it is often a good idea to make sure all similar projects use the same configuration. An existing configuration can be chosen on this option or `<custom>` can be selected, which means that the user has the opportunity to specify their own selections in the next screen of the wizard.

- The second page of the wizard (Figure 5-5) is the **Project Facets** page. This page allows the user to customize which features and versions of these features will be available in the new project. It is also possible to save the configuration used for subsequent projects. This project facets page also appears when creating a new Web, EJB, and connector project with facet options applicable to the type of project being created.



New EAR Application Project

Project Facets

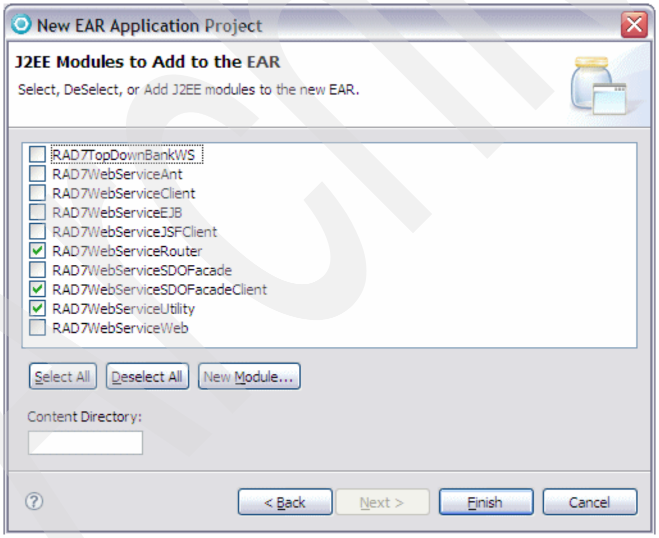
Enables the project to be deployed as an EAR module.

Configurations: <custom>

Project Facet	Version
☑ EAR	1.4 ...
□ Struts	1.1
☑ WebSphere Application (Co-existence)	6.1 ...
☑ WebSphere Application (Extended)	6.1 ...

Figure 5-5 New EAR Application Project wizard (2)

- ▸ The final page (Figure 5-5) gives the user the opportunity to select any projects that are part of the enterprise application. This dialog includes select boxes for all the Java, EJB, Web and Application Client projects in the current workspace, which, if selected, will be in the project references for the new project.



New EAR Application Project

J2EE Modules to Add to the EAR

Select, DeSelect, or Add J2EE modules to the new EAR.

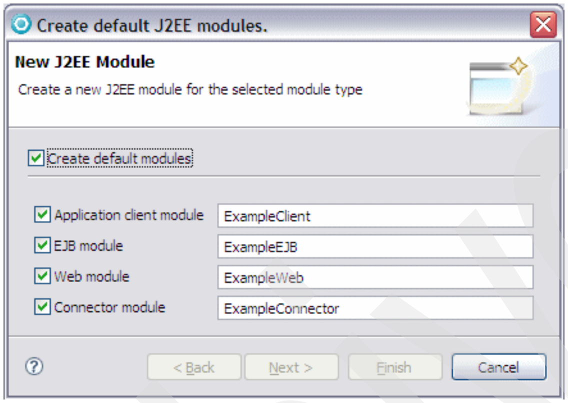
□ RAD7TopDownBankWS
□ RAD7WebServiceAnt
□ RAD7WebServiceClient
□ RAD7WebServiceEJB
□ RAD7WebServiceJSFClient
☑ RAD7WebServiceRouter
□ RAD7WebServiceSDOFacade
☑ RAD7WebServiceSDOFacadeClient
☑ RAD7WebServiceUtility
□ RAD7WebServiceWeb

Content Directory:

Figure 5-6 New EAR Application Project wizard (3)

On this page you can specify:

- **Content Directory**—This specifies the folder within the Enterprise Application project under which the contents will be stored. This can be left empty meaning that all contents will be stored under the root directory.
- **New Module**—This button provides the ability to automatically create empty projects referenced by the new Enterprise Application project. Clicking **New Module** displays the dialog shown in Figure 5-7.



Create default J2EE modules.

New J2EE Module

Create a new J2EE module for the selected module type

☑ Create default modules

☑ Application client module: ExampleClient

☑ EJB module: ExampleEJB

☑ Web module: ExampleWeb

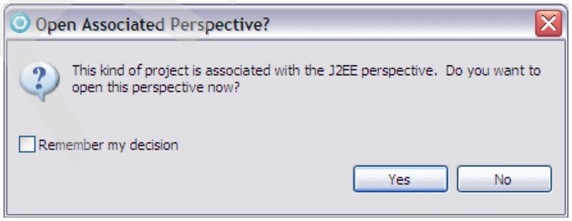
☑ Connector module: ExampleConnector

Figure 5-7   Create default J2EE modules

- Click **Cancel** if you do not want to create any modules.
- Select the boxes for the projects you want to create, change the names if desired, and click **Finish**.

▶ Click **Finish** again in the New Enterprise Application Project wizard (Figure 5-6), and the new project (and associated projects) are created.

If the project you have created is associated with a particular perspective (as in this case, with the J2EE perspective), Rational Application Developer offers to switch over to the relevant perspective (Figure 5-8), and from there development can commence.



Open Associated Perspective?

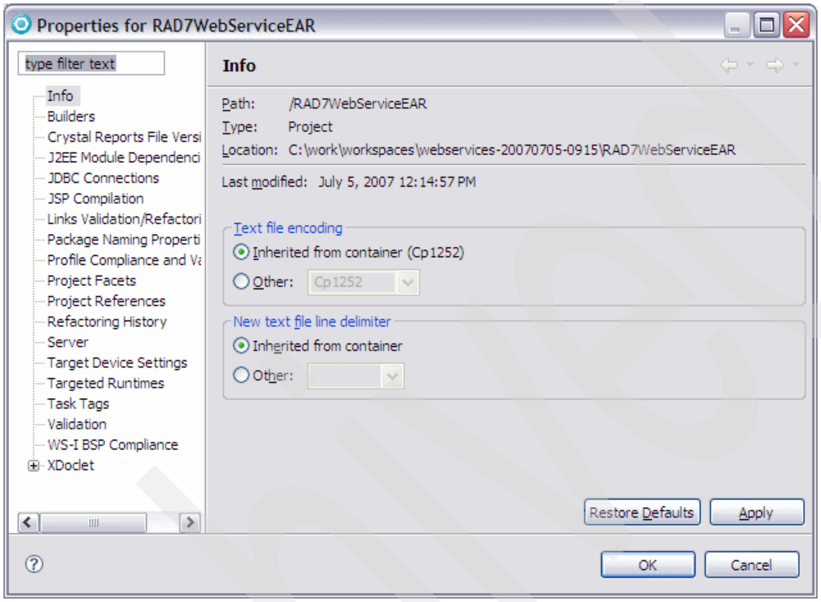
This kind of project is associated with the J2EE perspective. Do you want to open this perspective now?

□ Remember my decision

Figure 5-8   Prompt to switch perspective

Project properties

To make changes to the properties of a project, right-click the project and select **Properties** from the context menu. Figure 5-9 shows the properties dialog for an Enterprise Application project.



Properties for RAD7WebServiceEAR

Info

Path: /RAD7WebServiceEAR
Type: Project
Location: C:\work\workspaces\webservices-20070705-0915\RAD7WebServiceEAR

Last modified: July 5, 2007 12:14:57 PM

Text file encoding
● Inherited from container (Cp1252)
○ Other: Cp1252

New text file line delimiter
● Inherited from container
○ Other:

Figure 5-9 Project properties

In the Properties dialog you can edit most project attributes, and different types of projects have different options available.

The options available for an Enterprise Application project include these:

- **J2EE Module Dependencies**—Any other projects (Web, Java, or Resource) which this project is dependent upon.
- **Project Facets**—Shows which facets are available for this project and gives the opportunity to add or remove any of them.
- **Project References**—Configure project dependencies and classpath entries.
- **Server**—Specifies the default application server to use when running this application.
- **Validation**—Indicates whether to run any non-default validation tools, and if so, which ones to run after making changes.
- **Java Build Path** (not for an enterprise application)—Specifies the build path used to compile the Java code of the project.

Rational Application Developer samples

Rational Application Developer provides a wide range of sample applications that can help you to explore the features provided by the software development platform and the different types of projects that can be created.

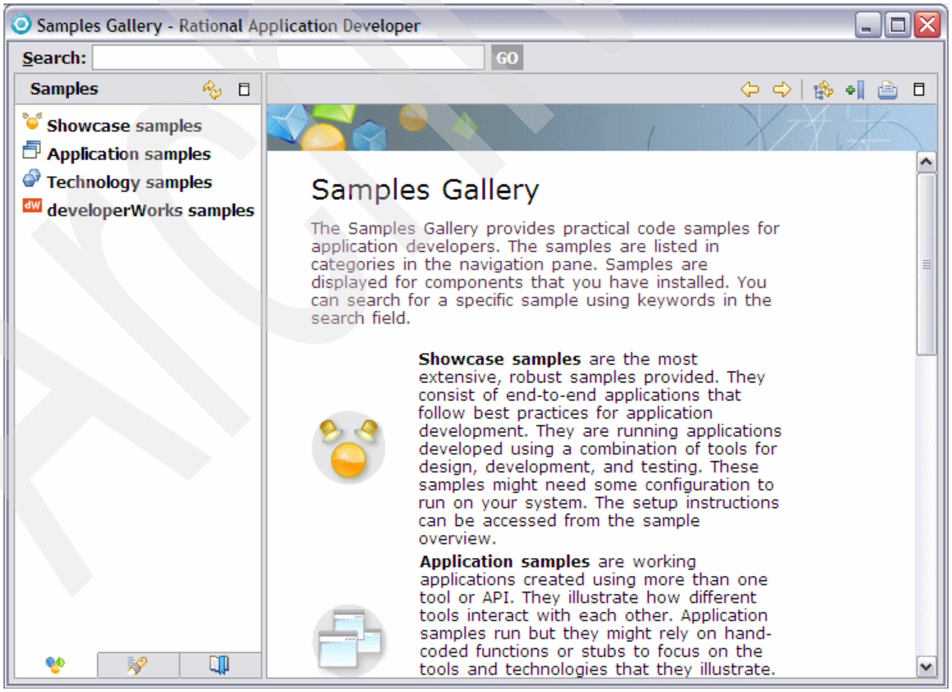
The samples gallery can be accessed via two methods:

- From the **Help** → **Samples Gallery** option from the main menu.
- On the welcome screen presented when a new workspace is started, you can click the **Samples** icon (a grey circle containing a yellow ball, blue cube, and green pyramid) , followed by clicking **Launch the Samples Gallery**.

Samples gallery

The samples gallery lists all the samples available in Rational Application Developer. At the time of writing, this included 26 samples applications within Application Developer and 46 links to tutorial topics within IBM developer works.

The samples can be selected from a hierarchical list in the left-hand pane (Figure 5-10).



The Samples Gallery provides practical code samples for application developers. The samples are listed in categories in the navigation pane. Samples are displayed for components that you have installed. You can search for a specific sample using keywords in the search field.

Showcase samples are the most extensive, robust samples provided. They consist of end-to-end applications that follow best practices for application development. They are running applications developed using a combination of tools for design, development, and testing. These samples might need some configuration to run on your system. The setup instructions can be accessed from the sample overview.

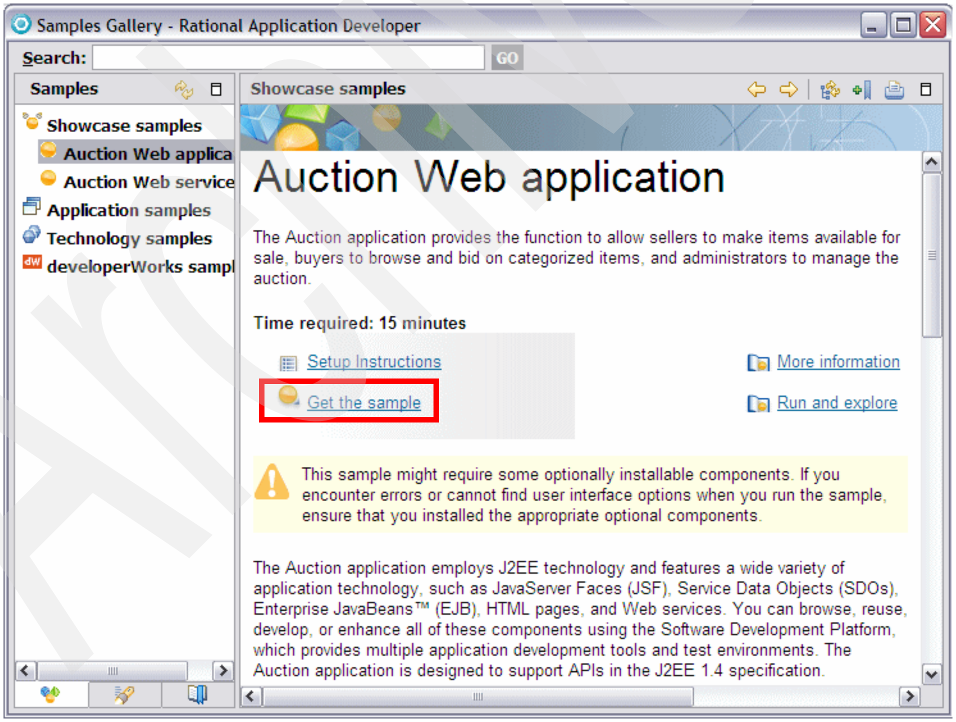
Application samples are working applications created using more than one tool or API. They illustrate how different tools interact with each other. Application samples run but they might rely on hand-coded functions or stubs to focus on the tools and technologies that they illustrate.

Figure 5-10 Samples Gallery

The samples are arranged in four main categories:

- **Showcase samples**—The most extensive samples provided. These contain complete multi-tier, end-to-end applications that follow best practices for application development.
- **Application samples**—Applications created using more than one tool or API, showing how different tools within Rational Application Developer interact with each other.
- **Technology samples**—Smaller, code-based samples that focus on a single tool or API.
- **developerWorks samples**—These samples are linked to the IBM developerWorks Web site and contain the latest samples published on this site. An internet connection is required to access these samples.

For example, the Auction Web application is one of the Web application Showcase samples. The front page for this sample (Figure 5-11) provides an introduction and links to setting up the sample, getting the sample code, running the sample, and references for further information.



Auction Web application

The Auction application provides the function to allow sellers to make items available for sale, buyers to browse and bid on categorized items, and administrators to manage the auction.

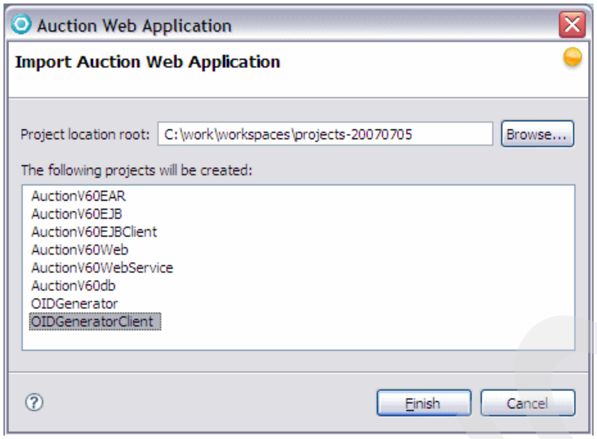
Time required: 15 minutes

This sample might require some optionally installable components. If you encounter errors or cannot find user interface options when you run the sample, ensure that you installed the appropriate optional components.

The Auction application employs J2EE technology and features a wide variety of application technology, such as JavaServer Faces (JSF), Service Data Objects (SDOs), Enterprise JavaBeans™ (EJB), HTML pages, and Web services. You can browse, reuse, develop, or enhance all of these components using the Software Development Platform, which provides multiple application development tools and test environments. The Auction application is designed to support APIs in the J2EE 1.4 specification.

Figure 5-11 Auction Web application sample

The **Get the sample** link shows the user the sample projects that can be imported to the current workspace (Figure 5-12).



Project location root: C:\work\workspaces\projects-20070705

The following projects will be created:

AuctionV60EAR
AuctionV60EJB
AuctionV60EJBClient
AuctionV60Web
AuctionV60WebService
AuctionV60db
OIDGenerator
OIDGeneratorClient

Figure 5-12 Import Auction Web application dialog

Click **Finish** to import the sample projects (in this case eight projects) and build them. From here the user can run, modify, and experiment with the imported code as required.

Example Projects wizard

In addition to the samples gallery, Application Developer also contains a number of example projects that can be added to the workspace through the **New Project wizard**, and selecting **New** → **Project** → **Examples**, then selecting one from the list (Figure 5-13).

For example, these example projects show model solutions for application logging and XML processing. Running the wizard adds the example project to the workspace and also displays an entry from Application Developer help describing the sample.

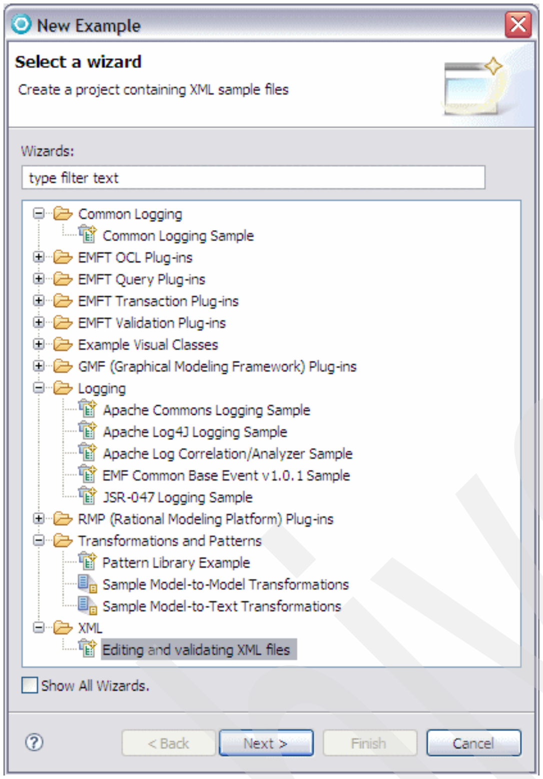

New Example
Select a wizard
Create a project containing XML sample files
Wizards:
type filter text
Common Logging
Common Logging Sample
EMFT OCL Plug-ins
EMFT Query Plug-ins
EMFT Transaction Plug-ins
EMFT Validation Plug-ins
Example Visual Classes
GMF (Graphical Modeling Framework) Plug-ins
Logging
Apache Commons Logging Sample
Apache Log4J Logging Sample
Apache Log Correlation/Analyzer Sample
EMF Common Base Event v1.0.1 Sample
JSR-047 Logging Sample
RMP (Rational Modeling Platform) Plug-ins
Transformations and Patterns
Pattern Library Example
Sample Model-to-Model Transformations
Sample Model-to-Text Transformations
XML
Editing and validating XML files
□ Show All Wizards.
< Back
Next >
Finish
Cancel

Figure 5-13 Example projects

Summary

In this chapter we discussed the main types of project within Application Developer, in particular, those used in the development of J2EE applications. We also presented the basic features for handling projects within the Application Developer workspace, and provided an introduction to the samples gallery that is supplied with Application Developer.

All development activities within Application Developer are based on this project structure, which comes from the Eclipse framework. In the remaining chapters of this book we discuss different types of projects in Application Developer and the specific features available in each project type for building different types of applications.

Part 2

Develop applications

In this part of the book, we describe the tooling and technologies provided by Application Developer to develop applications using Java, databases, XML, JSPs, servlets, Struts, JavaServer Faces, Service Data Objects, Enterpise Genration Language (EGL), Enterprise JavaBeans (EJB), Web services, and Portal Server.

> **Note:** The sample code for all the applications developed in this part is available for download at:
>
> `ftp://www.redbooks.ibm.com/redbooks/SG247501`
>
> Refer to Appendix B, “Additional material” on page 1307 for instructions.

6

RUP and UML

In this chapter we illustrate how a software development project can be assisted by using the Rational Unified Process (or simply RUP) tooling provided within Rational Application Developer. In addition, we demonstrate how the visual UML tooling included in Application Developer can be used within the context of RUP with an emphasis on the implementation phase.

The chapter is organized into the following sections:

- Overview
- Rational Unified Process
- Constructing and visualizing applications with UML
- Working with UML class diagrams
- Describing interactions with UML sequence diagrams
- More information on RUP and UML

Overview

Without having a proper software development process in place that tells an individual practitioner who is responsible for doing what and how and at which time of the project life cycle, there is a high risk of project failure. This is true not only for project managers and architects, but also for developers who are responsible to implement a given solution.

The Rational Unified Process (RUP) is a configurable process framework that provides a number of process artifacts from which a particular development process can be built. It is based on a set of building blocks, or content elements, describing what is to be produced, the necessary skills required, and the step-by-step explanation and guidance describing how specific development goals are achieved. The Rational Process Advisor is a feature provided by the Rational Software Delivery Platform that integrates the RUP seamlessly within the development environment.

The Rational Unified Process focuses on the development of working software rather than the production of a large amount of paper documents. Therefore, it leverages the Unified Modelling Language (UML), an industry standard language that allows you to describe and communicate requirements, architectures, and designs. To enable developers to leverage UML to visually develop and represent artifacts of Java Enterprise JavaBeans and Web services applications, Rational Application Developer provides a customizable UML 2.1 based modelling tool that integrates tightly in the development environment. In the second part of this chapter, we focus on this tool.

Rational Unified Process

A software development process describes an approach to build, deploy, and maintain software. It gives an answer to the question: *who* is doing *what* and *when*.

The Rational Unified Process (RUP) is a software development process framework, a general process description that can and should be refined into a more detailed process description for an organization or software development project. It describes a set of tasks, artifacts (or work products) and responsibilities that can be applied to a software development project to ensure the production of high quality software that meets the client’s requirements within a predictable schedule and budget.

The Rational Unified Process combines the following commonly accepted six best practices into a cohesive and well documented process description:

- Develop iteratively
- Manage requirements
- Use component architectures
- Model visually
- Continuously verify quality
- Manage change

As shown in Figure 6-1, the Rational Unified Process can be described along two dimensions that reflect the dynamic and static structure of the process:

- The horizontal axis represents the dynamic structure and shows the life cycle aspects of a process as time goes by. It is described in terms of phases and iterations.
- The vertical axis represents the static structure and content respectively. It shows the individual disciplines, which logically group the process content.

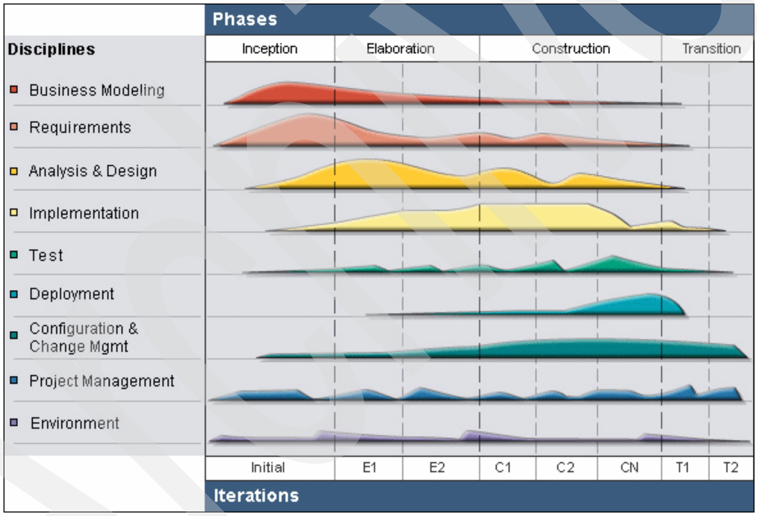


Figure 6-1 Overview of the Rational Unified Process

RUP divides the life cycle of a project in one or more individual cycles that focus on the generation of successive releases of a system (for example, version 1.0 and version 1.1). Each cycle consists of four successive phases: Inception, Elaboration, Construction, and Transition. Each of these phases has a special meaning and concludes with a well defined project milestone. A phase itself is further broken down in several individual iterations, each of them producing some kind of working software.

The static content of the process is based on a set of building blocks, or content elements, describing the individual artifacts to be produced (the artifacts themselves, their characteristics, and standards for creating them), the different roles, including the required skills and their responsibilities within a project, and tasks that provide a step-by-step explanation of how specific development goals are achieved. Related tasks and responsibilities are grouped together and categorized by certain disciplines that are related to a *major area of concern* within the overall project. Every discipline defines a standard way or workflow of doing the work it categorizes. The individual tasks from the disciplines are emphasized depending upon whether the iteration is early or late in the project.

The Rational Unified Process is not only a process framework that can be used to organize and structure a software development process. Beside a complete description of the process itself, it provides:

- Guidelines for all team members, and all portions of the software life cycle. Guidance is provided for both the high-level thought process, as well as for more tedious day to day activities. The guidance is published in HTML form for easy platform independent access on your desktop.
- Tool mentors, which provide additional guidance when working with any of the software development tools offered by IBM Rational Software, such as Rational Application Developer for software development and Rational ClearCase for configuration management.
- Templates and examples for all major process artifacts.

Rational Application Developer facilitates the use of the Rational Unified Process by the Process Advisor feature. This feature provides a seamless integration of a development process within a development tool. It enabled the development team to work with a common development process that is configured for a practitioner’s specific environment. This feature enables the practitioner to obtain context sensitive guidance, to use process elements such as templates and tool mentors, to browse and manage process views, and to manage process preferences without leaving his or her development environment.

If you want to use the Process Advisor feature, either you must select it during the normal product installation, or you must update the installed package using the IBM Installation Manager.

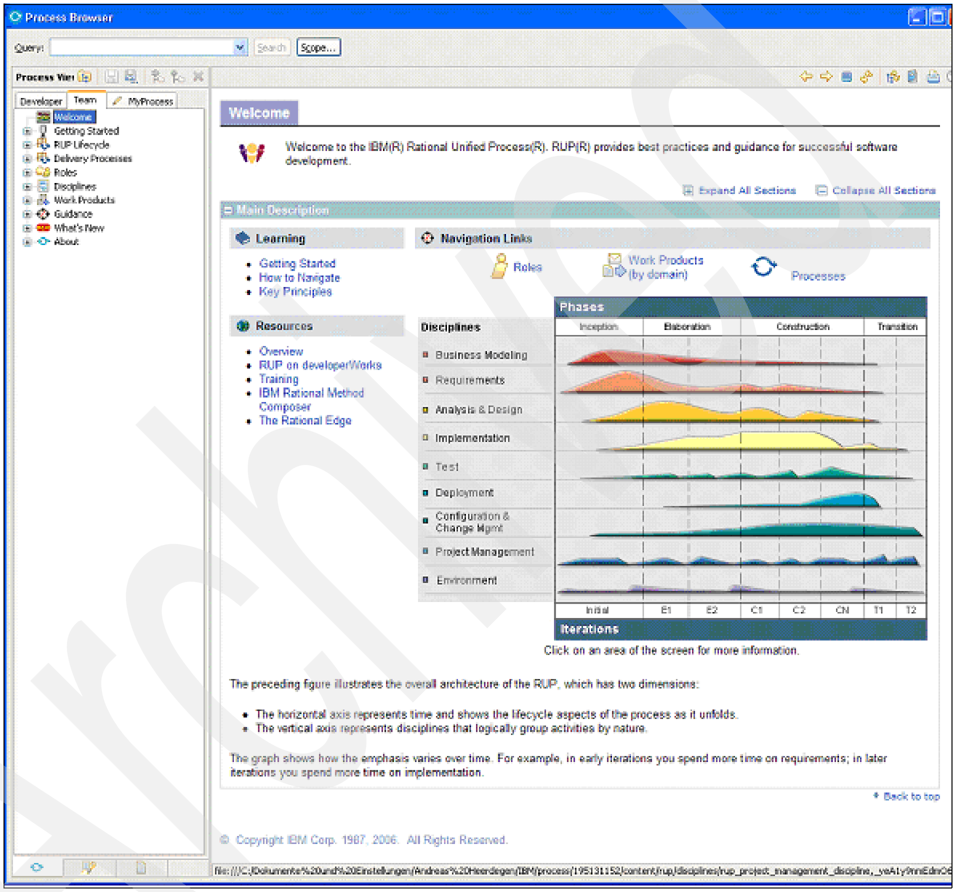
The Process Advisor consists of the four elements:

- IBM Rational Process Browser
- Process Advisor itself
- Process content
- Process preferences

Process Browser

The Process Browser window displays the full set of process content from the installed process configuration and provides navigation to topics with the use of three tabs: *Process Views*, *Search Results* and *Index* (Figure 6-2).

To launch the Process Browser, select **Help** → **Process Browser**.



Welcome to the IBM(R) Rational Unified Process(R). RUP(R) provides best practices and guidance for successful software development.

Click on an area of the screen for more information.

The preceding figure illustrates the overall architecture of the RUP, which has two dimensions:

- The horizontal axis represents time and shows the lifecycle aspects of the process as it unfolds.
- The vertical axis represents disciplines that logically group activities by nature.

The graph shows how the emphasis varies over time. For example, in early iterations you spend more time on requirements; in later iterations you spend more time on implementation.

Back to top
© Copyright IBM Corp. 1987, 2006. All Rights Reserved.
file:///C:/Dokumente%20und%20Einstellungen/Andreas%20Heerdegen/IBM/process/195131152/content/rup/disciplines/rup_project_management_discipline,_yeA1y9nnEdmO6

Figure 6-2 Process Browser

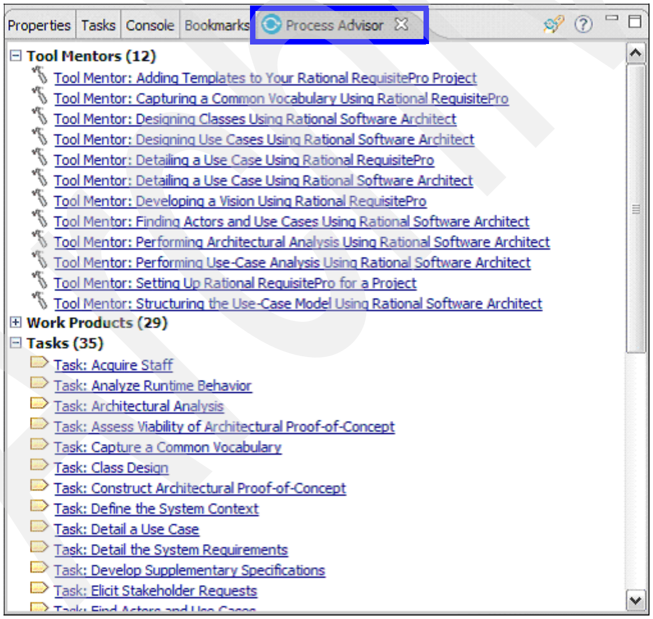
The **Process Views** topic contains several individual process views, each represented by its own tabbed pane. A process view is the hierarchical set of process elements represented in and displayed by the process content tree in the view pane, and associated with a particular role or major category. Process views are used to group and customize the various elements of the installed process configuration including templates, guidelines and examples.

The default process configuration provides two predefined views: Team and Developer. While the **Team** view contains the full method content, the **Developer** view provides that subset of elements that is suitable for developers.

If you want to adapt this configuration to your own requirements, you can create one or more new process views and customize them by using the toolbar buttons at the top of the window. Select a view you want to start with and select **Save Tree As** to create your own copy. This view can then be modified by adding nodes or selecting the desired information for displaying. The Tutorials Gallery contains the tutorial *Customize a process view* that provides a more detailed description of this task.

Process Advisor

The Process Advisor provides real time dynamic process guidance on selected UML diagram elements directly from within Rational Application Developer. Unlike the Process Browser it is a view that runs inside the development tool. After you select a certain context in the tool, for example, a class in an UML class diagram, the Process Advisor view is populated automatically with the process elements that are related to that context (Figure 6-3).



⊟ Tool Mentors (12)
- Tool Mentor: Adding Templates to Your Rational RequisitePro Project
- Tool Mentor: Capturing a Common Vocabulary Using Rational RequisitePro
- Tool Mentor: Designing Classes Using Rational Software Architect
- Tool Mentor: Designing Use Cases Using Rational Software Architect
- Tool Mentor: Detailing a Use Case Using Rational RequisitePro
- Tool Mentor: Detailing a Use Case Using Rational Software Architect
- Tool Mentor: Developing a Vision Using Rational RequisitePro
- Tool Mentor: Finding Actors and Use Cases Using Rational Software Architect
- Tool Mentor: Performing Architectural Analysis Using Rational Software Architect
- Tool Mentor: Performing Use-Case Analysis Using Rational Software Architect
- Tool Mentor: Setting Up Rational RequisitePro for a Project
- Tool Mentor: Structuring the Use-Case Model Using Rational Software Architect

⊞ Work Products (29)

⊟ Tasks (35)
- Task: Acquire Staff
- Task: Analyze Runtime Behavior
- Task: Architectural Analysis
- Task: Assess Viability of Architectural Proof-of-Concept
- Task: Capture a Common Vocabulary
- Task: Class Design
- Task: Construct Architectural Proof-of-Concept
- Task: Define the System Context
- Task: Detail a Use Case
- Task: Detail the System Requirements
- Task: Develop Supplementary Specifications
- Task: Elicit Stakeholder Requests
- Task: Find Actors and Use Cases

Figure 6-3 Process Advisor displaying method content for a selected context

If you want to see any process related information on a selected diagram element, first select **Help** → **Process Advisor**. This opens the Process Advisor view. Then open a UML diagram and select any element in this diagram. Based on your selection and your preference settings (see Process preferences) the tool displays the available content in the Process Advisor view. Now you can click on a link you are interested in. The Process Browser window opens with the search results displayed in the left pane, and the topic content displayed in the right pane. The tool automatically updates the search results in the Process Advisor view when you open or select a new UML diagram or you select any other element in an UML diagram.

Another important tool to search the current process configuration for more information is Process Search (Figure 6-4). It provides search capabilities with advanced filtering and a tight integration with the Eclipse search. You can access this tool either by clicking **Process Search** in the Process Advisor toolbar or directly from the top menu in Rational Application Developer (select **Search** → **Search**. Then select the **Process Search** tab).

Process Search allows you to customize the search results. By selecting the appropriate check boxes you can specify the relevant topics you want to include in the results.

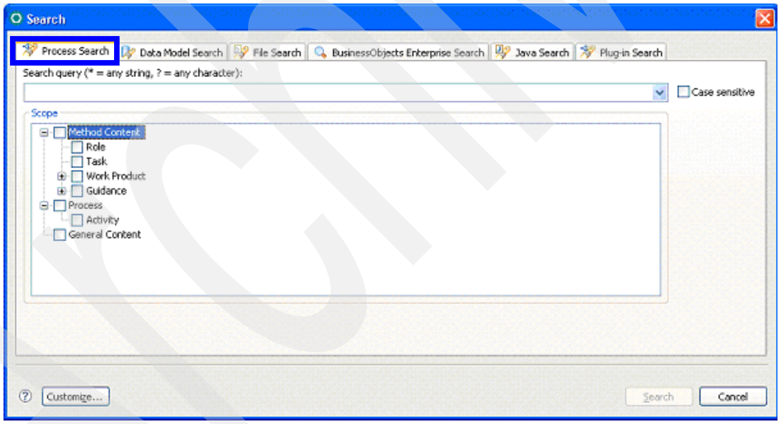
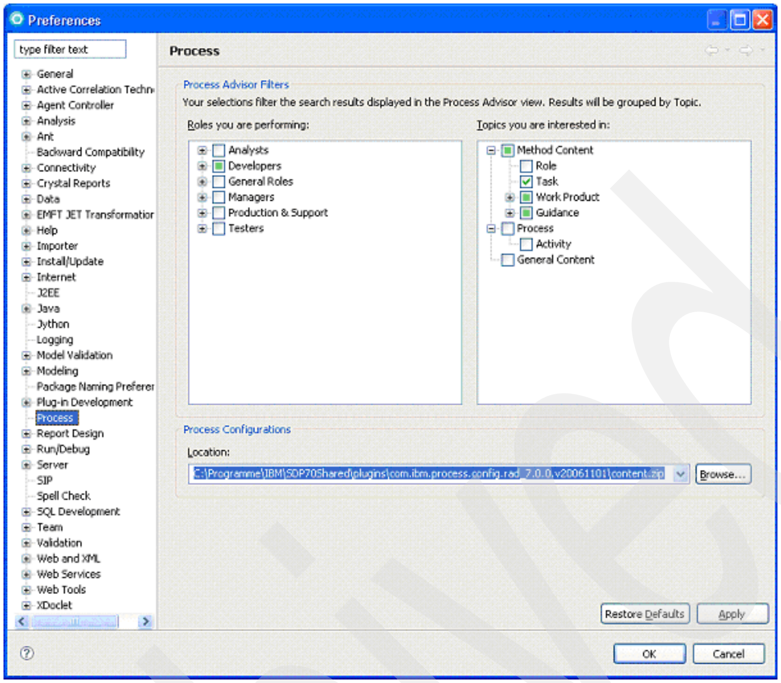


Figure 6-4 Process Search

Process preferences

The Process Advisor feature allows you to select different process configurations or to set content filtering options for dynamic searches displayed in the Process Advisor through the Process page of the Preferences window (see Figure 6-5). The Process preferences page is accessible from within Rational Application Developer by selecting **Window** → **Preferences**.



Process

Process Advisor Filters

Your selections filter the search results displayed in the Process Advisor view. Results will be grouped by Topic.

Roles you are performing:

- □ Analysts
- ▣ Developers
- □ General Roles
- □ Managers
- □ Production & Support
- □ Testers

Topics you are interested in:

- ▣ Method Content
  - □ Role
  - ☑ Task
  - ▣ Work Product
  - ▣ Guidance
- □ Process
  - □ Activity
- □ General Content

Process Configurations

Location: C:\Programme\IBM\SDP70Shared\plugins\com.ibm.process.config.rad_7.0.0.v20061101\content.zip Browse...

Figure 6-5 Process preferences

Rational Application Developer provides a default process configuration that focuses on requirements of application developers. This process configuration is a subset of the full process information available in the standard configuration of the RUP methodology.

In addition to this default process configuration, you can use specific process configurations that you create and publish by using the IBM Rational Method Composer, and then point to these process configurations with the process advisor. You can do this in the preferences page shown in Figure 6-5. Therefore you can simply publish a process configuration in Rational Method Composer and use the **Browse** button to point the Process Advisor to the published process configuration.

Another essential part of the Process Advisor feature is the ability to select filters to determine what context based content will appear in the Process Advisor view. On the process preferences page you can select the roles and topics you are interested in by selecting the appropriate check boxes.

Constructing and visualizing applications with UML

Rational Application Developer provides UML visual editing support to simplify the development of complex Java applications. Developers can create and modify Java classes and interfaces visually, using Class diagrams. They can also review the structure of an application by viewing relationships between various Java elements using Browse and Topic diagrams. Developers can easily design their application and make changes without having to go back and forth between the model and code.

The code visualization capabilities provide several diagrams which enable developers to view existing code from different perspectives. Unlike the diagrams offered in Rational Software Modeler or Rational Software Architect, these are visualizations only of actual code. Any changes to the diagrams directly affect the underlying code. So, true UML 2.1 modelling is not possible using Rational Application Developer. The UML 2.1 notation is used in these diagrams as a way to visualize and understand the code.

Visual editing offers developers the ability to build code without explicitly typing the code in a text editor. A palette is used to drag and drop different components such as Java classes onto a diagram. They can then further be edited visually such as adding methods and attributes or defining relations between them.

Rational Application Developer offers the following types of UML visual diagrams:

- **Class diagrams** show some or all of the components or elements in an application. You can use class diagrams to visually represent and develop the structures and relationships for Java classes and interfaces. You can create your own context to understand, collaborate, and develop an application by using a subset of its classes. You can also develop Java elements directly from class diagrams.
- **Browse diagrams** are temporary diagrams in which you can explore the details of an application and its underlying elements and relationships. A browse diagram provides a view of a context element that you specify and is similar in functionality to a Web browser. You cannot add or modify the individual diagram elements or save a browse diagram in an application. However, you can convert a browse diagram to a UML class diagram or save it in an image file that captures the inheritance hierarchy, which you can send in an e-mail message or publish on a Web site.

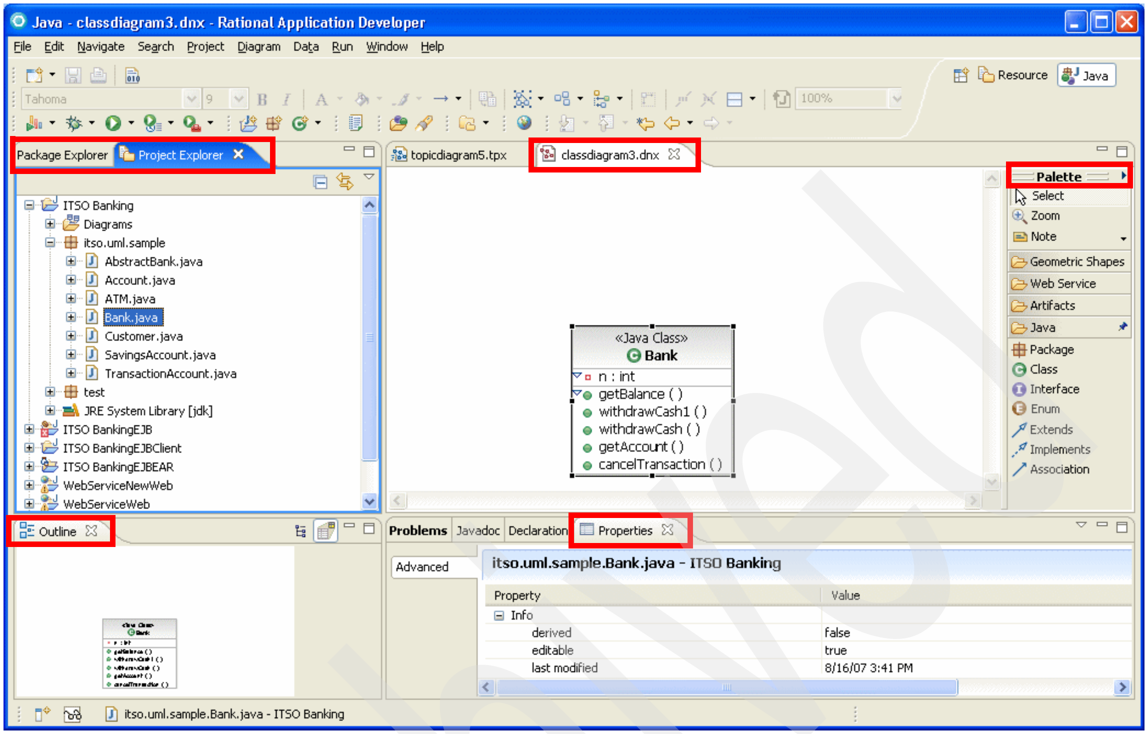
- **Topic diagrams** are non-editable diagrams that provide a quick way to show existing relationships between elements in an application. These diagrams query the current state of the application to show relationships between classes and the program hierarchy. You can customize the underlying query, open multiple topic diagrams at the same time and save them away for further use.
- **Sequence diagrams** show the interactions between objects with focus on the sequence (in time) of the messages that are exchanged between them.
- **Static method sequence diagrams** are non-editable diagrams that visually represent and explore the chronological sequence of messages between instances of Java elements in an interaction. You can create a static sequence diagram view of a method (operation), including signatures, in Java classes and interfaces to illustrate the logic inside that operation.

All of these diagrams help developers analyze and document code. To provide further documentation you can generate Javadoc HTML documentation that contains UML diagram images to provide more information about the source code.

The visualization tool is applicable not only for Java classes but also for other types of artifacts, such as Web services or Enterprise JavaBeans. Rational Application Developer also supports data visualization using Unified Modeling Language or Information Engineering notation.

Figure 6-6 provides an overview of a workspace that you might see when using the UML visualization capabilities:

- The center area is the UML editor. This editor is used to display and modify the different elements of the model.
- Built into the editor is a palette that is used to drag and drop artifacts into the editor work area. The items that appear in the palette are specific to the type of project that is being visualized. The palette is only available when the diagram is editable.
- The Outline view enables you to see the area of the diagram that you are zoomed into the editor area. You can switch this view to show the different elements that build up the current model.
- The Properties view enables you to review or change any property that is related to a selected diagram or a contained element.
- Finally, you can drag and drop project artifacts from the Project Explorer or Package Explorer view directly into the editor to add these items to the diagram.



Property	Value
Info	
derived	false
editable	true
last modified	8/16/07 3:41 PM

Figure 6-6 Using the UML visualization capabilities

Unified Modeling Language

A large part of the Rational Unified Process is about developing and maintaining models of the application under development. A model is a description of a system from a particular perspective, omitting irrelevant details so that the characteristics of interest are seen more clearly. Therefore models are useful for understanding problems, communicating with everyone involved with the project, preparing documentation, and designing applications. Modeling promotes better understanding of requirements, cleaner designs, and more maintainable applications.

The Unified Modeling Language (UML) is a standardized language for modelling the different aspects of an application. You can use this language to visualize, specify, construct and document the different artifacts of an application. The vocabulary of this language encompasses three kinds of building blocks:

- Things
- Relationships
- Diagrams

Things

Things are the basic elements of a model. The UML defines four kinds of things:

- **Structural** things, such as interfaces, classes, components, or uses cases, are the nouns of a model. They are used to describe the static parts of a model.
- **Behavioral** things are the dynamic parts of UML models representing the verbs of the language. For example, an interaction is a behavior that comprises a set of messages exchanged between different objects within a particular context to accomplish a specific goal.
- **Grouping** things are used to organize a model. They group semantically related things together to a certain context. The only grouping thing provided by the UML is a package.
- **Annotational** things provide further explanation about any element in a model. These are the comments in the UML.

Relationships

There are five kinds of relationships in the UML that describe how one element is related to another:

- **Dependency** is a semantic relationship between two things, where a change to one class might affect another class. For example, a `Bank` class depends on an `Account` class because it is used as parameter for an operation.
- An **association** is a structural relationship that describes a connection between objects so that you can navigate between them. Such a relationship maps to the declaration of class members in a class. For example, a `Customer` class might have a single association to an `Account` class, which indicates that each `Account` instance is owned by one `Customer` instance.
- The **extends** relationship describes the case when one thing is a specialization or concretion of another more general thing. Such a relationship maps to the `extends` keyword in Java. For example, a `SavingsAccount` class that extends an `Account` class provides a more specific behavior than its parent.
- The **implements** relationship maps to the `implements` keyword in Java. Such a relationship represents a class that realizes operations (behavior) that is defined by an interface.
- The **owned** element association represents classes that are owned by a package. This construct is primarily used for implementation reasons and for hiding information

Diagrams

A diagram is a graphical presentation of a set of elements, most often rendered as a connected graph of things and their relationships. The UML provides thirteen types of diagrams that let the users capture, communicate, and document all aspects of an application using a standard graphical notation that is recognized by many different stakeholders.

The individual diagrams can be categorized into three groups, each of them representing a different view of an application.

- **Structural** diagrams show the building blocks of an application, characteristics that do not change over time. For this purpose they use objects, attributes, operations, and relationships. Examples are component diagrams, class diagrams and deployment diagrams.
- **Dynamic behavior** diagrams emphasizes how an application responds to requests or otherwise evolves over time by showing collaborations among objects and changes to the internal states of objects. Examples are use case diagrams and activity diagrams.
- **Interaction** diagrams are a subset of behavior diagrams that focus on the exchange of messages within a collaboration (that is, a group of cooperating objects) to accomplish a goal. Examples are sequence diagrams.

Working with UML class diagrams

A UML class diagram is a concretion of a structural diagram. It shows some or all of the components or elements in an application and the relationships between them (including inheritance, aggregation, and association). You can use class diagrams to visually represent and develop artifacts of Java and Enterprise JavaBean applications as well as Web Service Description Language (WSDL) Version 1.1 elements, such as WSDL services, port types, and messages. You can also use class diagrams to develop and represent the relationships between Java classes and interfaces or between Enterprise JavaBeans.

The content of a class diagram is stored in a file with a .dnx extension. The corresponding diagram editor consists of a panel displaying the different elements and a tool palette that contains individual tools that can be used to create new UML elements and add them to the current diagram.

Creating class diagrams

A new class diagram is created by the Create Class Diagram wizard. You can launch this wizard directly from the top menu in Rational Application Developer (select **File** → **New** → **Other** → **Modeling** → **Class Diagram**) or within the Project Explorer view from the context menu of any resource such as projects or packages (**New** → **Class Diagram**).

Once the wizard is started, enter the name of the file that should contain the content of this diagram and specify the folder where this file should be stored. Then click **Finish**. The new class diagram is created and opened for editing with an associated palette on the right side (Figure 6-7).

Alternatively, you can create a UML class diagram of existing source elements, including packages, classes, interfaces, Enterprise JavaBeans, and data types. In the Project Explorer select the desired source element(s) and then select **Visualize** → **Add to New Diagram File** → **Class Diagram** from the context menu.

Of course it is possible to create more than one class diagram within a folder, each of them depicting different aspects of an application.

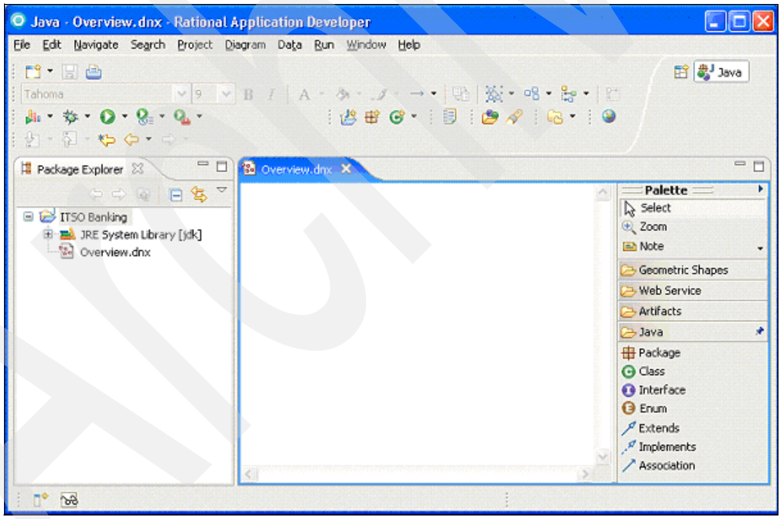


Figure 6-7 Class diagram

Creating, editing, and viewing Java elements in UML class diagrams

A class diagram enables developers to create, edit and delete Java elements such as packages, classes, interfaces, and enum types to visually develop Java application code.

To create a new element, select the desired tool in the tool palette and place it on an empty space inside the class diagram. This launches the appropriate wizard that guides you to create the selected element, both in the diagram and under the project. Alternatively, you can create these elements directly and place them afterwards into the diagram.

As shown in Figure 6-8, an element or more precisely a class is represented by a rectilinear shape with three different compartments.

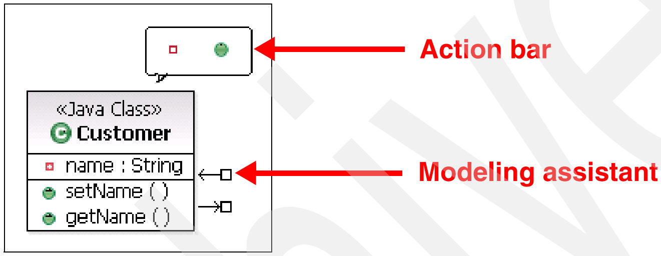


Figure 6-8 Action bar and modeling assistant for Java elements

- The upper compartment (name compartment) contains the class name and a stereotype. A stereotype is a string, surrounded by guillemets or angle quotes (`<<sterotypestring>>`), that describes the UML type of the class.
- The middle compartment (attribute compartment) contains the attributes.
- The lower compartment (operation compartment) contains the operations.

You can show or hide individual compartments so that only the compartments you are interested in appear in the shape. Right-click on the shape and select **Filters** → **Show/Hide Compartment**, and a submenu offers you the available options.

When you hover over a Java element that has been visualized on a class diagram, the action bar and the modeling assistant are displayed:

- The **action bar** is an icon-based context menu that provides a quick access to commands that allow you to edit the selected element. Using the buttons on the action bar, you can edit the Java element by adding fields or methods to it. These actions are also available through the Add Java submenu in the element's context menu.

- The **modeling assistant** enables you to create and view relationships between the selected element and other classifiers such as packages, classes or interfaces. It is available in the form of two arrows, one pointed towards the element and the other pointed away. The arrow pointed towards the element is for incoming relationships. Thus, when creating a relationship where the selected element is the target, you would use the incoming arrow. Similarly, the arrow pointed away from the element is for outgoing relationships.

To create a relationship from one Java element to another element:

- First, select the source element so that the modeling assistant is available and click on the small box at the end of the outgoing arrow.
- Then, drag the resulting connector and drop it on the desired element or on an empty space inside the diagram if you want to create a new element.
- Finally, select the type of relationship you want to create from the context menu that appears when the connector is dropped (Figure 6-9).

You can create incoming relationships in the same way by using the incoming arrow. Alternatively, you can select the desired relationship in the tool palette and place it on the individual elements.

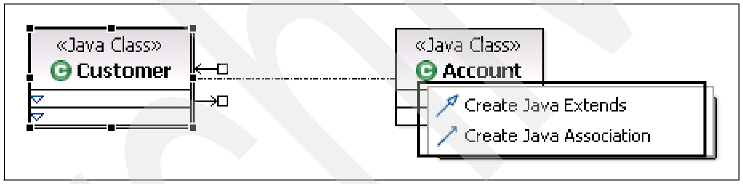


Figure 6-9 Using the Modeling Assistant to create a relationship

The modeling assistant also allows you to view related elements that are based on a specific relationship. Double-click the small box at the end of the outgoing or incoming arrow and select the desired relationship from the resulting context menu (Figure 6-10). This is equivalent to selecting **Filter** → **Show Related Elements** from the element's context menu, then selecting a single relationship and restricting it to a single level for either incoming or outgoing relationships.

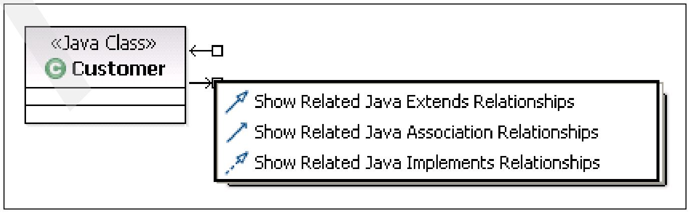
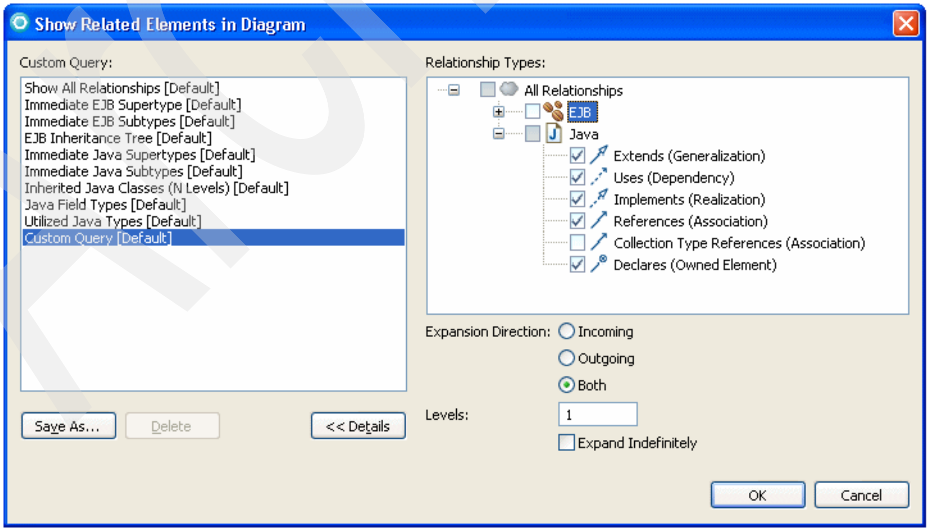

«Java Class»
Customer
Show Related Java Extends Relationships
Show Related Java Association Relationships
Show Related Java Implements Relationships

Figure 6-10 Viewing related Java elements using the Modeling Assistant

The context menu contains several other options to edit a selected class or to change its appearance:

- **Add Java** allows you to add a new field or method to the selected class (or rather a class or interface to a package). You can also select the appropriate action from the action bar to do this.
- **Delete from Diagram** removes the visual representation of the selected element(s) from the diagram, whereas **Delete from Project** deletes these elements from both the diagram and project permanently. Note, when you delete Java elements from a class diagram, the underlying semantics remain intact.
- **Format** changes the properties of a selected element that govern its appearance and location in a diagram. Modifying these properties only changes the appearance of the element and does not affect any other element that represents the same application element. Some of these properties can be configured globally by the modeling preferences dialog.
- **Filters** allow you to manage the visual representation of classifiers in UML class diagrams. For example, you can show or hide attributes and operations, determine if operation signatures are displayed or specify if the fully qualified names of individual classes are shown.
- **Show Related Elements** is another useful feature that helps developers to query for related elements in a diagram. As shown in Figure 6-11, you are able to select from a set of predefined queries. By clicking **Details** you can view and change the actual relationships, along with other settings related to the selected query.


Show Related Elements in Diagram

Custom Query:

Show All Relationships [Default]
Immediate EJB Supertype [Default]
Immediate EJB Subtypes [Default]
EJB Inheritance Tree [Default]
Immediate Java Supertypes [Default]
Immediate Java Subtypes [Default]
Inherited Java Classes (N Levels) [Default]
Java Field Types [Default]
Utilized Java Types [Default]
Custom Query [Default]

Relationship Types:

- ☐ All Relationships
  - ☐ EJB
  - ☐ Java
    - ☑ Extends (Generalization)
    - ☑ Uses (Dependency)
    - ☑ Implements (Realization)
    - ☑ References (Association)
    - ☐ Collection Type References (Association)
    - ☑ Declares (Owned Element)

Expansion Direction: ○ Incoming ○ Outgoing ● Both

Levels: 1

☐ Expand Indefinitely

Figure 6-11 Show Related elements dialog

- **Refactor** and **Source** provide the same functionality to change and edit the underlying Java code as invoked on the class directly.

Figure 6-12 provides an overview of a class diagram showing different elements and the relationships between them.

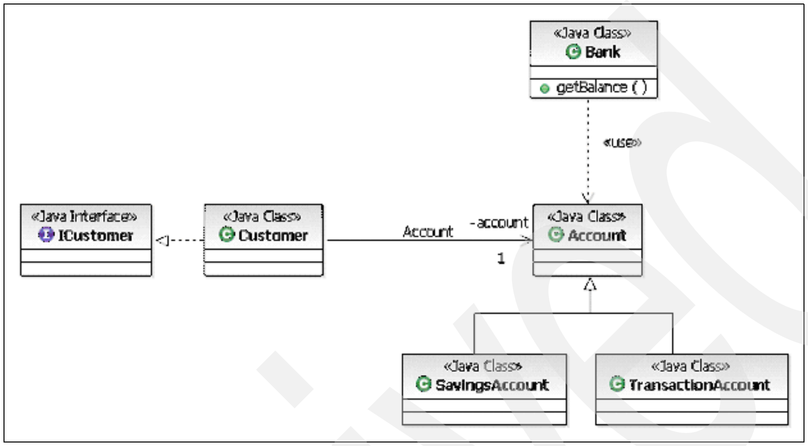


Figure 6-12 Class diagram showing different UML elements

For a class diagram of the ITSO Bank application, refer to Figure 7-12 on page 238.

Creating, editing, and viewing EJBs in UML class diagrams

A class diagram provides developers the capabilities to visually represent and develop Enterprise JavaBeans (EJB) in EJB applications. Developers can create class diagrams and populate them with EJBs and their relationships. They can also use class diagrams to develop new EJBs, create the relationships between them, and add security management concepts.

To use these capabilities, a class diagram must be created within the context of an EJB project. Then you can use the tool palette in the same way as described in the previous section to create and edit EJBs, to develop the relationships between them, and to define a mapping to an underlying relational database table. Alternatively, you can create these elements directly and place them afterwards on the diagram.

Figure 6-13 shows the graphical representation of an EJB and the corresponding EJB module (to visualize an EJB module select its deployment descriptor from the Project Explorer and drop it into the diagram). Because they are more complex than a Java class, EJBs are mapped to UML components, a subtype of a UML class. A component defines its behavior in terms of provided and required interfaces. For an EJB, provided interfaces are those which expose the bean's functionality to a client, such as a home interface or a remote interface. Required interfaces are those which an EJB must offer. The bean class realizes the defined behavior of the EJB.

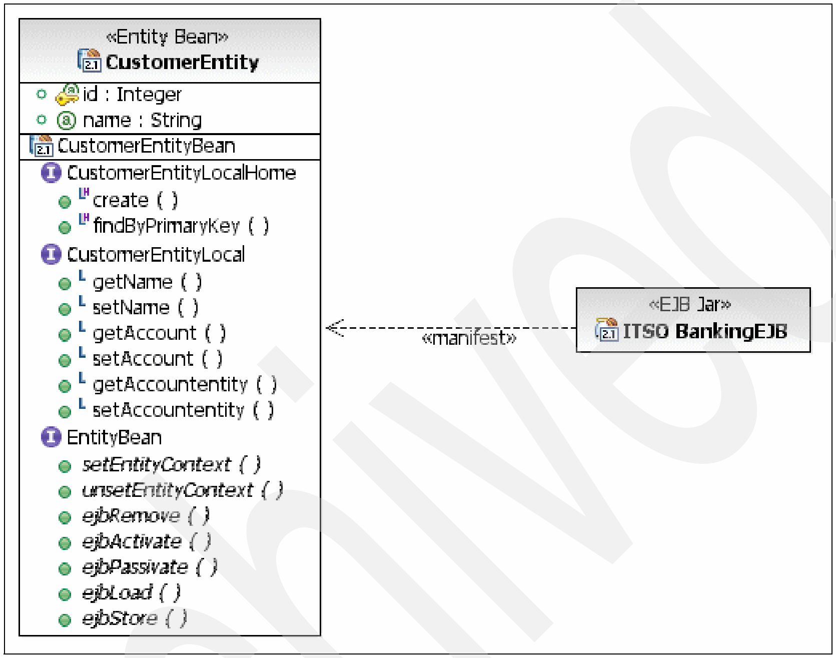


Figure 6-13 Visualization of an Enterprise JavaBean

In Figure 6-13 an Entity Bean is shown with the following compartments displayed: Name, Attribute, Realization, and Required Interfaces.

On the top you can see the Attribute compartment that displays the persistent attributes of that bean. If you want to show or hide any compartment, select **Filters** → **Show/Hide compartments** from the context menu.

Relationships between EJBs

You can use the class diagram to create relationships between different EJBs. From the tool palette choose one of the following EJB relationships and drag it on the desired components. As described previously you can also use the modeling assistant or the EJB deployment descriptor to create these relationships.

- EJB inheritance
- EJB relationship
- Relationships between container-managed persistence (CMP) entity beans

Figure 6-14 shows three EJBs with two different relationships between them. The `BankingFacade` session bean references the `CustomerEntity` entity bean through an EJB local reference, while the two entity beans are connected using a container-managed persistence (CMP) relationship with each other.

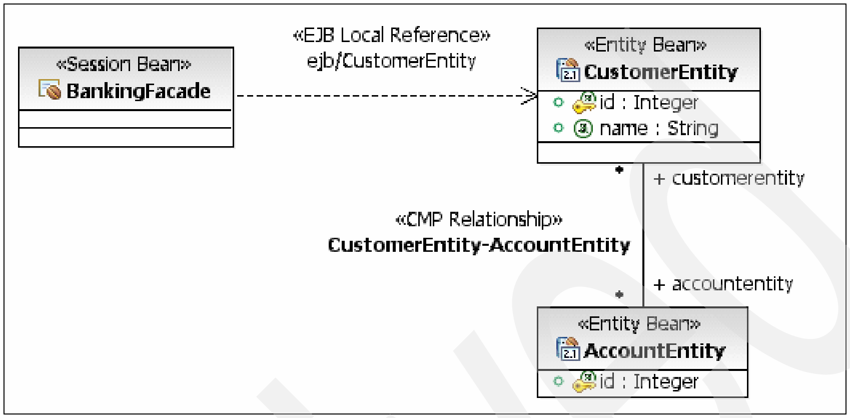
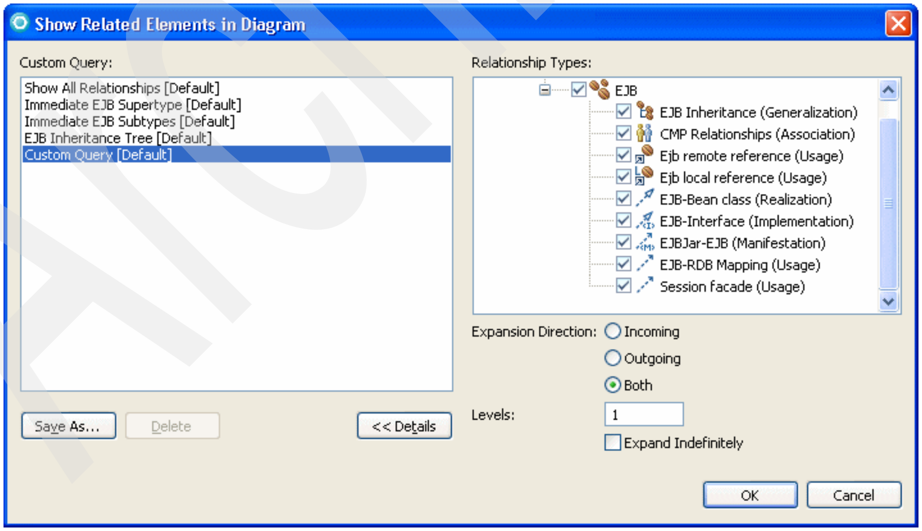


Figure 6-14 Class diagram showing different relationships between EJBs

After selecting an EJB the Show Related Elements feature (**Filters** → **Show Related Elements**) searches and displays related elements based on selection criteria. This feature provides a set of predefined custom queries that can be quickly used to view the related elements (Figure 6-15).


Show Related Elements in Diagram

Custom Query:
- Show All Relationships [Default]
- Immediate EJB Supertype [Default]
- Immediate EJB Subtypes [Default]
- EJB Inheritance Tree [Default]
- Custom Query [Default]

Relationship Types:
- ☑ EJB
  - ☑ EJB Inheritance (Generalization)
  - ☑ CMP Relationships (Association)
  - ☑ Ejb remote reference (Usage)
  - ☑ Ejb local reference (Usage)
  - ☑ EJB-Bean class (Realization)
  - ☑ EJB-Interface (Implementation)
  - ☑ EJBJar-EJB (Manifestation)
  - ☑ EJB-RDB Mapping (Usage)
  - ☑ Session facade (Usage)

Expansion Direction: ○ Incoming ○ Outgoing ● Both

Levels: 1

□ Expand Indefinitely

Figure 6-15 Show Related Elements dialog

By default, the details are collapsed and only the left pane in the dialog is visible. By clicking **Details** the actual relationships along with other settings related to the selected query can be viewed. You can select the different types of relationship that should be included in the query along with the expansion direction. If you select **Incoming**, all elements are shown that are related to the selected element. On the other side, if you want to see all elements that have a relationship to the selected element, select **Outgoing**. Any changes made to the queries can be saved for future use.

From the context menu there are several other options available to edit a selected EJB or to change its appearance. Most of these features have been described previously so the following focuses on topics that are specific to EJBs. Some of these features are exposed as wizards that are also available through the action bar, the tool palette, or the deployment descriptor.

You can use UML class diagrams to manage EJB security visually. This comprises creating and modifying security roles, security role references, run-as security identities, method permissions, and excludes lists.

Security roles

Developing security roles for an Enterprise JavaBean comprises the following tasks: Create security roles, add references to security roles, and link security role references to security roles:

- A security role is created with the help of the Add Security Role wizard. To launch this wizard, right-click an EJB JAR artifact and select **Add EJB** → **Security Role**. Once the wizard is started, enter the name and description for the security role and click **Finish**. As shown in Figure 6-16, the newly created security role is visualized in the diagram as an actor notation of stereotype <<`Role Name`>>, with a dependency of stereotype <<`Security Role`>> to the EJB JAR artifact.

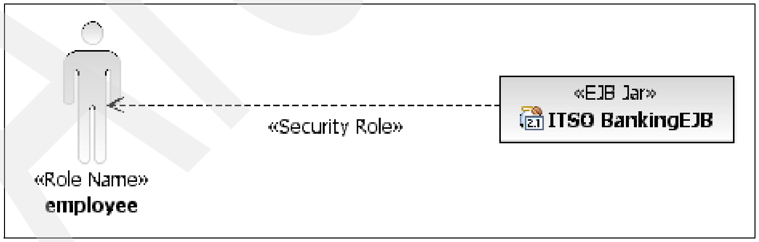


Figure 6-16 Security role

- The next step is to create references to security roles defined for an EJB JAR. In the diagram editor, right-click the desired Enterprise JavaBean, and select **Add EJB** → **Security Role Reference**. In the dialog provide a name for the reference. Optionally you might select a security role to link to. Then click **Finish**. This creates the specified reference. Figure 6-17 shows the security role reference being linked to the appropriate security role. There is also a relationship of stereotype `<<Security Role Reference>>` to the EJB.

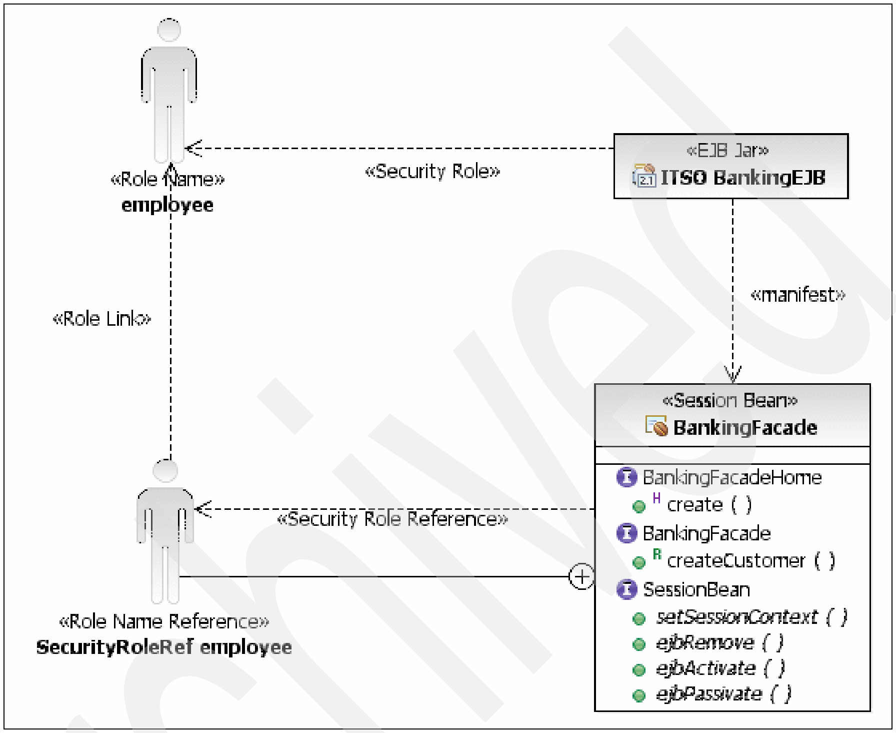
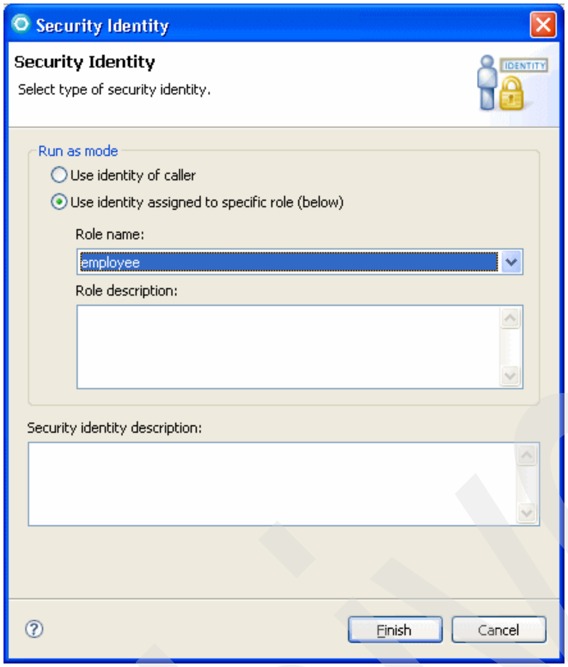


Figure 6-17 A security role with a linked security role reference

- If not already done in the previous step, you must link a security role to a security role reference. In the class diagram, right-click on the actor representing the security role reference, and select **Link to Security Role**. In the following dialog you can add or change the desired role.

Run-as security identities

You are able to directly create run-as security identities for container-managed persistence entity beans in the class editor. Note that this security identity will be applied to all methods defined in the home interfaces. From the context menu select **Create Run-As Identity**. This opens the dialog shown in Figure 6-18, where you can select if either the identity of the caller or the identity assigned to a specific role should be used for authentication and authorization.



Security Identity

Select type of security identity.

Run as mode

○ Use identity of caller

● Use identity assigned to specific role (below)

Role name: employee

Role description:

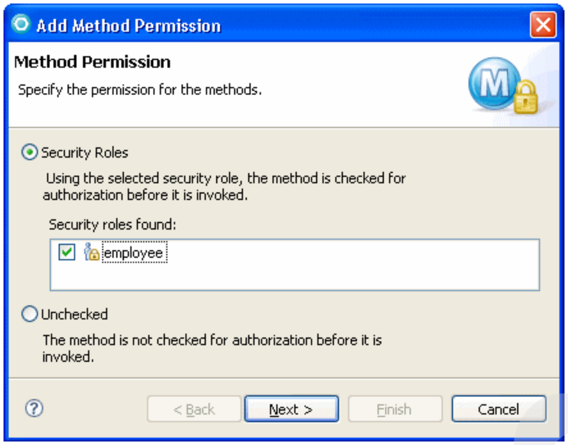
Security identity description:

Figure 6-18 Specifying the security identity

Method permissions

Another aspect of security management that can be visually created and manipulated are method permissions. Once the security roles for an EJB are defined, you can specify the methods of the home and component interface that each security role is allowed to invoke:

- You can add method permissions with the help of the Add Method Permission wizard. Right-click a session or entity bean and select **Add EJB** → **Method Permission**. In the first step, you must specify the permission for the methods. You have the option to select from an existing security role or to allow the methods to be invoked without being checked for authorization (Figure 6-19).



Add Method Permission

Method Permission

Specify the permission for the methods.

● Security Roles

Using the selected security role, the method is checked for authorization before it is invoked.

Security roles found:

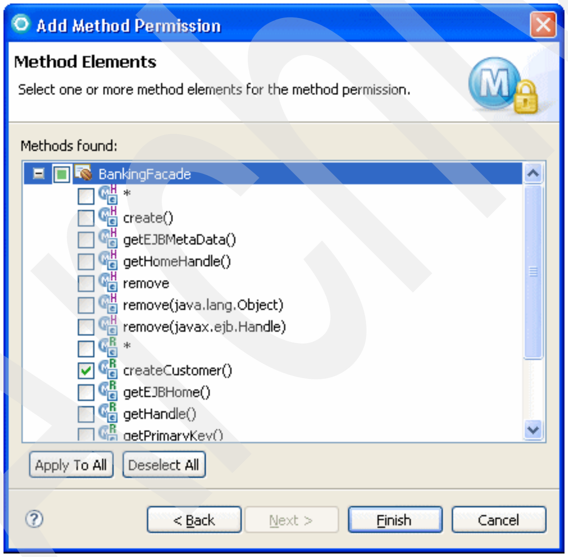
☑ employee

○ Unchecked

The method is not checked for authorization before it is invoked.

Figure 6-19 Adding method permissions dialog (1)

- In the next step, select the methods from the home and component interfaces that each selected security role is allowed to invoke. Then click **Finish** (Figure 6-20).



Add Method Permission

Method Elements

Select one or more method elements for the method permission.

Methods found:

- ☐ BankingFacade
  - ☐ *
  - ☐ create()
  - ☐ getEJBMetaData()
  - ☐ getHomeHandle()
  - ☐ remove
  - ☐ remove(java.lang.Object)
  - ☐ remove(javax.ejb.Handle)
  - ☐ *
  - ☑ createCustomer()
  - ☐ getEJBHome()
  - ☐ getHandle()
  - ☐ getPrimaryKey()

Figure 6-20 Adding method permissions dialog (2)

- Figure 6-21 shows the new method permission being linked to the EJB through a dependency of stereotype <<`Method`>>. If you want to edit a method permission, right-click the desired method permission and select **Edit** → **Edit**, then follow the instructions in the Add Method Permission wizard.

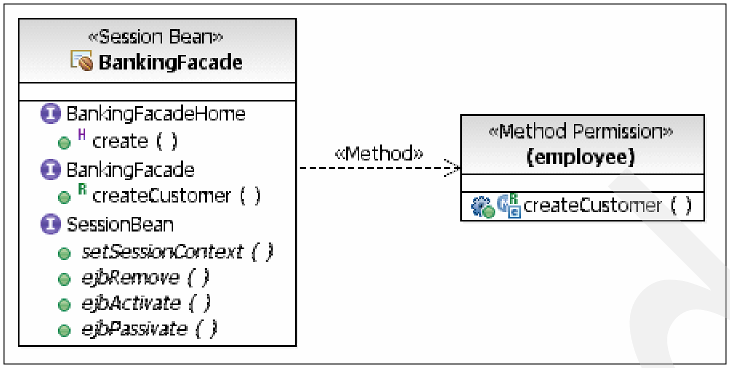
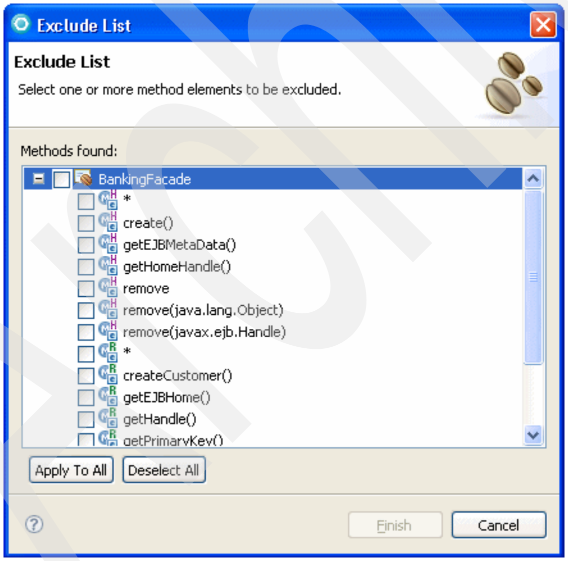


Figure 6-21 Class diagram visualizing a method permission

Exclude lists

Finally, you can specify methods that you want to mark as not callable. In the diagram editor, right-click an EJB session or entity bean and select **Add EJB** → **Exclude List**. In the dialog select the methods to be excluded, and click **Finish** (Figure 6-22). To edit an exclude list, right-click and select **Edit** → **Edit** and follow the instructions in the Exclude List wizard.



Exclude List

Select one or more method elements to be excluded.

Methods found:

- □ BankingFacade
  - □ *
  - □ create()
  - □ getEJBMetaData()
  - □ getHomeHandle()
  - □ remove
  - □ remove(java.lang.Object)
  - □ remove(javax.ejb.Handle)
  - □ *
  - □ createCustomer()
  - □ getEJBHome()
  - □ getHandle()
  - □ getPrimaryKey()

Figure 6-22 Exclude List wizard

Additionally, there are many other features available from the context menu of an Enterprise JavaBean. For example, you can create session bean facades and visualize them on a class diagram, generate mappings between EJBs and relational database tables, and develop custom EJB queries.

For examples of EJB class diagrams refer to Chapter 16, “Develop Web applications using EJBs” on page 719.

Creating, editing, and viewing WSDL elements in UML class diagrams

New to Rational Application Developer 7 is the feature that enables developers to represent and create Web Service Description Language (WSDL) Version 1.1 and XML Schema Definition (XSD) elements using UML class diagrams.

To use this feature, the Web Service Modeling capability must be enabled beforehand. In the preferences dialog (**Window** → **Preferences**) expand **General** → **Capabilities**. In the Capabilities page select **Modeling** and click **Advanced**. In the dialog expand **Web Service Developer** and select **Web Service Development**.

Figure 6-23 shows the graphical representation of a WSDL service. To visualize a service, select its WSDL file from the Project Explorer and drop it on the diagram (alternatively, right-click a WSDL file and select **Visualize** → **Add to New Diagram File** → **Class Diagram**).

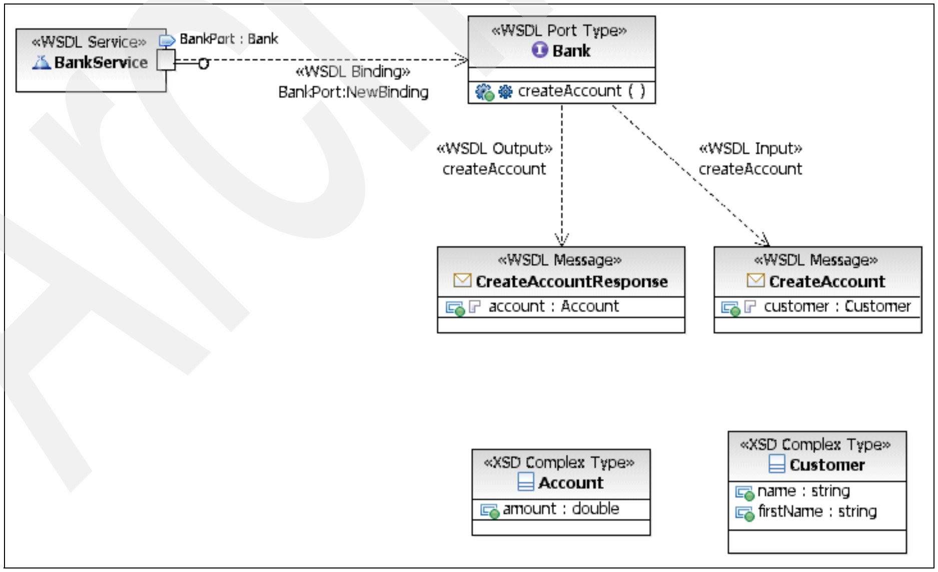


Figure 6-23 Graphical representation of a Web Service

Note that per default the external view of a service is shown. If you want to switch to the compressed view, select the **Filter** submenu and clear **Show External View**. By default, only WSDL services and port types are visually represented in a class diagram. This default setting can be changed by the Modelling page of the Preferences window (see “Class diagram preferences” on page 207).

Like Enterprise JavaBeans, WSDL services are mapped to UML components:

- The individual ports of a service are depicted as small squares on the sides of the component's shape. The functionality provided by a port are exposed by a port type.
- A port type is mapped to an UML interface that is modelled using the *lollipop* notation. In Figure 6-23 you can see the port type explicitly displayed as an interface being linked to its port. This link is realized as a dependency with stereotype <<`WSDL Binding`>>. It describes the binding being used for this port type. A port type references messages that describe that type of data being communicated with clients.
- A message itself consists of one or more parts that are linked to types. Types are defined by XSD elements. Figure 6-23 shows that messages and XSD elements are mapped to UML classes. The different parts of a message are described as attributes.

Creating a WSDL service

This section provides a sample scenario that describes how the UML class diagram editor can be used to create a new Web service from scratch. We use different tools provided by the tool palette to create the various elements of a Web service, such as services, ports, port types, operations, and messages, and to link them together.

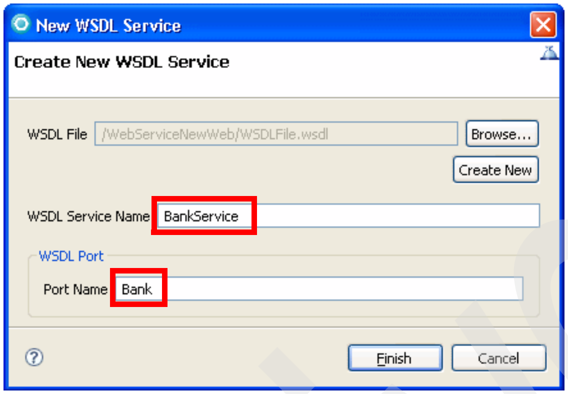
This scenario comprises the following tasks:

- Creating WSDL services
- Adding ports to WSDL services
- Creating WSDL port types and operations
- Creating WSDL messages and parts
- Editing parts and creating XSD types
- Creating bindings between WSDL ports and port types

Creating WSDL services

If you select the WSDL Service element in the tool palette and drop it on an empty space inside the class diagram the New WSDL Service wizard starts to create a new WSDL service along with a port (Figure 6-24).

At first you must specify the WSDL file that will contain the service. You can either click **Browse** to select an existing file or you can click **Create New** to launch the New WSDL File wizard (on the Options page, clear **Create WSDL Skeleton**; you will create these elements later in the next tasks). Finally, provide a name for the service and port and click **Finish**.


New WSDL Service

Create New WSDL Service

WSDL File: /WebServiceNewWeb/WSDLFile.wsdl

WSDL Service Name: BankService

WSDL Port

Port Name: Bank

Figure 6-24 New WSDL Service wizard

The result of this task is shown in Figure 6-25. Like an EJB, a WSDL service is mapped to an UML component with stereotype <<`WSDL Service`>>. The port is depicted as a small square on the right side of the component. Note that the external view of the component is shown. To switch to the compressed view, open the context menu and clear **Show External View** from the Filter submenu.

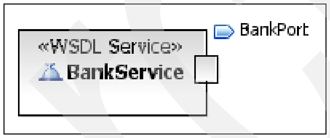


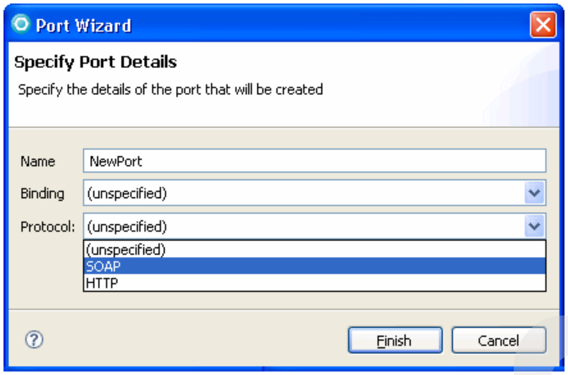
Figure 6-25 Visualization of a WSDL service component

Adding ports to WSDL services

A WSDL service consists of one or more individual ports. A port describes an endpoint of a WSDL service that can be accessed by clients. It contains the properties name, binding, and address. The name property provides a unique name among all ports defined within the enclosing WSDL, the binding property references a specific binding and the address property contains the network address of the port.

To add a port to a WSDL service, right-click the service and select **Add WSDL → Port**. This launches the Port wizard shown in Figure 6-26. Enter the name of the port. Optionally you can specify a binding and a protocol. Then click **Finish**.

Note that this task is not required for our scenario because a port has already been created and added to the service in the previous task.



Specify Port Details

Specify the details of the port that will be created

Name: NewPort

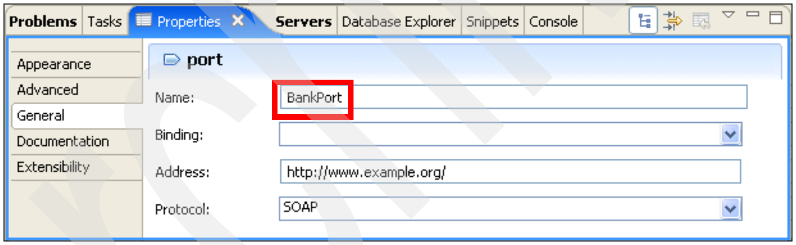
Binding: (unspecified)

Protocol: (unspecified)

(unspecified)
SOAP
HTTP

Figure 6-26 Port wizard

After you have created a port, you can use the Properties view to review or change any property. Right-click the square representing the desired port and select **Show Properties View**. Then select **General** on the left side of the Properties view. On this page, you can enter a new name and address and you can select a binding and protocol (Figure 6-27).



port

Name: BankPort

Binding:

Address: http://www.example.org/

Protocol: SOAP

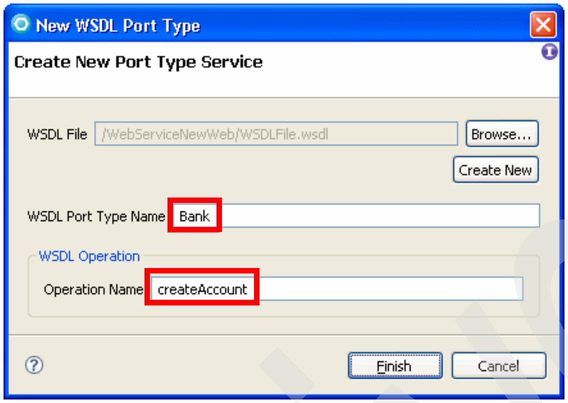
Figure 6-27 Port properties shown in the Properties view

Creating WSDL port types and operations

A port type describes the behavior of a port. It defines individual operations that can be performed and the messages that are involved. An operation is an abstract description of an action supported by a service. It provides a unique name and lists the expected inputs and outputs. It might also contain a fault element that describes any error date the operation might return.

You can create a new port type together with an operation with the help of the New WSDL Port Type wizard (Figure 6-28). You can launch this wizard either by placing the WSDL Port Type tool from the tool palette in the diagram or by right-clicking in the diagram and selecting **Add WSDL** → **Port Type**.

First you must specify the WSDL file that should contain the port type. A port type is not restricted to be in the same WSDL file as the enclosing WSDL service. As described previously you can click **Browse** to select an existing WSDL file or click **Create New** to create a new file. Then provide the port type name and operation name and click **Finish**.



Create New Port Type Service

WSDL File /WebServiceNewWeb/WSDLFile.wsdl

WSDL Port Type Name Bank

WSDL Operation

Operation Name createAccount

Figure 6-28 New WSDL Port Type wizard

The result is shown in Figure 6-29. A port type is visualized in the diagram using the interface notation with a stereotype <<`WSDL Port Type`>>. You can add further operations by selecting **Add WSDL** → **Operation** from the context menu.

Note that you did not create a connection between this port type and the port so far. You will do this in the last task.

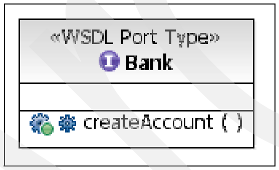


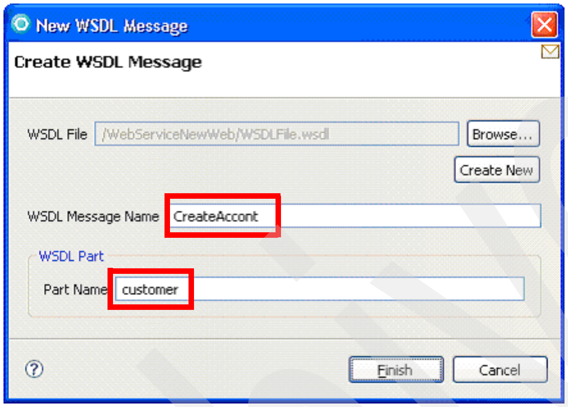
Figure 6-29 Class diagram visualizing a port type

Creating WSDL messages and parts

Messages are used by operations to describe the kind of data being communicated with clients. An operation can have an input, output and a fault message. A message is composed of one or several parts and each part is linked to a type. The individual parts of a message can be compared to the parameters of a method call in the Java language.

To create a new message along with a part, select the WSDL Message tool in the tool palette and place it in the diagram. This opens the dialog shown in Figure 6-30.

First, you must specify the WSDL file that should contain the message. Like WSDL services or port types, messages are top-level objects that can be defined in a separate WSDL file. As described previously you can either browse to select an existing file or create a new one. Finally, enter the message name and part name and click **Finish**.



New WSDL Message

Create WSDL Message

WSDL File: /WebServiceNewWeb/WSDLFile.wsdl

WSDL Message Name: CreateAccont

WSDL Part

Part Name: customer

Figure 6-30 New WSDL Message wizard

The result is shown in Figure 6-31. The newly created message is displayed using a UML class notation with stereotype <<`WSDL Message`>>. If you want to add any further part to this massage, right-click the shape and select **Add WSDL** → **Add Part**.

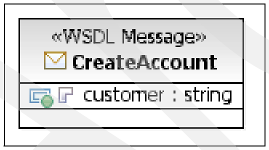


Figure 6-31 Representation of a WSDL message

The `CreateAccount` message is associated with the input-element of the `createAccount` operation in the next task. Before you proceed, create a second message `CreateAccountResponse` along with a part named account. This message will be associated with the output-element of this operation.

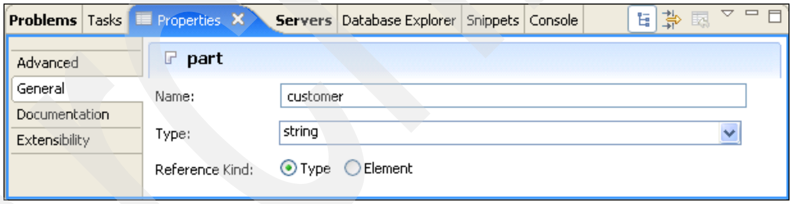
Editing parts and creating XSD types

WSDL recommends the use of XML Schema Definition (XSD) to define the type of a part. You can use the class diagram to create and edit the required XSD objects such as XSD elements, simple types, or complex types. Right-click in the diagram and select **Add WSDL**. From the following submenu select the required item (alternatively, drop the corresponding tool from the tool palette on the diagram). The different wizards are similar to the ones described beforehand. Specify the WSDL file and enter a name for the XSD object to be created.

You can add new elements to a complex type. In the diagram editor, right-click the complex type and select **Add XSD** → **Add New Element**. This creates a new element within the selected complex type with type string. You can further set or change the type of an existing XSD element. In the diagram editor, right-click an XSD element or an element within a complex type and select **Add XSD** → **Set XSD Type**. The dialog that opens provides a list of available types that you can select.

Note that once you have created an XSD element, you just can delete it from the diagram. If you want to delete it permanently from the underlying WSDL file, you must edit the file directly.

To review or change the type associated with a part, select the corresponding diagram element to bring up its properties in the Properties view. Then select the **General** tab. As shown in Figure 6-32, you can select the desired type in the drop-down combo box.



part

Name: customer

Type: string

Reference Kind: ● Type ○ Element

Figure 6-32 Properties view showing the properties of a part.

To proceed with the scenario, create two complex types, `Account` and `Customer`, and add the attributes `amount`, `name`, and `firstName` (Figure 6-33).

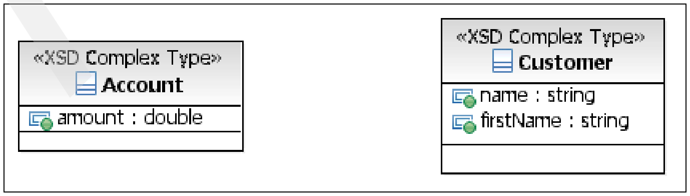


Figure 6-33 Complex types with attributes

Finally, link these types to the parts of the messages you have created (Figure 6-34).

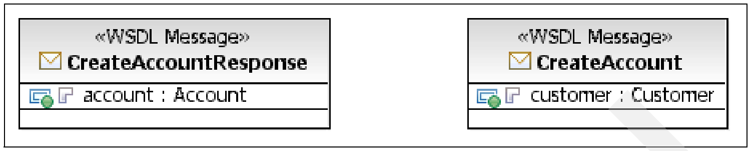
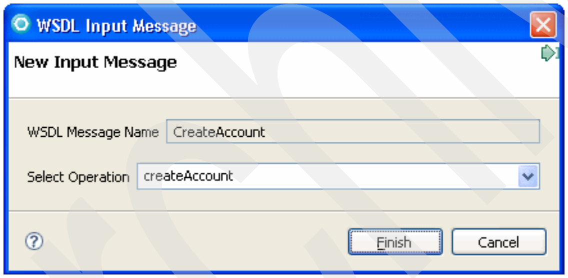


Figure 6-34 Linking complex types to WSDL messages

Adding messages to WSDL operations

An operation can reference three different types of messages to describe the data communicated with clients: Input message, output message, and fault message.

To add a message to an operation, you can use one of the following tools offered by the tool palette: WSDL Input Message, WSDL Output Message, or WSDL Fault Message. Select the desired tool, click the port type, and drag the cursor to a message you want to add. In the dialog select the desired operation and click **Finish** (Figure 6-35). Alternatively, you can use the modeling assistant to do this.



WSDL Message Name: CreateAccount

Select Operation: createAccount

Figure 6-35 Create WSDL Input Message wizard

To proceed with the example scenario, create a WSDL input message from the `createAccount` operation to the `CreateAccount` message. Then create a WSDL output message from the same operation to the `CreateAccountResponse` message. (Figure 6-36).

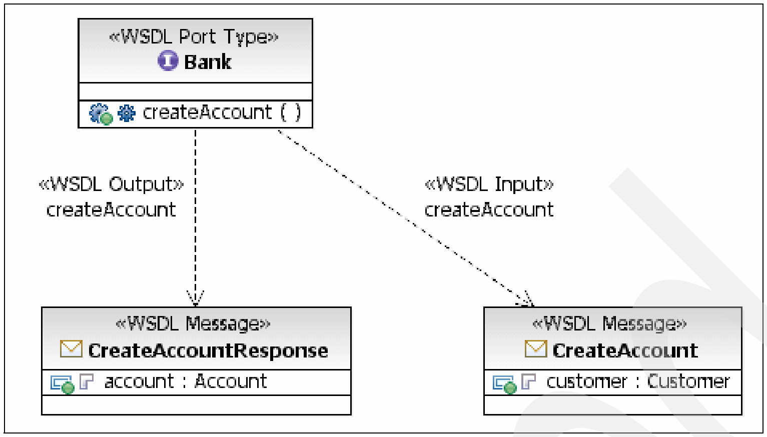


Figure 6-36 Class diagram displaying a port type with messages

Create bindings between WSDL ports and port types

A binding is used to link a port type to a port. The class diagram editor offers you several ways to do this. For example, you can use the WSDL Binding tool in the tool palette. Select the tool and select the port (remember that a port is shown as a small square on the side of a service). Then drag the cursor to the port type. A new binding is created between the port and the port type. As shown in Figure 6-37, this binding is modeled as a dependency with stereotype `<<WSDL Binding>>` between the two elements. The lollipop notation is used to represent the interface provided by the port.

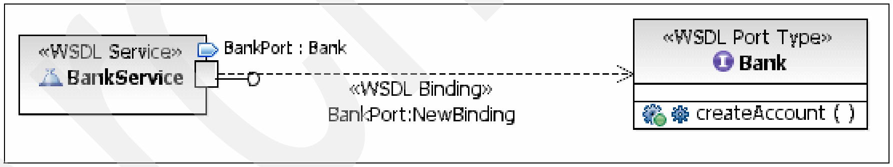
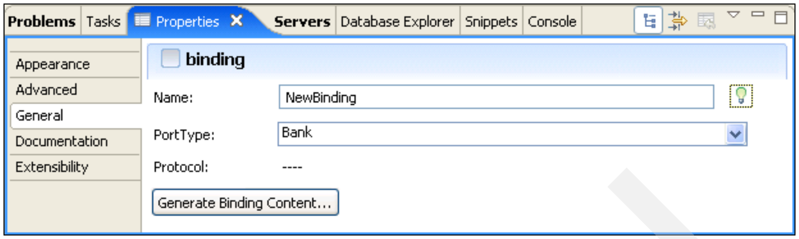


Figure 6-37 Class diagram showing a binding between a port and its port type

The last step is to generate the content for this binding. Select the dependency representing the binding and open the Properties view. On the General page click **Generate Binding Content** and complete the wizard (Figure 6-38).



binding

Name: NewBinding

PortType: Bank

Protocol: ----

Figure 6-38 Properties view displaying binding properties

After you have created the entire Web service, you can directly create an implementation of this Web Service within the diagram editor. Right-click a WSDL service component and select **Implement Web Service**. This will start the Web Service wizard that guides you through the process. You can also use the diagram editor to create a Web Service client for a given Web Service. Right-click a WSDL service component and select **Consume Web Service**.

Class diagram preferences

Rational Application Developer allows users to review and edit default settings or preferences that affect the appearance and the behavior of UML class diagrams and their content. These preferences are organized under the Modeling category within the Preferences window (select **Window** → **Preferences** → **Modeling**).

Before you create a new UML class diagram, you can set the default global preferences for attributes and operations, such as visibility styles, showing or hiding attributes and operations, showing or hiding operation signatures, and showing or hiding parent names of classifiers.

- **UML diagram settings**—Using the Diagrams page and the pages beneath users can specify several preferences regarding to the style, fonts and colors that are displayed in UML diagrams when they are created. Users can change the default settings for showing or hiding attributes, operations, operation signatures, or parent names of classifiers. They can also specify which compartments are shown by default when a new UML element is created.
- **Java development settings**—This category allows users to specify if the corresponding wizards should be used when new fields or methods are created within a class diagram. They can further specify the default values that should be applied to these wizards. The **Show Related Elements Filters** page provides the option to filter out binary Java types when the Show Related Elements action is executed. Binary Java types are types that are not defined in the workspace, but instead are available to the workspace through referenced libraries (jars).

- **EJB creation wizard settings**—The EJB preferences page allows users to specify if newly generated or created EJBs are visualized in a selected class diagram. If this option is selected and a diagram is not selected while creating or generating a new bean, this bean is opened in a *default* class diagram.
- **WSDL elements settings**—The Web Service preferences page allows users to change the default settings for visually representing existing WSDL elements. You can specify if the external or compressed view of an existing WSDL element is shown. Furthermore you can specify which WSDL components are visually represented in class diagrams. To show or hide WSDL components, select or clear the corresponding check boxes on this page.

Exploring relationships in applications

Rational Application Developer provides browse and topic diagrams that can be used to explore and navigate through an application and to view the details of its elements and relationships. They are designed to assist developers to understand and document existing code by quickly creating UML representations.

Browse diagrams

A browse diagram is a structural diagram which provides a view of a context element such as a class, a package, or an EJB. It is a temporary and non editable diagram that provides the capability to explore the given context element. You can view the element details, including attributes, methods, and relationships to other elements, and you are able to navigate to these elements. Browse diagrams can be applied to various elements including Java and Enterprise JavaBeans artifacts, but excluded are all artifacts related to Web services.

You can create a browse diagram from any source element or its representation within a class diagram. Right-click the desired element and select **Visualize → Explore in Browse Diagram**. A browse diagram is created and shown in the corresponding diagram editor. The diagram editor consists of a panel displaying the selected element along with its relationships and a tool bar. Because a browse diagram is not editable, the tool palette and the modeling assistant are not available. Depending on the elements shown, the diagram is displayed either using the radial or generalization tree layout type. The radial layout type shows the selected element in the center of the diagram, whereas the generalization tree layout type organizes the general classes at the top of the diagram and the subclasses at the bottom.

The browse diagram acts like a Web browser for your code. It provides a history and navigation, and you can customize the UML relationships you want to see. The tool bar located at the top of the browse diagram displays the context element you are currently browsing. At any one time, there is only one browse diagram open. When you browse another element, it is displayed in the same diagram replacing the previous element.

Figure 6-39 shows a sample browse diagram with the Java class `Account` as context element. You can see all attributes and methods declared by this class, and because all filter buttons are highlighted all the related elements are shown as well. You can see that `Account` is directly referenced by `Customer` and that it has two subclasses. Further, the diagram shows that somewhere in the implementation code the class `Account` is used by the `Bank` class.

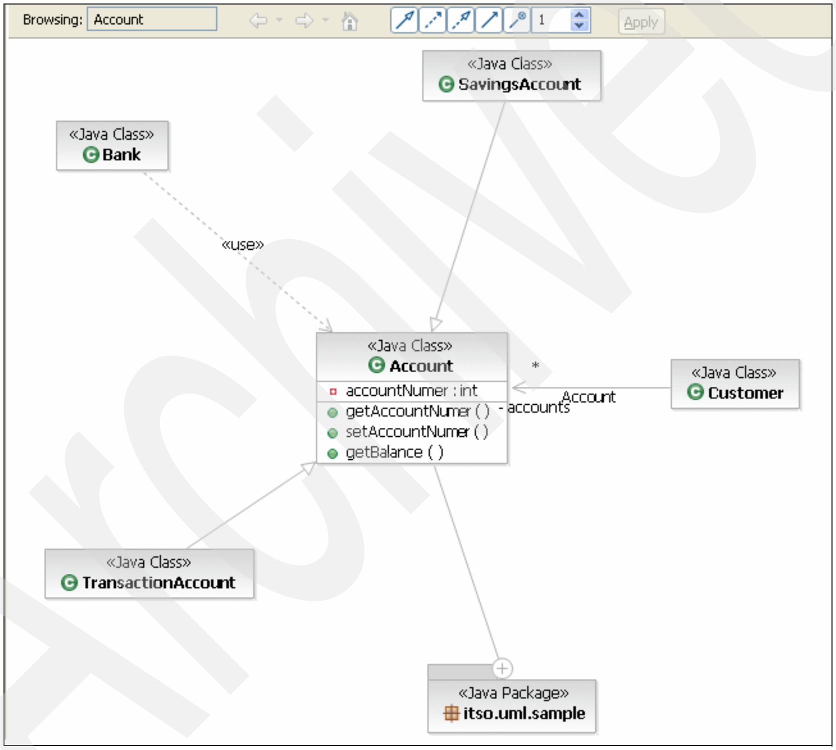


Figure 6-39 Browse diagram

The browse diagram retains the history of elements you have viewed so far. You can use the two arrow buttons provided in the tool bar to navigate backward or forward to browse previously viewed elements.

When you click the **Home** icon, the first element in the history is displayed. Furthermore, the tool bar contains a list of filter icons that can be used to filter the types of relationships that are shown along with the context element. Note that there are different filters available depending on the type of element you are currently browsing (Java or EJB artifact). To enable and disable a filter, click the appropriate icon and then click **Apply**. You can also change the number of levels of relationships that are shown for the context element. The default value is one. To change this value, specify a number and click **Apply**.

In Figure 6-39 the **Home** icon and the two arrow icons are disabled, so the current element is the first element in the browse diagram history.

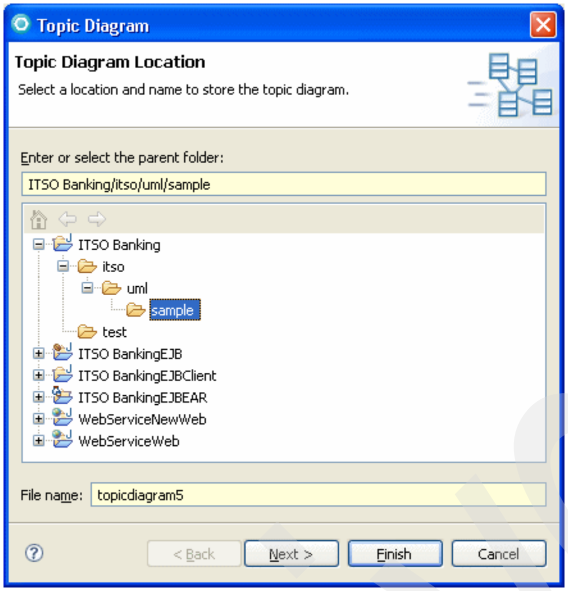
If you want to explore the details of a diagram element, double-click it. This element becomes the new context element. When you right-click a diagram element, the **Navigate** submenu provides several options to navigate to the Java source of a diagram element.

A browse diagram cannot be changed or saved, but Rational Application Developer lets you save any browse diagram view as a diagram file that is fully editable. Right-click an empty space inside the diagram and select **File** → **Save as Diagram File**. If you want to use a browse diagram as a part of the permanent documentation, you can save a browse diagram view as an image file (**File** → **Save as Image File**).

Topic diagrams

Topic diagrams provide another way to create structural diagrams from the code in your application. They are used to quickly create a query based view of relationships between existing elements in your application. These queries are called *topics* and represent commonly needed views of your code, such as showing the super type or sub types of a given class. Topic diagrams are applicable to various elements, such as Java and EJB artifacts, or WSDL files. Like browse diagrams these diagrams are not editable, but they can be saved as editable UML diagrams and shared with other team members.

A new topic diagram of an application element is created by the Topic Diagram wizard. To launch this wizard right-click the desired element and select **Visualize** → **Add to New Diagram File** → **Topic Diagram**. Once the wizard is stared, on the Topic Diagram Location page, enter or select the parent folder and provide a name for the file (Figure 6-40). Then click **Next**.


Topic Diagram

Topic Diagram Location

Select a location and name to store the topic diagram.

Enter or select the parent folder:

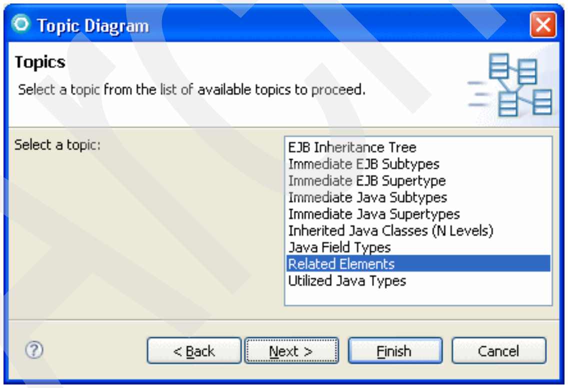
ITSO Banking/itso/uml/sample

- ITSO Banking
  - itso
    - uml
      - sample
  - test
- ITSO BankingEJB
- ITSO BankingEJBClient
- ITSO BankingEJBEAR
- WebServiceNewWeb
- WebServiceWeb

File name: topicdiagram5

Figure 6-40 Topic Diagram wizard (1)

The Topics page (Figure 6-41) provides a list of standard topics Rational Application Developer can create. Select a predefined query and click **Finish**. This creates a new topic diagram based on default values associated with the selected topic.


Topic Diagram

Topics

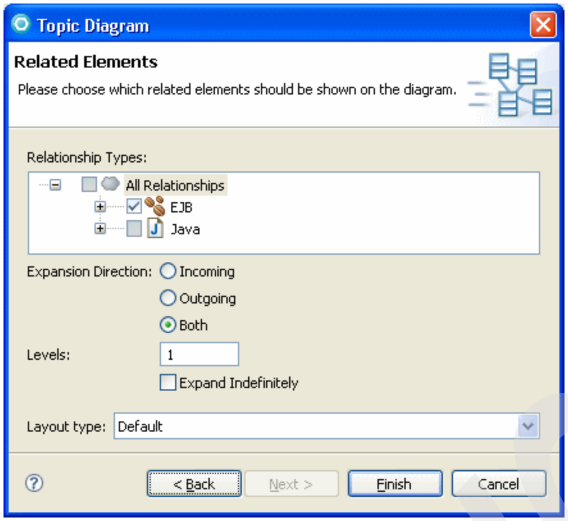
Select a topic from the list of available topics to proceed.

Select a topic:

EJB Inheritance Tree
Immediate EJB Subtypes
Immediate EJB Supertype
Immediate Java Subtypes
Immediate Java Supertypes
Inherited Java Classes (N Levels)
Java Field Types
Related Elements
Utilized Java Types

Figure 6-41 Topic Diagram wizard (2)

If you want to review or change these values click **Next** instead. The Related Elements page shown in Figure 6-42 appears.



Related Elements

Please choose which related elements should be shown on the diagram.

Relationship Types:

- □ All Relationships
  - ☑ EJB
  - □ Java

Expansion Direction: ○ Incoming ○ Outgoing ● Both

Levels: 1

□ Expand Indefinitely

Layout type: Default

Figure 6-42 Topic Diagram wizard (3)

This page shows the details of the previous selected topic and allows you to change these values. You can select different types of relationship that should be included in the query along with the expansion direction:

- If you select **Incoming**, all elements are shown that are related to the context element.
- On the other side, if you want to see all elements that have a relationship to the context element, select **Outgoing**.
- You can further specify the number of levels of relationships to query and the layout type for the diagram. The possible values are **Default** and **Radial**. These values map to the generalization and radial tree layout type described previously.

Once a topic diagram is created, you are able to review or change the underlying query. Right-click an empty space inside the diagram and select **Customize Query**.

Like browse diagrams, topic diagrams are not editable, so the tool palette and the modeling assistant are not available. You can add more elements to the diagram by right-clicking and selecting **Visualize** → **Add to Current Diagram**.

You are also able to save a topic diagram as editable diagram or as an image. Right-click an empty space and select either **File** → **Save as Diagram** or **File** → **Save as Image**.

The query and the context element that you have specified are persisted in the topic diagram. Each time when you open a topic diagram the underlying elements are queried and the diagram is automatically populated with the most current results. If you make changes to the underlying elements when a topic diagram is open, the diagram might not represent the current status of the elements until you refresh the diagram manually: Right-click an empty space in side the diagram and select **Refresh**.

Describing interactions with UML sequence diagrams

Rational Application Developer provides the capability to develop and manage sequence diagrams. A sequence diagram is an interaction diagram that can be used to describe the dynamic behavior of a system. It depicts the sequence of messages which are sent between objects in a certain interaction or scenario.

Sequence diagrams can be applied at different stages during the development process:

- Within the analysis phase, a sequence diagram can be used to describe the realization of a use case.
- Within the design phase, sequence diagrams can be more refined to show how a system accomplishes an interaction.

A sequence diagram consists of a group of objects that are represented by lifelines, and messages that these objects exchange over time during the interaction. In this context, the term object does not refer to software objects; an object represents any structural thing defined by the UML.

Figure 6-43 provides an overview of a sample sequence diagram. It describes the scenario where a customer wants to withdraw cash from an ATM. A sequence diagram has a two-dimensional nature. The horizontal axis shows the lifeline of objects that are involved in an interaction, while the vertical axis shows the sequence of creation, invocation, and destruction of these objects. Most objects that appear in a sequence diagram are in existence for the duration of the entire interaction, so their lifelines are placed on top the diagram aligned vertically. Objects can be created or destroyed during an interaction, so their lifelines start and end respectively with the receipt of a corresponding message.

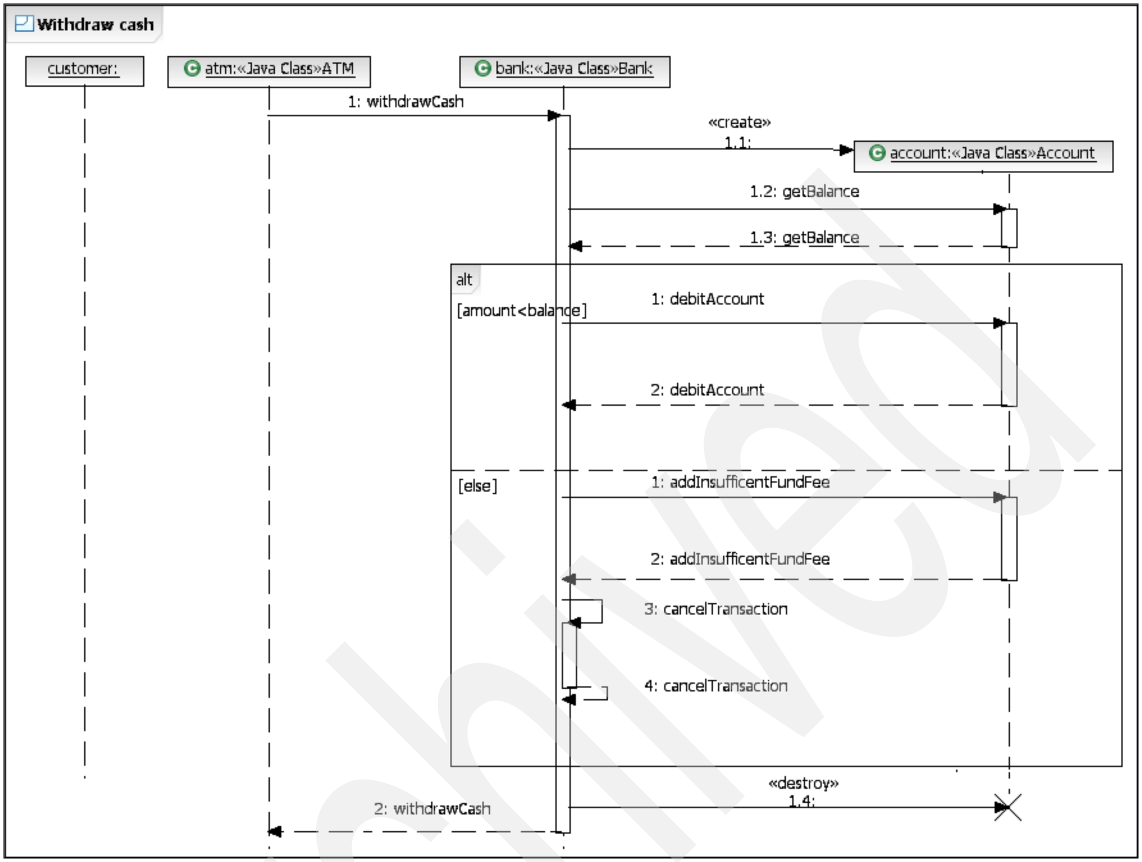


Figure 6-43 Overview of a sequence diagram

The main focus of Rational Application Developer is to model and visualize the dynamic behavior of a system rather than developing code. The tool enables developers to create, edit, and delete the various elements of a sequence diagram such as lifelines, messages and combined fragments in a visual manner. In contrast to a class diagram, the elements of a sequence diagram are not related to existing elements, such as classes or interfaces. So changes made in a sequence diagram do not affect any code.

Rational Application Developer provides two different concepts of a sequence diagram. If you create a new sequence diagram, you can add lifelines, messages and other elements to it to develop a model of an interaction. On the other side, there is another type of sequence diagram, referred to as Static Method Sequence Diagram. This non-editable diagram is used to visualize the flow of messages between existing Java objects.

Creating sequence diagrams

To create a new sequence diagram, you must use the New Sequence Diagram wizard that can be launched either directly from the top menu of Rational Application Developer (select **File** → **New** → **Other** → **Modeling** → **Sequence Diagram**) or within the Project Explorer view from the context menu of any resource, such as projects or packages.

You can also create a new sequence diagram of an existing class or interface. In the Project Explorer select the desired source element and select **Visualize** → **Add to New Diagram File** → **Sequence Diagram**. Once the wizard is started, provide a name for the file that will be created to contain the content of the diagram, and specify the parent folder where this file should be stored. Then click **Finish**.

A new sequence diagram is created and shown in the corresponding diagram editor. A tool palette on the right side offers different tools which can be used to add new elements to the current diagram such as lifelines, messages or combined fragments. There are two items in the tool palette where a solid triangle is shown right next to the item. When you click this triangle, a context menu is displayed that allows you to select another tool from this category.

The sequence diagram itself is enclosed in a frame, or more preciously a diagram frame that provides a visual border and enables the diagram to be easily reused in a different context. The frame is depicted as a rectangle with a notched descriptor box in the top left corner that provides a place for the diagram name. If you want to change the name, select this box and enter the new name.

Creating lifelines

A lifeline represents the existence of an object involved in an interaction over a period of time. A lifeline is depicted as a rectangle containing the object's name, type and stereotype. A vertical dashed line beneath the object icon indicates the progress of time. Figure 6-44 shows several examples of possible lifelines:

- The first lifeline represents an instance of the `Customer` Java class named `customer`.
- The second lifeline represents an anonymous instance of the `Customer` Java class
- The last lifeline represents an object named `customer` whose type is not specified.

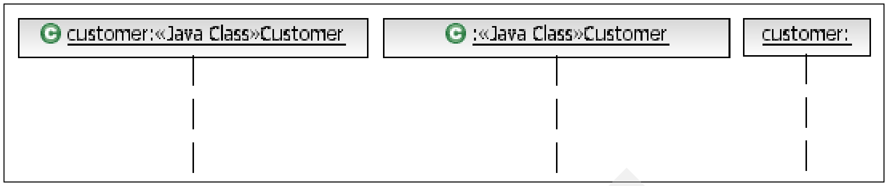


Figure 6-44 Different representations of lifelines

To add a lifeline from an existing Java element to a sequence diagram, select the desired element in the Project Explorer view and drag it on an empty place in the diagram. This creates a new lifeline and places it on top the diagram aligned vertically with the other lifelines.

You can also use the tool palette to create a new lifeline. Select the **Lifeline** element and drop it on an empty space inside the diagram. Note, that this creates a lifeline representing an object whose type is not specified. Once a lifeline is created, you can change the name and type of the object it represents.

If you want to change the name, select the lifeline's shape and enter the new name. If you want to change the type, select the desired element in the Project Explorer view and drag it on the lifeline's shape. You can also use the Properties view to review or change any property of a given lifeline.

Per default, a lifeline is shown as a rectangle containing the lifeline's name, type, and stereotype. If you right-click a lifeline, the **Filters** submenu provides several options to change the lifeline's appearance.

Creating messages

A message describes the kind of communication that occurs between two lifelines. A message is sent from a source to a target lifeline to initiate some kind of action or behavior such as invoking an operation, creating, or destroying a lifeline. The target lifeline can respond with a further message to indicate that it has finished processing.

A message is visualized as a labeled arrow that originates from the source lifeline that sends the message and ends at the target lifeline that receives it. The label is used to identify the message. It contains either a name or an operation signature if the message is used to call an operation. The label also contains a (nested) sequence number that indicates the order in which the message occurs.

- To create a message between two lifelines, hover the mouse pointer over the source lifeline so that the modeling assistant is available. Then click on the small box at the end of the outgoing arrow and drag the resulting connector on the desired target lifeline. In the context menu that appears when you drop the connector, click the desired message and enter either a name with an optional stereotype, or select an operation from the drop-down combo box.

- You can also use the tool palette to create a message. Select the desired message type by clicking the solid triangle right next to the **Message** category, then click the source lifeline and drag the cursor to the target lifeline.
- A **Create Message** (from the Palette) enables the source lifeline to create a new lifeline. The new lifeline starts when it receives this message. The symbol at the head of this lifeline is shown at the same level than the message that caused the creation (Figure 6-45). The message itself is visualized as a line with a solid arrowhead. It is common practice to indicate a Create Message with the stereotype of <<create>>. This type of message is used to highlight that a new object is created during an interaction.

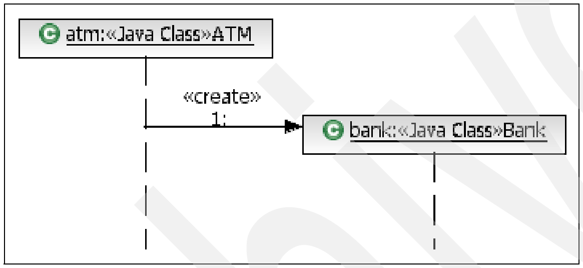


Figure 6-45 Creating a new lifeline during an interaction

- In contrast, a **Destroy Message** enables a lifeline to delete an existing lifeline. The target lifeline is terminated at that point when it receives the message. The end of the lifeline is denoted using the stop notation, a large X (Figure 6-46). A Destroy Message is drawn similar to a Create Message. The stereotype <<destroy>> is typically used to indicate a Destroy Message. You can use this type of message to describe that an object is destroyed during an interaction. Once a lifeline is destroyed, it is not possible any more to send messages to it.

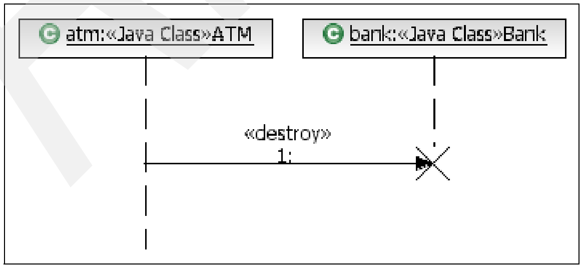


Figure 6-46 Destroying a lifeline during an interaction

- A **Synchronous Message** enables the source lifeline to invoke an operation provided by the target lifeline. The source lifeline continuous processing only after it receives a response from the target lifeline. When you create a Synchronous Message, Application Developer places three elements in the diagram:
  - A line with a solid arrowhead representing a synchronous operation invocation.
  - A dashed line with a solid arrowhead representing the return message.
  - A thin rectangle called activation bar or execution occurrence representing the behavior performed (Figure 6-47). A Synchronous Message being sent from a lifeline to itself is shown using a nested activation bar.

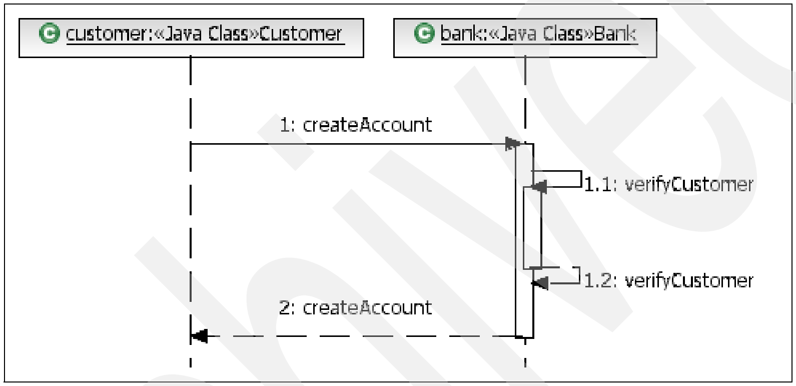


Figure 6-47 Synchronous message invocation

  Per default, only the operation's name is shown. If you want to see the full operation signature, right-click the arrow representing the message and select **Filters** → **Show Signature**.

- An **Asynchronous Message** allows the source lifeline to invoke an operation provided by the target lifeline. It continuous with the next step immediately without waiting for the target lifeline to finish processing the message. An Asynchronous Message is drawn similar as a synchronous message, but the line is drawn with an open arrowhead, and the response is omitted. You can further use the Asynchronous Message to model a signal, a special form of a message that is not associated with a particular operation.

Creating combined fragments

UML version 2.0 introduces the concept of combined fragments to support conditional and looping constructs such as if-then-else statements or to enable parts of an interaction to be reused.

Combined fragments are frames that encompass portions of a sequence diagram or provide reference to other diagrams.

A combined fragment is represented by a rectangle that comprises one or more lifelines. Its behavior is defined by an interaction operator that is drawn as a notched descriptor box in the upper left corner of the combined fragment. For example, the alternative (**alt**) interaction operator acts like an if-then-else statement (Figure 6-48).

UML 2 specifies ten further interaction operators for combined fragments. Depending on its type, a combined fragment can have one or more interaction operands. Each interaction operand represents a fragment of the interaction with an optional guard condition. The interaction operand is executed only if the guard condition is true at runtime. The absence of a guard condition means that the combined fragment is always executed. The guard condition is displayed as plain text enclosed within two brackets. A combined fragment separates the contained interaction operands with a dashed horizontal line between each operand within the frame of the combined fragment. When the combined fragment contains only one operand, the dashed line is unnecessary.

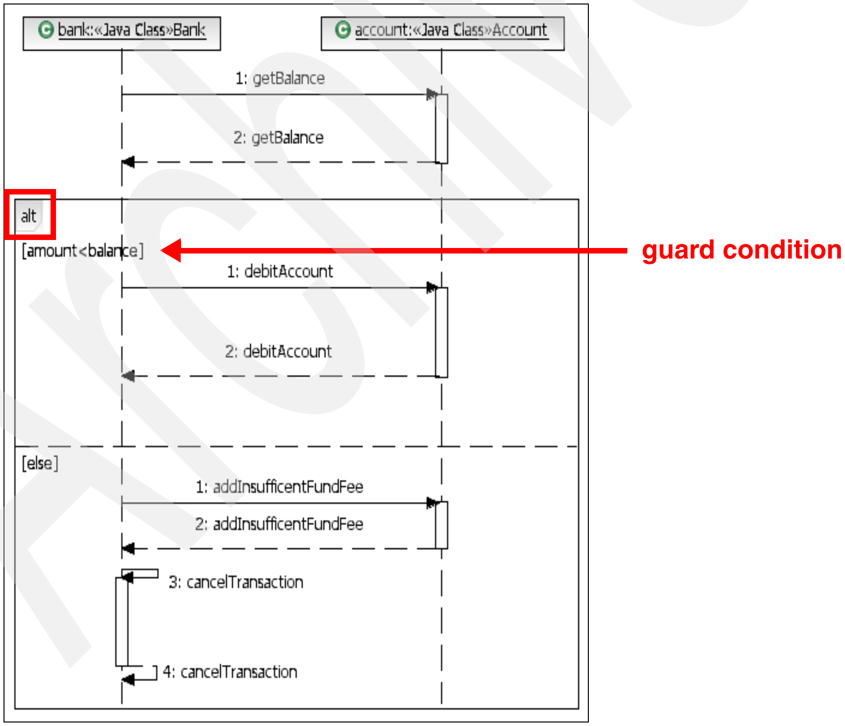


Figure 6-48 Sequence diagram with an alternative combined fragment

Figure 6-48 shows a fragment of the withdraw cash interaction. The combined fragment is used to model an alternative flow in the interaction. Because of the guard condition `[amount<balance]`, if the account balance is greater than the amount of money the customer wants to withdraw, the first interaction operand is executed. This means the interaction continuous to debit the account. Otherwise the `[else]` guard forces the second interaction operand to be executed. An insufficient fund fee is added to the account and the transaction is canceled.

To create a combined fragment, you must first select the desired tool in the tool palette. Click the solid triangle right next to the **Combined Fragment** category and select the desired tool from the available options. Then place the cursor within an empty place in the diagram and drag the combined fragment across the lifelines that you want to include in it. When you drop the cursor, the Add Covered Lifelines dialog opens that allows you to select the individual lifelines to be covered by the combined fragment. Each lifeline is represented by a check box, and each of them is selected by default. When you click **OK** a new combined fragment along with one or two interaction operands is created.

Figure 6-49 shows a newly created alternative combined fragment with two empty interaction operands (the lifelines are omitted in this diagram). If you want to specify a guard condition for an interaction operand, select the corresponding brackets and enter the text. You can create messages between the individual lifelines covered by the combined fragment in the same way as described previously. Note how the sequence numbers of the individual messages change within an interaction operand. You can also nest other combined fragments within an existing combined fragment.

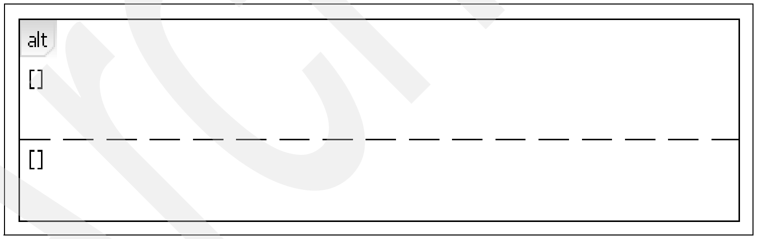


Figure 6-49 Empty combined fragment with two interaction operands

Once created, it is not possible to change the type of a combined fragment, or more preciously its interaction operator afterwards. But if you right-click a combined fragment the context menu allows you to add new interaction operands (select **Add Interaction Operand**) or to add and remove lifelines from the selected element (select **Covered Lifelines** and from the following menu select the desired action).

When you create an interaction operand, it appears in an expanded state. By clicking the small triangle at the top of the interaction operand you can collapse it to hide the entire operand and its associated messages. From its context menu there are several options available to remove or reposition the selected operand. Further you are able to add a guard condition to the operand or to add a new interaction operand if the enclosing combined fragment allows multiple operands

Creating references to external diagrams

UML Version 2.0 provides the capability to reuse interactions that are defined in another context. This provides you the ability to create complex sequence diagrams from smaller and simpler interactions. A reference to another diagram is modeled using the Interaction Use element. Like a combined fragment, an interaction use element is represented by a frame. The operator ref is placed inside the descriptor box in the upper left corner, and the name of the sequence diagram being referenced is placed inside the frame's content area along with any parameters to the sequence diagram.

An interaction use can encompass a single activation bar or several lifelines. To create a reference to another sequence diagram:

- Select **Interaction Use** in the tool palette and place the cursor on an empty space inside the source sequence diagram.
- When you drop the curser, you can specify whether you want to create an interaction use with an unspecified interaction that you will create later, or if you want to create an interaction use with a new interaction that you want to create now, or if you want to create an interaction use from an existing sequence diagram.
- In the Add Covered Lifelines dialog that opens, select the lifelines that should be encompassed by the interaction use element and click **OK**.

In Figure 6-50 the interaction use element references an interaction called Interaction 2, which provides further information how the `withdrawCash` operation is realized. When you double-click this frame, the referenced diagram is opened.

Note: Note, that at the time of writing it was not possible to truly reference another diagram. Instead it was only possible to provide a simple name.

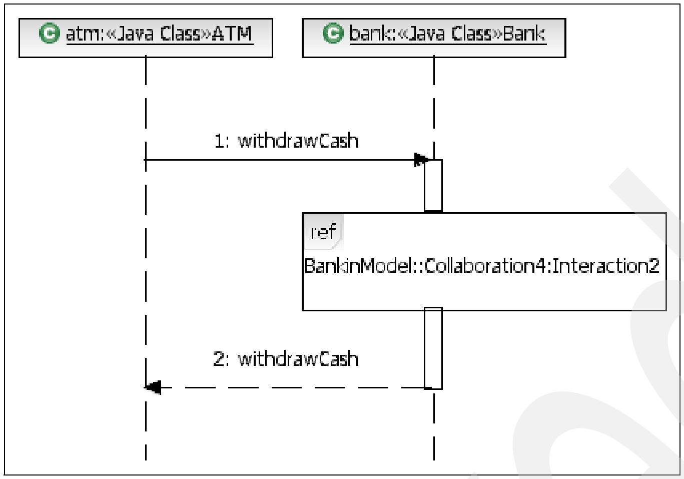


Figure 6-50 Creating a reference to another diagram

Exploring Java methods by using Static Method Sequence diagrams

The static method sequence diagram feature provided by Rational Application Developer enables developers to visualize a Java method. Existing Java code can quickly be rendered in a sequence diagram in order to visually examine the behavior of an application. A static sequence diagram from a Java method provides the full view into the entire call sequence.

To create a static method sequence diagram of a Java method:

- Right-click the desired method in the Project Explorer or Package Explorer view and select **Visualize** → **Add to New Diagram File** → **Static Method Sequence Diagram**.
- The diagram is created and shown in the corresponding diagram editor.

A static method sequence diagram is a topic diagram, so the diagram content is stored in a file with a .tpx extension. Like other topic diagrams, it is read only; the tool palette, the tool bar, and the modeling assistant are not available.

When you right-click an empty space inside the diagram, the **File** submenu provides you the options to either save this diagram as an image (**Save as Image File**), to convert this diagram to an editable UML sequence diagram (**Save as Diagram File**) or to print the entire diagram (**Print**).

Figure 6-51 shows a basic example of a static sequence diagram. It describes the flow of control when the `withdrawCash` method provided by the `ATM` class is called. The synchronous message from the diagram frame that invokes the method is called a found message. The corresponding return message is referred to as a lost message.

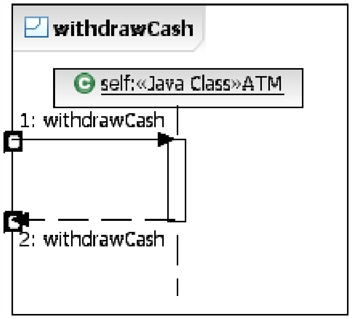


Figure 6-51 static method sequence diagram

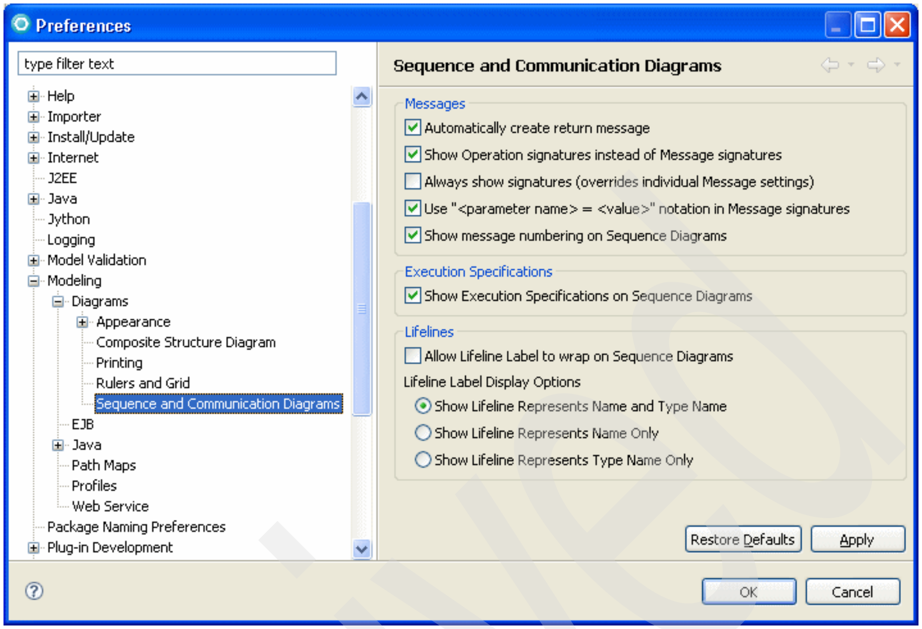
A static sequence diagram for a Java method has to be created only once. Like other topic diagrams, the query and context that have been specified when creating the diagram are stored in the diagram itself. So each time a sequence diagram is opened, Rational Application Developer queries the underlying elements and populates the diagram with the latest updates. If you want to refresh the contents of a static sequence diagram to reflect the latest changes in the source code, right-click an empty space inside the diagram and select **Refresh**.

Note: At the time of writing, Rational Application Developer failed to update the contents of a static sequence diagram when the source code was changed. A workaround is to restart Rational Application Developer with the **`-clean`** option.

Sequence diagram preferences

Using the Sequence and Communication Diagrams page, you can change default values that affect the appearance of sequence diagrams. This page is provided by the Modeling category within the Preferences window (Figure 6-52).

For example, you can specify if return messages should be created automatically or a message numbering is shown.



Sequence and Communication Diagrams

Messages

- ☑ Automatically create return message
- ☑ Show Operation signatures instead of Message signatures
- □ Always show signatures (overrides individual Message settings)
- ☑ Use "<parameter name> = <value>" notation in Message signatures
- ☑ Show message numbering on Sequence Diagrams

Execution Specifications

- ☑ Show Execution Specifications on Sequence Diagrams

Lifelines

- □ Allow Lifeline Label to wrap on Sequence Diagrams

Lifeline Label Display Options

- ● Show Lifeline Represents Name and Type Name
- ○ Show Lifeline Represents Name Only
- ○ Show Lifeline Represents Type Name Only

Figure 6-52 Sequence diagram preferences

For examples of sequence diagrams, refer to Chapter 12, “Develop Web applications using JSPs and servlets” on page 465, such as Figure 12-9 on page 486.

More information on RUP and UML

For more information about RUP and UML, we recommend the following resources. These three Web sites provide information on modeling techniques, best practices, and UML standards:

- IBM developerWorks Rational: Provides guidance and information that can help you implement and deepen your knowledge of Rational tools and best practices. This network includes access to white papers, artifacts, source code, discussions, training, and other documentation:

  `http://www.ibm.com/developerworks/rational/`

  The following link focuses on RUP and related Rational Software products. It contains further information about learning resources, products, support sites, and the latest news about this topic:

  `http://www.ibm.com/developerworks/rational/products/rup/`

In particular, we would like to highlight the following series of high quality Rational Edge articles focusing on UML topics:

http://www.ibm.com/developerworks/rational/library/content/RationalEdge/archives/uml.html

- The IBM Rational Software UML Resource Center: This is a library of UML information and resources that IBM continues to build upon and update. In addition to current news and updates about the UML, you can find UML documentation, white papers, and learning resources:

  http://www.ibm.com/software/rational/uml/index.html

- Object Management Group (OMG): These OMG Web sites provide formal specifications on UML that have been adopted by the OMG and are available in either published or downloadable form, and technical submissions on UML that have not yet been adopted:

  http://www.omg.org
  http://www.uml.org

- Craig Larmann's home page: Provides articles and related links on topics regarding to UML and RUP:

  http://www.craiglarman.com/

7

Develop Java applications

This chapter introduces the Java development capabilities and tooling features of Application Developer by developing the ITSO Bank application.

The chapter is organized into the following sections:

- Java perspectives, views, and editor overview
- Developing the ITSO Bank application
- Additional features used for Java applications
- Java editor and rapid application development

Note: Application Developer V7 fully supports the Java SE 6.0 compliance. However, Java SE 5.0 compliance is used by default. To change the compliance, follow these steps:

- Select **Window** → **Preferences** → **Java** → **Compiler**.
- Select the compliance level in the Compiler compliance level field.
- Click **Apply** and then click **OK**.

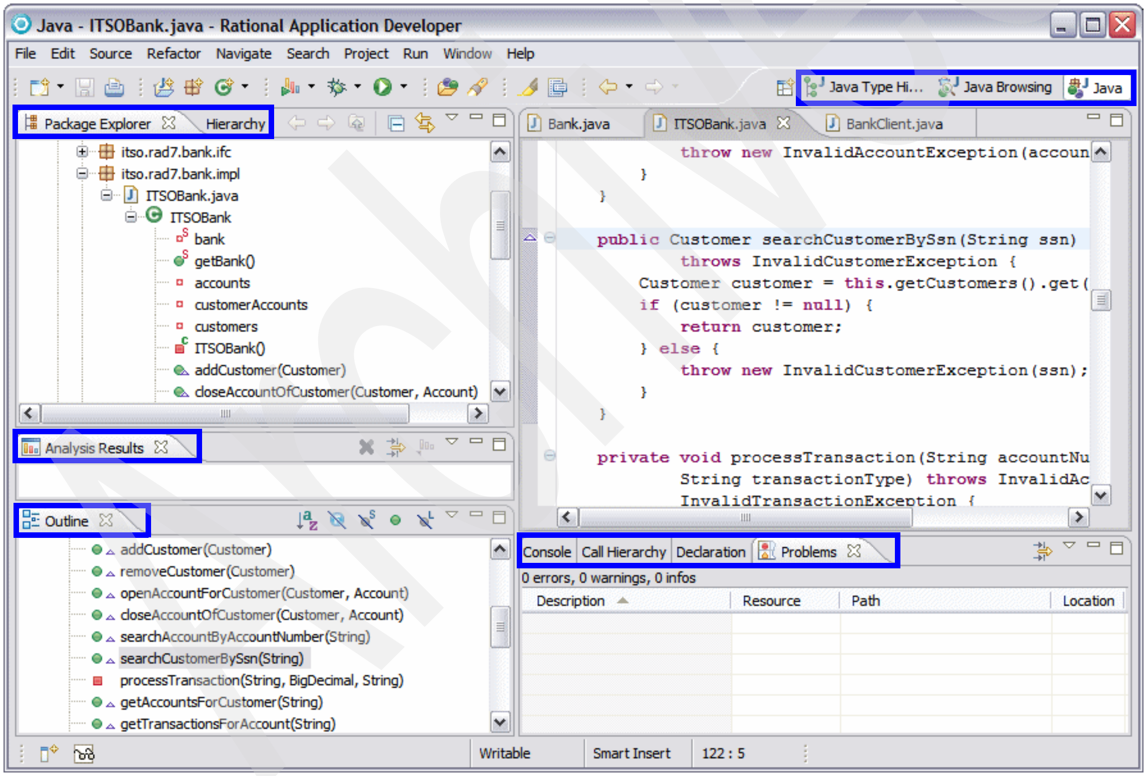
Be aware that, even when Java SE 6.0 is supported, Application Developer V7 is shipped with an IBM JRE 1.5 (5.0). Newer JRE versions can be downloaded, installed, and used in the Application Developer by the customer themselves, as described in “Plugable Java Runtime Environment (JRE)” on page 289.

Java perspectives, views, and editor overview

Within Application Developer, there are three predefined perspectives containing the views and editor that are most commonly used while developing Java SE applications:

- Java perspective
- Java Browsing perspective
- Java Type Hierarchy perspective

Those perspectives and their main views were briefly introduced in Chapter 4, “Perspectives, views, and editors” on page 117. In this section we go deeper into the details and describe some more useful views. The highlighted areas in Figure 7-1 indicate all perspectives and views we discuss.


Java - ITSOBank.java - Rational Application Developer
File Edit Source Refactor Navigate Search Project Run Window Help
Java Type Hi...
Java Browsing
Java
Package Explorer
Hierarchy
itso.rad7.bank.ifc
itso.rad7.bank.impl
ITSOBank.java
ITSOBank
bank
getBank()
accounts
customerAccounts
customers
ITSOBank()
addCustomer(Customer)
closeAccountOfCustomer(Customer, Account)
Analysis Results
Outline
addCustomer(Customer)
removeCustomer(Customer)
openAccountForCustomer(Customer, Account)
closeAccountOfCustomer(Customer, Account)
searchAccountByAccountNumber(String)
searchCustomerBySsn(String)
processTransaction(String, BigDecimal, String)
getAccountsForCustomer(String)
getTransactionsForAccount(String)
Bank.java
ITSOBank.java
BankClient.java
Console
Call Hierarchy
Declaration
Problems

```
            throw new InvalidAccountException(accoun
        }
    }

    public Customer searchCustomerBySsn(String ssn)
            throws InvalidCustomerException {
        Customer customer = this.getCustomers().get(
        if (customer != null) {
            return customer;
        } else {
            throw new InvalidCustomerException(ssn);
        }
    }

    private void processTransaction(String accountNu
            String transactionType) throws InvalidAc
            InvalidTransactionException {
```

0 errors, 0 warnings, 0 infos

Description	Resource	Path	Location

Writable   Smart Insert   122 : 5

Figure 7-1 Views in the Java perspective [customized]

Note: Figure 8-1 shows a customized Java perspective. It is the predefined Java perspective with some more useful views added. We recommend that you also customize the perspectives so that they fit your requirements. A customized perspective can be saved: **Window** → **Save Perspective As**. A modified perspective can be set back to a predefined or saved perspective: **Window** → **Reset Perspective**.

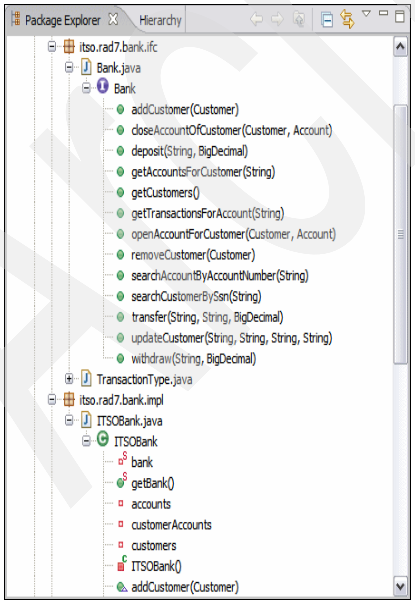
Java perspective

We use the Java perspective to develop Java SE applications or utility Java projects (utility JAR files) containing code that is shared across multiple modules within an enterprise application.

Views can be added by selecting **Window** → **Show View**.

Package Explorer view

The Package Explorer view displays all projects, packages, interfaces, classes, member variables, and member methods contained in the workspace, as shown in Figure 7-2. It allows us to easily navigate through the workspace.



```
itso.rad7.bank.ifc
  Bank.java
    Bank
      addCustomer(Customer)
      closeAccountOfCustomer(Customer, Account)
      deposit(String, BigDecimal)
      getAccountsForCustomer(String)
      getCustomers()
      getTransactionsForAccount(String)
      openAccountForCustomer(Customer, Account)
      removeCustomer(Customer)
      searchAccountByAccountNumber(String)
      searchCustomerBySsn(String)
      transfer(String, String, BigDecimal)
      updateCustomer(String, String, String, String)
      withdraw(String, BigDecimal)
  TransactionType.java
itso.rad7.bank.impl
  ITSOBank.java
    ITSOBank
      bank
      getBank()
      accounts
      customerAccounts
      customers
      ITSOBank()
      addCustomer(Customer)
```

Figure 7-2 Package Explorer view

Hierarchy view

We use the Hierarchy view to display the type hierarchy of a selected type.

To view the hierarchy of a class type, select the class in the Package Explorer, and press **F4** or right-click the class and select **Open Type Hierarchy**. The hierarchy of the selected class is displayed, as shown in Figure 7-3.

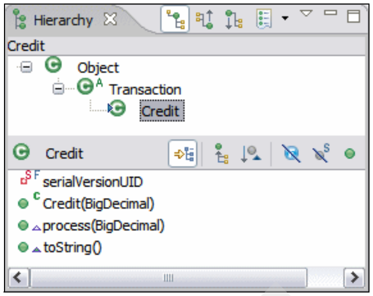


Figure 7-3 Hierarchy view for a selected class

The Hierarchy view provides three kinds of hierarchy layouts:

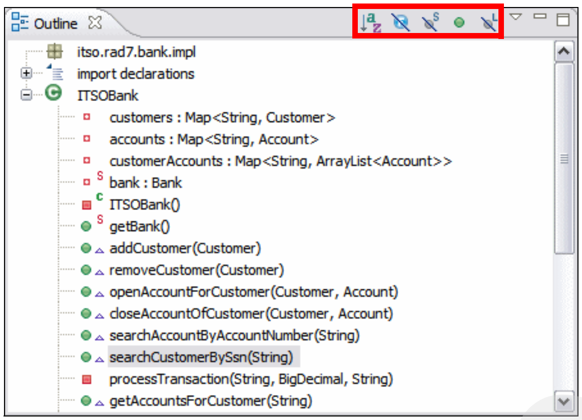
- Type Hierarchy: All supertypes and subtypes of the selected type are shown.
- Supertype Hierarchy: Only all supertypes of the selected type are shown.
- Subtype Hierarchy: Only all subtypes of the selected type are shown.

Other options in the Hierarchy view:

- Locks the view and shows members in hierarchy. For example, use this option if you are interested in all types implementing the `toString()` method.
- Shows all inherited members.
- Sorts members by theirs defining types. Defining type is displayed before the member name.
- ● Filters the displayed members.

Outline view

The Outline view is very useful and is the recommended way to navigate through a type that is currently opened in the Java editor. It lists all elements including package, import declarations, type, fields, and methods. The developer can sort and filter the elements that are displayed by using the icons highlighted in Figure 7-4.



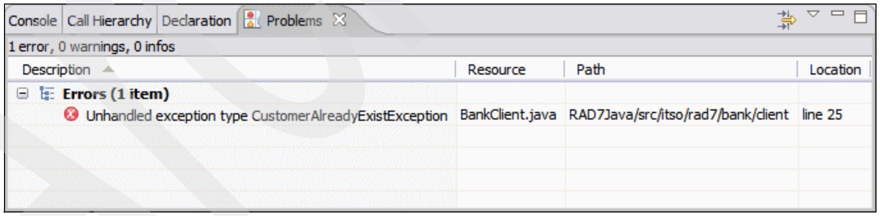
itso.rad7.bank.impl
import declarations
ITSOBank
customers : Map<String, Customer>
accounts : Map<String, Account>
customerAccounts : Map<String, ArrayList<Account>>
bank : Bank
ITSOBank()
getBank()
addCustomer(Customer)
removeCustomer(Customer)
openAccountForCustomer(Customer, Account)
closeAccountOfCustomer(Customer, Account)
searchAccountByAccountNumber(String)
searchCustomerBySsn(String)
processTransaction(String, BigDecimal, String)
getAccountsForCustomer(String)

Figure 7-4 Outline view

Problems view

While editing resource files, various builders can automatically log problems, errors, or warnings in the Problems view. For example, when you save a Java source file that contains syntax errors, those will be logged as errors, as shown in Figure 7-5. When you double-click the icon for a problem ⊗, error ⊗, or warning ⚠, the editor for the associated resource automatically opens to the relevant line of code.

Application Developer provides a quick fix for some problems. How to process a quick fix is described in “Quick fix” on page 310.



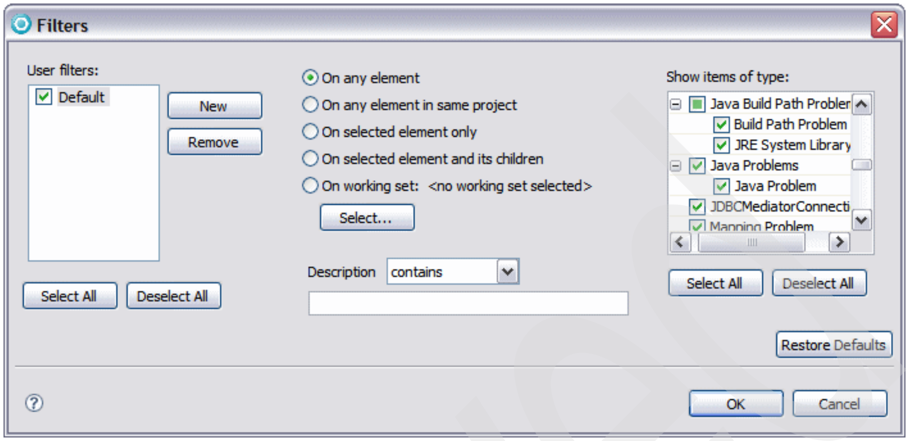
1 error, 0 warnings, 0 infos

Description	Resource	Path	Location
Errors (1 item)			
Unhandled exception type CustomerAlreadyExistException	BankClient.java	RAD7Java/src/itso/rad7/bank/client	line 25

Figure 7-5 Problems view with an error notification

Note: Java builder is responsible for all Java resource files. However, in an enterprise application project, other builders can be used. Builders can be enabled and disabled for each project. Right-click the project in the Package Explorer and select first **Properties** and then **Builders**.

The Problems view allows you to filter the problems to show only specific types of problems by clicking , which opens the Filters dialog (Figure 7-6).



☑ Default

● On any element
○ On any element in same project
○ On selected element only
○ On selected element and its children
○ On working set: <no working set selected>

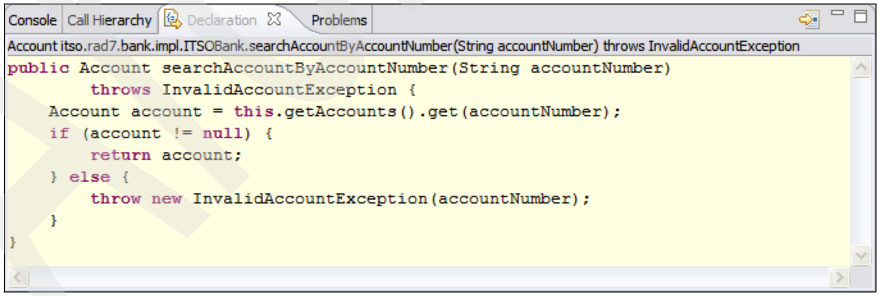
Description contains

☑ Java Build Path Probler
☑ Build Path Problem
☑ JRE System Library
☑ Java Problems
☑ Java Problem
☑ JDBCMediatorConnecti
☑ Mapping Problem

Figure 7-6 Filters dialog of Problems view

Declaration view

This view displays the declaration and definition of the currently selected type or element, as shown in Figure 7-7. It is most useful to see the source code of a referenced type within your code. For example, if you reference a `customer` within your code and you want to see the implementation of the `Customer` class, just select the referenced type `Customer`, and the source code of the `Customer` class is displayed in this view. Clicking directly opens the source file of the selected type in the Java editor.



Account itso.rad7.bank.impl.ITSOBank.searchAccountByAccountNumber(String accountNumber) throws InvalidAccountException

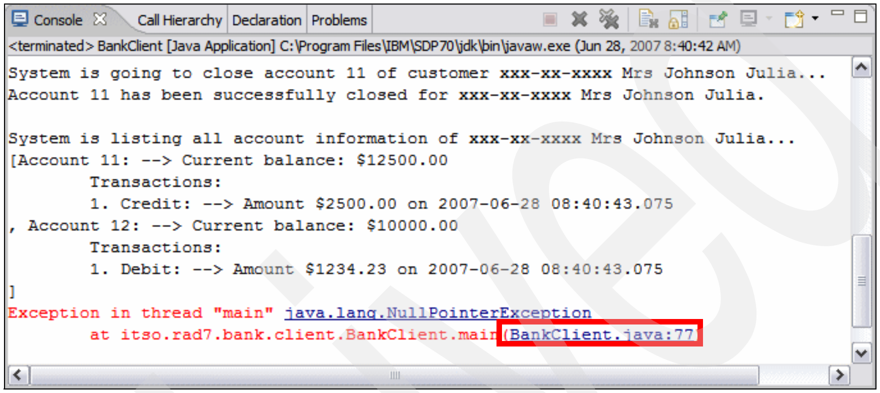
```
public Account searchAccountByAccountNumber(String accountNumber)
        throws InvalidAccountException {
    Account account = this.getAccounts().get(accountNumber);
    if (account != null) {
        return account;
    } else {
        throw new InvalidAccountException(accountNumber);
    }
}
```

Figure 7-7 Declaration view

Console view

The console is the view in which the Application Developer writes all outputs of a process and allows you to provide keyboard inputs to the Java application while running it. Uncaught exceptions are also displayed in the console.

The highlighted link in Figure 7-8 directs you to the line in the source code where the exception has been thrown.



```
<terminated> BankClient [Java Application] C:\Program Files\IBM\SDP70\jdk\bin\javaw.exe (Jun 28, 2007 8:40:42 AM)
System is going to close account 11 of customer xxx-xx-xxxx Mrs Johnson Julia...
Account 11 has been successfully closed for xxx-xx-xxxx Mrs Johnson Julia.

System is listing all account information of xxx-xx-xxxx Mrs Johnson Julia...
[Account 11: --> Current balance: $12500.00
        Transactions:
        1. Credit: --> Amount $2500.00 on 2007-06-28 08:40:43.075
, Account 12: --> Current balance: $10000.00
        Transactions:
        1. Debit: --> Amount $1234.23 on 2007-06-28 08:40:43.075
]
Exception in thread "main" java.lang.NullPointerException
        at itso.rad7.bank.client.BankClient.main(BankClient.java:77)
```

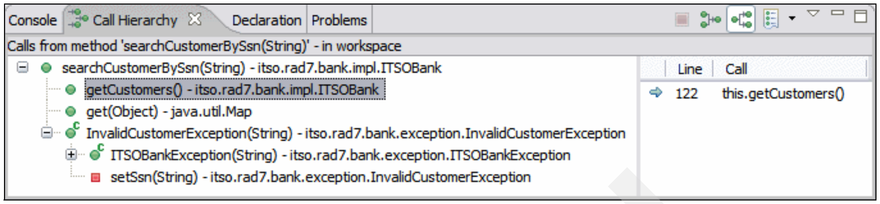
Figure 7-8 Console view with standard outputs and an exception

Other options in the Console view:

- 🟥 Terminates the currently running process. It is a useful button to terminate a process running in an endless loop.
- ✖ and ✖ Removes terminated launches from the console.
- 🗒 Clears the console.
- 🔒 Enables scroll lock in the console.
- 📌 Pins the current console to remain on the top.

Call Hierarchy view

The Call Hierarchy view displays all callers and callees of a selected method, as shown in Figure 7-9. To view the call hierarchy of a method, select it in the Package Explorer or in the source code, and press **Ctrl+Alt+H** or right-click, and select **Open Call Hierarchy**.



Calls from method 'searchCustomerBySsn(String)' - in workspace

```
searchCustomerBySsn(String) - itso.rad7.bank.impl.ITSOBank
    getCustomers() - itso.rad7.bank.impl.ITSOBank
    get(Object) - java.util.Map
    InvalidCustomerException(String) - itso.rad7.bank.exception.InvalidCustomerException
        ITSOBankException(String) - itso.rad7.bank.exception.ITSOBankException
        setSsn(String) - itso.rad7.bank.exception.InvalidCustomerException
```

Line	Call
122	this.getCustomers()

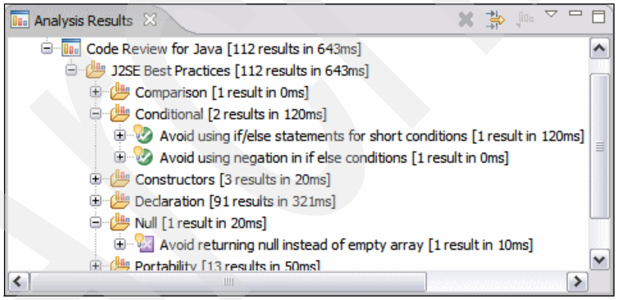
Figure 7-9   Call Hierarchy view [Callee Hierarchy]

Call Hierarchy view provides two kind of hierarchy layouts:

- Caller Hierarchy: All the members calling the selected method are shown.
- Callee Hierarchy: All members called by the selected method are shown.

Analysis Results view

The **Java code review provider** is a part of the Test and Performance Tools Platform (TPTP) analysis framework. It lets you run a static analysis of the resources that you are working with to detect violations of rules and rule categories. The generated report of results that do not conform to the rules you specified is displayed in the Analysis Results view, as shown in Figure 7-10. The view allows you to directly open resources in their associated editors to correct problems, or apply quick fixes, if they are available, the same way as in the Problems view.



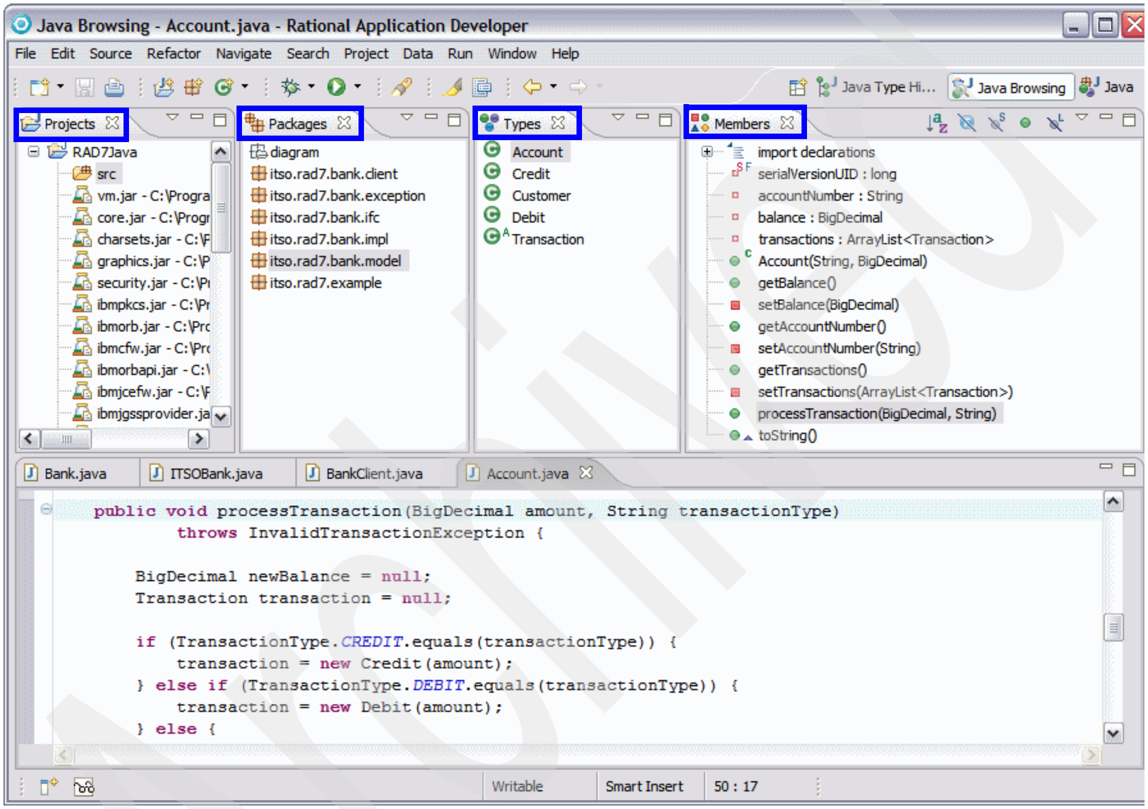
```
Code Review for Java [112 results in 643ms]
    J2SE Best Practices [112 results in 643ms]
        Comparison [1 result in 0ms]
        Conditional [2 results in 120ms]
            Avoid using if/else statements for short conditions [1 result in 120ms]
            Avoid using negation in if else conditions [1 result in 0ms]
        Constructors [3 results in 20ms]
        Declaration [91 results in 321ms]
        Null [1 result in 20ms]
            Avoid returning null instead of empty array [1 result in 10ms]
        Portability [13 results in 50ms]
```

Figure 7-10   Analysis Results view [Java code review]

How to configure and perform a source code analysis is described in “Analyzing source code” on page 282.

Java Browsing perspective

The Java Browsing perspective is used to browse and manipulate your code. In contrast to the Package Explorer view, which organizes all Java elements in a tree, this perspective uses distinct views highlighted in Figure 7-11 to present the same information.



```
public void processTransaction(BigDecimal amount, String transactionType)
        throws InvalidTransactionException {

    BigDecimal newBalance = null;
    Transaction transaction = null;

    if (TransactionType.CREDIT.equals(transactionType)) {
        transaction = new Credit(amount);
    } else if (TransactionType.DEBIT.equals(transactionType)) {
        transaction = new Debit(amount);
    } else {
```

Figure 7-11 Views in the Java Browsing perspective

Java Type Hierarchy perspective

The Java Type Hierarchy perspective contains only the Hierarchy view, which was described in “Hierarchy view” on page 230.

Developing the ITSO Bank application

In this section we demonstrate how Application Developer can be used to develop a Java SE application. We create a Java project including several packages, interfaces, classes, fields, and methods.

Note: The sample code described in this chapter can be completed by following along in the procedures documented. Alternatively, you can import the sample Java code provided in:

```
c:\7501code\zInterchangeFiles\java\RAD7Java.zip
```

Refer to Appendix B, “Additional material” on page 1307 for instructions on how to download the sample code.

ITSO Bank application overview

The example application to work through in this section is called ITSO Bank. The banking example is deliberately over-simplified, and the exception handling is ignored to keep the example concise and relevant to our discussion.

Interfaces and classes overview

The application contains these classes and interfaces:

- **`Bank`** interface—This defines common operations a bank would perform, and typically includes customer, account, and transaction related services.
- **`TransactionType`** interface—This defines the kind of transactions the bank allows.
- **`ITSOBank`** class—This is an implementation of the `Bank` interface.
- **`Account`** class—This is the representation of a bank account. It logs all transactions performed on it for logging and querying purposes.
- **`Customer`** class—This is the representation of a client of bank, an account holder. A customer can have one or more accounts.
- **`Transaction`** class—This is an abstract supertype of all transactions. A transaction is a single operation that will be performed on an account. In the example only two transaction types exist: Debit and credit.

- **Debit** class—This is one of the two existing concrete subtypes of the Transaction class. This transaction results in an account being debited by the amount indicated.
- **Credit** class—This is the other concrete subtype of Transaction class. It results in an account being credited by the amount indicated.
- **BankClient** class—This is the executable class of the ITSO Bank application. It creates instances of ITSOBank, Customer, Account classes, and performs some transactions on the accounts.
- All exception classes in the package itso.rad7.bank.exception—These are the implemented exceptions that can occur in the ITSO Bank application. **ITSOBankException** is the supertype of all ITSO Bank application exceptions.

Packaging structure

The ITSO Bank application contains several packages. Table 7-1 lists the packages and describes theirs purposes.

Table 7-1 ITSO Bank application packages

Package	Description
itso.rad7.bank.ifc	Contains the interfaces of the application
itso.rad7.bank.impl	Contains the bank implementation class
itso.rad7.bank.model	Contains all business model classes of the application
itso.rad7.bank.exception	Contains the exception classes of the application
itso.rad7.bank.client	Contains the application client which we use to run the application

Class diagram

A UML class diagram helps to overview the interfaces and classes and their relationships. In the class diagram in Figure 7-12, we added the packages as well to get a complete picture of the ITSO Bank application.

We will create this diagram by using Application Developer's UML modeling tool in "Creating a UML class diagram" on page 242.

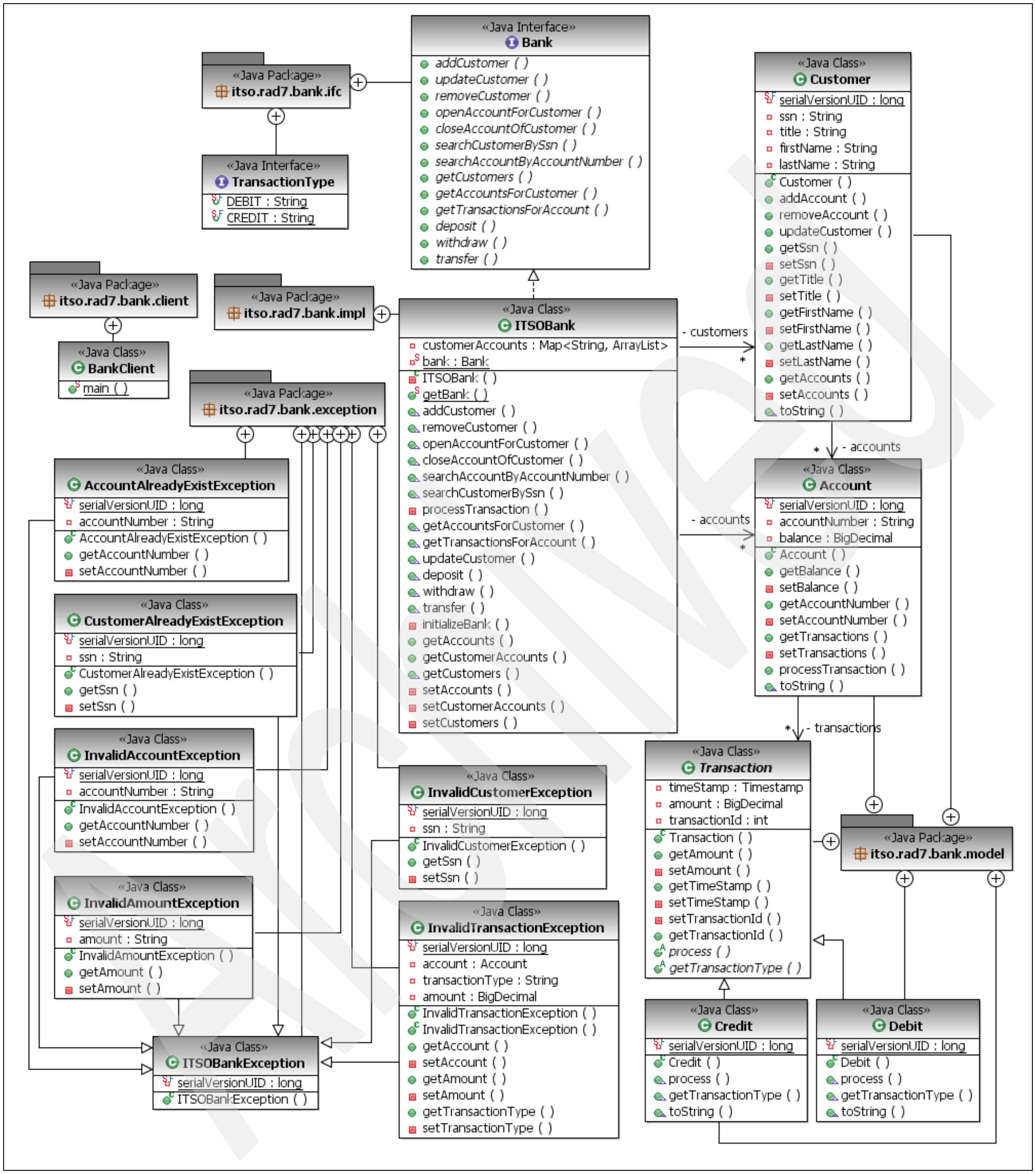


Figure 7-12 UML class diagram: ITSO Bank application

ITSO Bank application step-by-step development guide

In the next few sections we provide you a step by step guide to develop the ITSO Bank application in Application Developer.

Creating a Java project

A Java project contains all resources needed for a Java application as source and class files and all other resources, for example, image or properties files.

> **Note:** Projects are not defined in the Java SE specification. They are used in Application Developer as the lowest unit to organize the workspace.

Application Developer must be started at that time, and we recommend that you work with the Java perspective. The way to change a perspective is described in “Switching perspectives” on page 121.

To create a new Java project, do the following steps:

- From the workbench select **File** → **New** → **Project** or use the New Java Project wizard by clicking in the toolbar. If you use the New Java Project wizard, you do not have to select the type of project.
- Select **Java Project** or **Java** → **Java Project** and click **Next** (Figure 7-13).

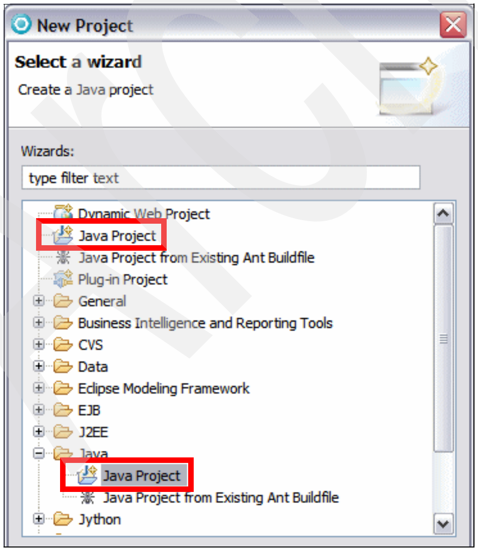


Figure 7-13 New Project dialog

- In the New Java Project dialog, enter the following items (Figure 7-14, left):
  - Project name: `RAD7Java`
  - Contents: Select **Create new project in workspace** (default).

    > **Note:** By default, the project files are stored in a folder created under Application Developer workspace directory. Select **Create project from existing source** to change the directory of the project.
    >
    > In the Package Explorer view, you cannot see the generated `.class` files. If you want to see the folder structure and the files as they are in the file system, you have to open the Navigator view. Select **Window** → **Show View** → **Navigator**. Now you can see the sources in the `src` directory and the bytecode files in the `bin` directory.

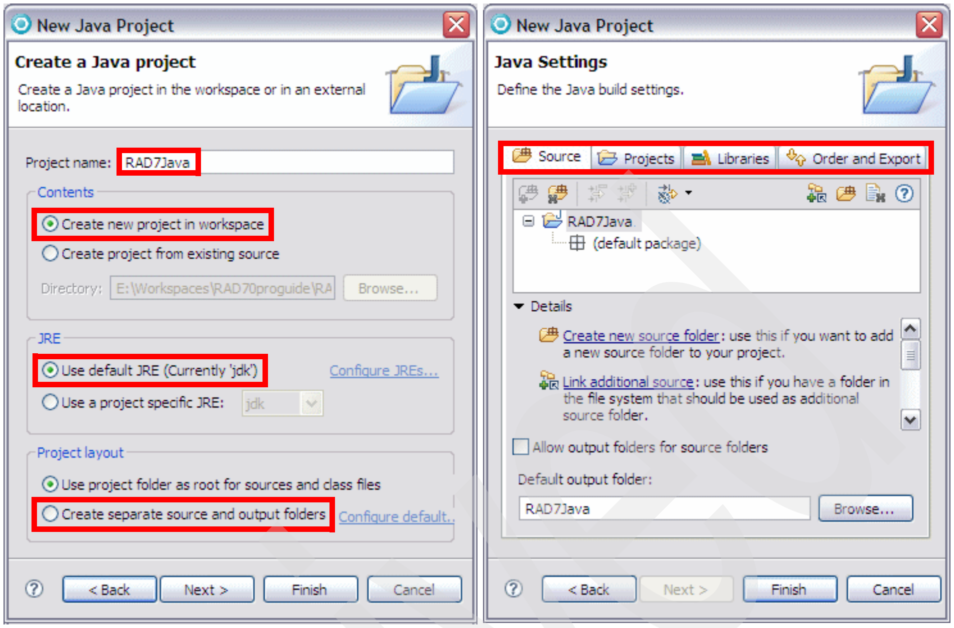
  - JRE: Select **Use default JRE** (default).

    > **Note:** If your project requires to run on a specific JRE, select **Use a project specific JRE** and select the JRE which has to be used. Clicking **Configure JREs** opens the preferences dialog where you can configure or add new JREs.

  - Project layout: Select **Create separate source and output folders**.

    > **Note:** By default, **Use project folder as root for source and class files** is selected. If you select **Create separate source and output folders**, you can optionally change the directories to something other than `src` and `bin` by clicking **Configure default**.

  - Click **Next**.



New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Project name: RAD7Java

Contents

● Create new project in workspace

○ Create project from existing source

Directory: E:\Workspaces\RAD70proguide\RA Browse...

JRE

● Use default JRE (Currently 'jdk') Configure JREs...

○ Use a project specific JRE: jdk

Project layout

● Use project folder as root for sources and class files

○ Create separate source and output folders Configure default...

< Back | Next > | Finish | Cancel

New Java Project

Java Settings

Define the Java build settings.

Source | Projects | Libraries | Order and Export

RAD7Java

(default package)

Details

Create new source folder: use this if you want to add a new source folder to your project.

Link additional source: use this if you have a folder in the file system that should be used as additional source folder.

□ Allow output folders for source folders

Default output folder:

RAD7Java Browse...

< Back | Next > | Finish | Cancel

Figure 7-14 New Java Project: Create a Java project and Java Settings

- In the New Java Project - Java Settings dialog (Figure 7-14, right) we accept the default settings for each of the tabs by just clicking **Finish**.

  This dialog allows you to change the build path settings for a Java project. The build class path is a list of paths visible to the compiler when building the project.

  – **Source** tab—This tab allows you to add and remove source folders from the Java project. The compiler translates all `.java` files found in the source folders to `.class` files and stores them to the output folder. The output folder is defined per project except if a source folder specifies an own output folder. Each source folder can define an exclusion filter to specify which resources inside the folder are not visible to the compiler.

  – **Projects** tab—This tab allows you to add another project within the workspace to the build path for this new project (project dependencies).

  – **Libraries** tab—This tab allows you to add libraries to the build path. There are five options:

    - Workspace-managed (internal) JAR files:

      Add JARs—Allows you to navigate the workspace hierarchy and select JAR files to add to the build path.

- File system (external) JAR files:

  Add External JARs—Allows you to navigate the file system (outside the workspace) and select JAR files to add to the build path.

- Folders containing `.class` files:

  Add Class Folder—Allows you to navigate the workspace hierarchy and select a class folder for the build path.

- Predefined libraries like the JRE system library:

  Add Library—Allows you to add predefined libraries like JUnit or Standard Widget Toolkit (SWT).

- JAR files indirectly as class path variables:

  Add Variable—Allows you to add classpath variables to the build path. Classpath variables are an indirection to JARs with the benefit of avoiding local file system paths in a classpath. This is needed when projects are shared in a team.

  Variables can be created and edited in the Classpath Variable preference page. Select **Window** → **Preferences** → **Java** → **Build Path** → **Classpath Variables**.

– **Order and Export** tab—This tab allows you to change the build path order. You specify the search order of the items in the build path.

  Select an entry in the list if you want to export it. Exported entries are visible to other projects that require the new Java project being created. The source folder itself is always exported and therefore cannot be deselected.

Creating a UML class diagram

Application Developer supports UML class diagrams. It allows the developer to create a static visual representation of the packages, interfaces, classes, and their relationships. Application Developer calls automatically the related wizard to create the Java code while adding elements to the diagram. Of course, you can also add already existing elements to a class diagram. Drag the element in the Package Explorer with the left mouse button and drop it in the class diagram by releasing the button.

To create an UML class diagram, do the following steps:

▸ Create a new folder called `diagram` in the `RAD7Java` project under `src`:

  – Right-click the `src` folder in the `RAD7Java` project in the Project Explorer and select **New** → **Folder** or **New** → **Other** → **General** → **Folder**.

  – For Folder name, type `diagram` and click **Finish** to create the folder.

- Create an empty class diagram:
  - Right-click the `diagram` folder in the Project Explorer and select **New** → **Class Diagram** or **New** → **Other** → **Modeling** → **Class Diagram**.
  - Type `ITSOBank-ClassDiagram` as the name and click **Finish**.
  - Click **OK** to confirm enabling of Java Modeling.
- The file `ITSOBank-ClassDiagram.dnx` appears in the `diagram` folder and opens in the Visualizer Class Diagram editor.
- Notice that the Java Drawer is open in the Palette, as shown in Figure 7-15. Application Developer automatically opens the Java Drawer by default for a Java project.

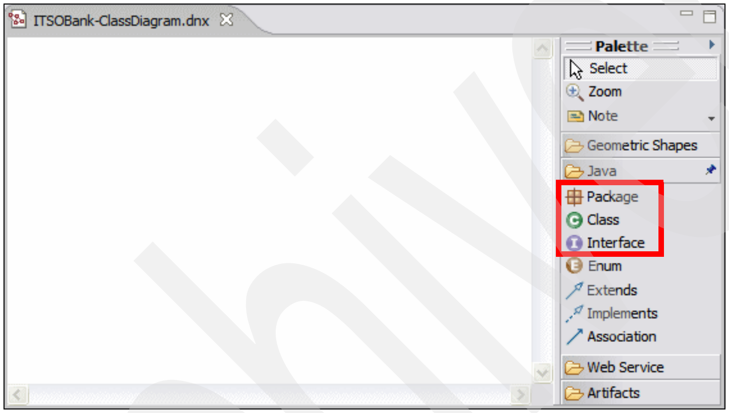


Figure 7-15 Visualizer Class Diagram editor with Java Drawer in the Palette

Creating Java packages

Once the Java project has been created, Java packages can be added to the project using either the Visualizer Class Diagram editor or the New Java Package wizard. In fact, Visualizer Class Diagram editor is calling the New Java Package wizard as well, so there is no differences between these two mechanisms.

We create the following Java packages:

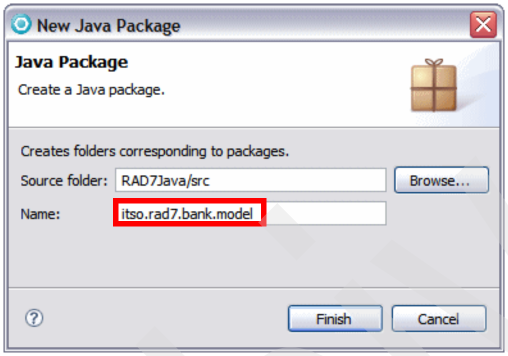
```
itso.rad7.bank.ifc
itso.rad7.bank.impl
itso.rad7.bank.model
itso.rad7.bank.exception
itso.rad7.bank.client
```

The packages are described in Table 7-1 on page 237.

Create a Java package in the Visualizer Class Diagram editor

To create a Java package in the Visualizer Class Diagram editor, do these steps:

- Select Package in the Java Drawer, as shown in the Figure 7-15 on page 243.
- Once the package icon is selected, click anywhere in class diagram editor. The New Java Package wizard opens.
- Type one of the package names (`itso.rad7.bank.model`) and click **Finish** to create the package (Figure 7-16).



New Java Package

Java Package

Create a Java package.

Creates folders corresponding to packages.

Source folder: RAD7Java/src

Name: itso.rad7.bank.model

Figure 7-16 Create a Java package

Create a Java package using the New Java Package wizard

To create a Java package using the New Java Package wizard, do these steps:

- Right-click the `src` folder in the `RAD7Java` project and select **New** → **Package** or **New** → **Other** → **Java** → **Package**, or click in the toolbar.
- Type a package name and click **Finish**.
- To add a package to the class diagram, you can just drag and drop the package to the class diagram editor, or right-click the package in the Package Explorer and select **Visualize** → **Add to Current Diagram**.

> **Note:** Do not forget to create all required Java packages.

Creating Java interfaces

In this section we create the following Java interfaces for the ITSO Bank application:

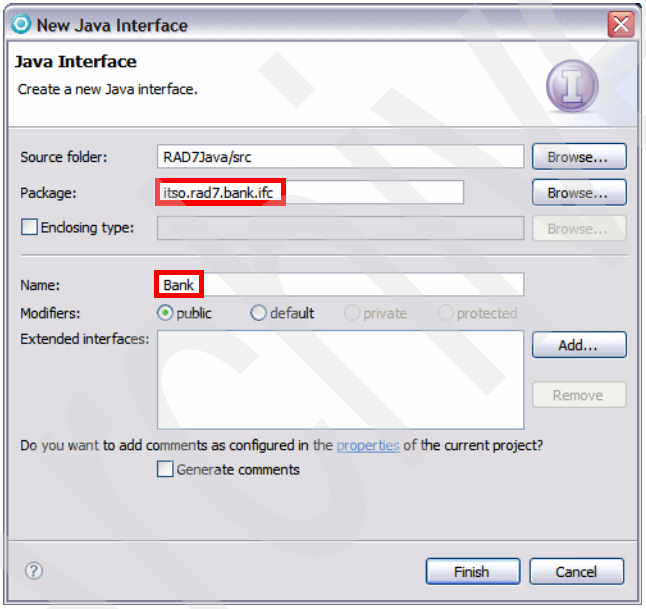
- `Bank`
- `TransactionType`

Interfaces and classes are described in “Interfaces and classes overview” on page 236. Again, you can select to create the interfaces in the Visualizer Class Diagram editor or by using the New Java Interface wizard.

Create Java interface in the Visualizer Class Diagram editor

To create a Java interface in the Visualizer Class Diagram editor, do these steps:

- Select Interface in the Java Drawer and click anywhere in class diagram editor field. The New Java Interface wizard opens.
- In the New Java Interface dialog, enter the following data (Figure 7-17):
  - Source folder: `RAD7Java/src` (default)
  - Package: `itso.rad7.bank.ifc` (click **Browse** to select the package)
  - Name: `Bank`
  - Keep the default for all other settings.



New Java Interface

Java Interface

Create a new Java interface.

Source folder: RAD7Java/src

Package: itso.rad7.bank.ifc

□ Enclosing type:

Name: Bank

Modifiers: ● public ○ default ○ private ○ protected

Extended interfaces:

Do you want to add comments as configured in the properties of the current project?

□ Generate comments

Figure 7-17 Create a Java interface

- Click **Finish** to create the Java interface.
- Notice that a line appears between the package and the `Bank` interface.

Create a Java interface using the New Java Interface wizard

To create a Java interface using the New Java Interface wizard, do these steps:

- Right-click the `itso.rad7.bank.ifc` package and select **New** → **Interface** or click the arrow in the ▾ icon in the toolbar and then select Interface .

- In the dialog type `TransactionType` for the interface and click **Finish**.
- To add an interface to the class diagram, you can just drag&drop the interface to the class diagram editor, or select the interface in the Package Explorer and **Visualize** → **Add to Current Diagram**.

Creating Java classes

In this section we create the all Java classes as they are listed in Table 7-2 of the ITSO Bank application:

Table 7-2 ITSO Bank application classes

Class name	Package	Superclass	Modifiers	Interfaces
ITSOBank	itso.rad7.bank.impl	java.lang.Object	public	itso.rad7.bank.ifc.Bank
Account	itso.rad7.bank.model	java.lang.Object	public	java.io.Serializable
Customer	itso.rad7.bank.model	java.lang.Object	public	java.io.Serializable
Transaction	itso.rad7.bank.model	java.lang.Object	public abstract	java.io.Serializable
Credit	itso.rad7.bank.model	itso.rad7.bank.model.Transaction	public	
Debit	itso.rad7.bank.model	itso.rad7.bank.model.Transaction	public	
BankClient Note: Select *public static void main(String[] args)*	itso.rad7.bank.client	java.lang.Object	public	
ITSOBankException	itso.rad7.bank.exception	java.lang.Exception	public	
AccountAlreadyExistException	itso.rad7.bank.exception	itso.rad7.bank.exception.ITSOBankException	public	
CustomerAlreadyExistException	itso.rad7.bank.exception	itso.rad7.bank.exception.ITSOBankException	public	
InvalidAccountException	itso.rad7.bank.exception	itso.rad7.bank.exception.ITSOBankException	public	
InvalidAmountException	itso.rad7.bank.exception	itso.rad7.bank.exception.ITSOBankException	public	
InvalidCustomerException	itso.rad7.bank.exception	itso.rad7.bank.exception.ITSOBankException	public	

Class name	Package	Superclass	Modifiers	Interfaces
InvalidTransaction Exception	itso.rad7.bank.exception	itso.rad7.bank.exception. ITSOBankException	public	

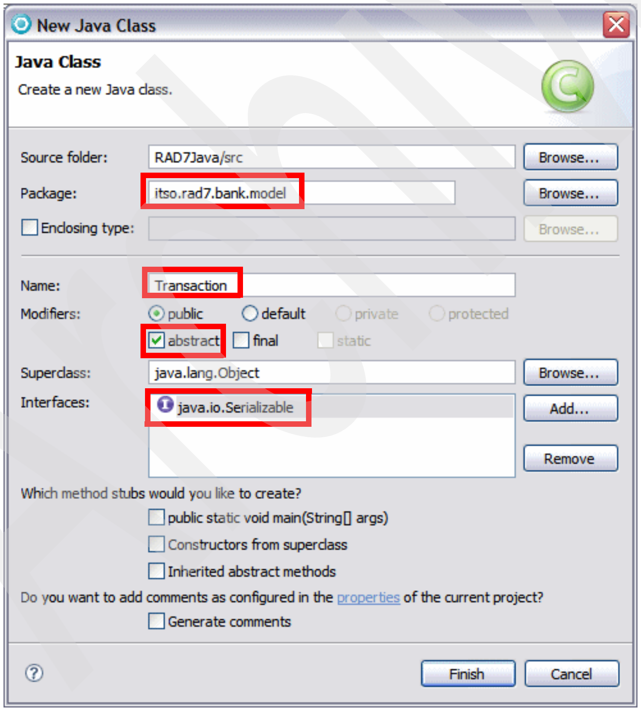
The interfaces and classes are described in “Interfaces and classes overview” on page 236.

> **Note:** The following step-by-step guide explains how to create the `Transaction` class as an example.

Create a Java class in the Visualizer Class Diagram editor

To create a Java class in the Visualizer Class Diagram editor, do these steps:

- Select Class in the Java Drawer and click anywhere in class diagram editor. The New Java Class wizard opens.
- In the New Java Class dialog, enter the following data (Figure 7-18).



Source folder: RAD7Java/src

Package: itso.rad7.bank.model

□ Enclosing type:

Name: Transaction

Modifiers: ● public ○ default ○ private ○ protected

☑ abstract □ final □ static

Superclass: java.lang.Object

Interfaces: java.io.Serializable

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

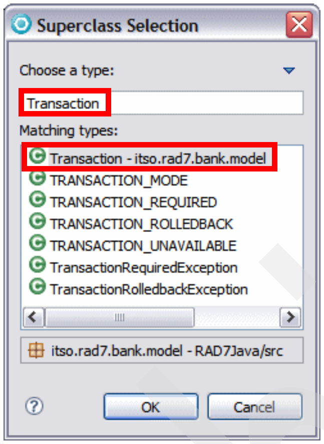
□ Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

□ Generate comments

Figure 7-18 New Java Class dialog

- Package: `itso.rad7.bank.model` (click **Browse** to select the package)
- Name: `Transaction`
- Modifiers: Select **public** (default) and **abstract**
- Superclass: `java.lang.Object` (default). To change the superclass, click **Browse** and overwrite `java.lang.Object` in the Choose a type field with the name of the required superclass. All matching types are listed. Select the required one and click **OK** to leave the dialog (Figure 7-19).


Superclass Selection

Choose a type:

Transaction

Matching types:

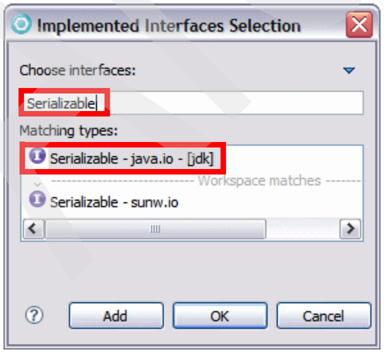
Transaction - itso.rad7.bank.model
TRANSACTION_MODE
TRANSACTION_REQUIRED
TRANSACTION_ROLLEDBACK
TRANSACTION_UNAVAILABLE
TransactionRequiredException
TransactionRolledbackException

itso.rad7.bank.model - RAD7Java/src

Figure 7-19 Superclass Selection dialog

- Interfaces: `java.io.Serializable`

  To add an interface, click **Add** and type the interface name in the Choose interfaces field. All matching types are listed. Select the required interface and click **Add**. If you have added all required interfaces, click **OK** to leave the dialog (Figure 7-20).


Implemented Interfaces Selection

Choose interfaces:

Serializable

Matching types:

Serializable - java.io - [jdk]
------------------------ Workspace matches -------
Serializable - sunw.io

Figure 7-20 Implemented Interfaces Selection dialog

- Which method stubs would you like to create: clear all:
  - `public static void main(String[] args)`—Adds an empty main method to the class and makes the class an executable one. In the example, only the class `BankClient` must be executable.
  - Constructors from superclass—Copies the constructors from the superclass to the new class.
  - Inherited abstract methods—Adds to the new class stubs of any abstract methods from superclasses or methods of interfaces that need to be implemented. In the example, it is useful for the classes `ITSOBank`, `Credit`, and `Debit`.
- Generate comments: clear (default)

- Click **Finish** to create the Java class. A line is drawn from the package to the new class.

Create a Java class using the New Java Class wizard

To create a Java class using the New Java Class wizard:

- Right-click the appropriate package and select **New** → **Class** or click in the toolbar. Then enter the data into the dialog (Figure 7-18 on page 247) and click **Finish**.
- Add the class to the class diagram using drag&drop or select **Visualize** → **Add to Current Diagram**.

> **Note:** Add all required Java classes according to Table 7-2 on page 246. Method and field declarations are added later. Alternatively you can wait and import the finished classes later.
>
> To continue with the instructions we require only two classes:
>
> - `Customer` class in `itso.rad7.bank.model`
> - `InvalidCustomerException` in `itso.rad.7.bank.exception`

Creating Java attributes (fields) and getter and setter methods

This section describes how to add Java attributes (fields) and getter and setter methods to the interfaces and classes of the ITSO Bank application.

Table 7-3 lists the fields of the interfaces, and Table 7-4 lists the attributes and getter and setter methods of the classes which must be created.

Table 7-3 Constant fields of the TransactionType interface

Interface	Field name	Type	Initial value	Visibility, Modifiers
TransactionType	CREDIT	String	”CREDIT”	public static final
	DEBIT	String	”DEBIT”	public static final

Table 7-4 Attributes and getter and setter methods of the classes

Class	Attribute name	Type	Initial value	Visibility, Modifiers	Methods
ITSOBank	accounts *)	Map <String, Account>	new HashMap <String, Account>()	private	getter: public, setter: private
	customers *)	Map <String, Customer>	new HashMap <String, Customer>()		
	customer Accounts	Map <String, ArrayList<Account>>	new HashMap <String, ArrayList <Account>>()		
	bank	ITSOBank	new ITSOBank()	private static	getter: public static
Account	accountNumber	String	null	private	getter: public, setter: private
	balance	BigDecimal			
	transactions *)	ArrayList <Transaction>			

Class	Attribute name	Type	Initial value	Visibility, Modifiers	Methods
Customer	ssn	String	null	private	getter: public, setter: private
	title				
	firstName				
	lastName				
	accounts *)	ArrayList<Account>			
Transaction	timeStamp	Timestamp	null	private	
	amount	BigDecimal			
	transactionId	int	0		
AccountAlready ExistException	accountNumber	String	null	private	
CustomerAlready ExistException	ssn	String	null	private	
InvalidAccount Exception	accountNumber	String	null	private	
InvalidAmount Exception	amount	String	null	private	
InvalidCustomer Exception	ssn	String	null	private	
InvalidTransaction Exception	transactionType	String	null	private	
	amount	BigDecimal			
	account	Account			

[*] These attributes are in fact the implementations of UML class associations. So you can also create them by following the steps described in “Creating relationships between Java types” on page 264.

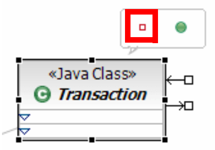
Create an attribute in the Visualizer Class Diagram editor

There are two ways to add a field to an interface or a class. You can write the field declaration direct into the interface or class body in the Java editor, or you can add it in the Visualizer Class Diagram editor. The Visualizer Class Diagram editor calls the Create Java Field wizard to complete the work.

> **Note:** The Create Java Field wizard gets only called through the Visualizer Class Diagram editor. It is not possible to call it by yourself, for example, by clicking an icon in the toolbar.

To create a field in the Visualizer Class Diagram editor, do these steps:

- Right-click the interface or class in the diagram editor and select **Add Java → Field**, or just move the mouse pointer anywhere over the interface or class in the diagram editor, and select □ in the action bar which appears above the class:



- In the Create Java Field dialog, enter the following data (Figure 7-21).
  - Name: `timeStamp`
  - Type: `java.sql.Timestamp`

> **Note:** To change the type, click ⌄, to change the type to any primitive type or to any reference type of the `java.lang.*` package, or click **Browse** and type the interface or class name in the Pick a class or interface field. All matching types are listed and you select the required class and click **OK** to leave the dialog.
>
> All required import statements are also added to the source code.

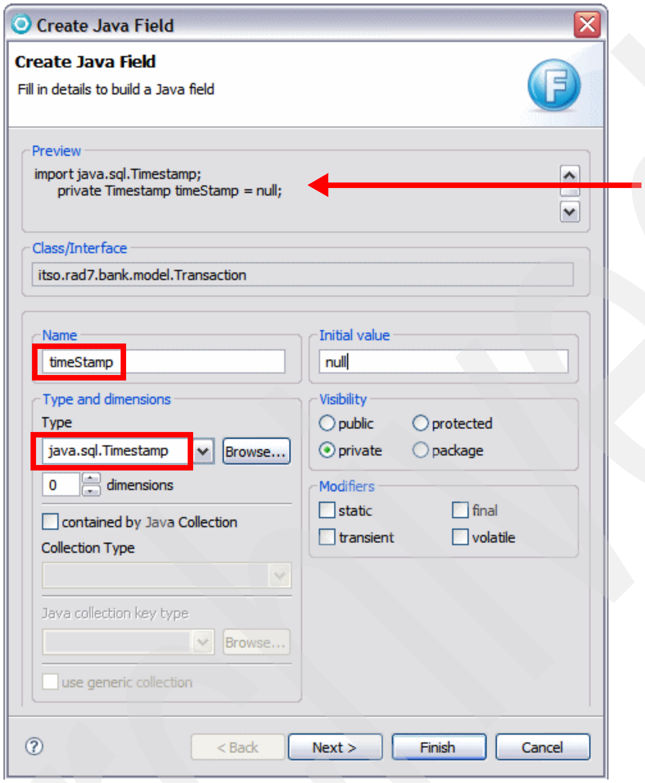
  - Dimensions: 0 (default)—Changing the value of this field is for creating an array of the selected type with the selected dimension.
  - Contained by Java Collection: clear (default)

> **Note:** If the required attribute has a multiplicity higher than 1, select this field. If selected, the wizard allows you to select the required Java collection class. If you select any kind of `Map` class, you can select the type of the key in the Java collection key type field. Finally, you can create parameterized types by selecting the use generic collection checkbox. More information about generic types can be found here:
>
> `http://java.sun.com/developer/technicalArticles/J2SE/generics/`

– Initial value: `null`

– Visibility: `private` (default)

– Modifiers: clear all (default)



Create Java Field

Fill in details to build a Java field

Preview

```
import java.sql.Timestamp;
    private Timestamp timeStamp = null;
```

Class/Interface: itso.rad7.bank.model.Transaction

Name: timeStamp

Initial value: null

Type and dimensions

Type: java.sql.Timestamp

0 dimensions

□ contained by Java Collection

Collection Type

Java collection key type

□ use generic collection

Visibility: ○ public ○ protected ● private ○ package

Modifiers: □ static □ final □ transient □ volatile

The Preview field shows the source code that is created

Figure 7-21 Create Java Field dialog

- Click **Finish** to create the Java field.

> **Note:** There are two reasons why you might not see the attributes in the class diagram:
>
> - Class diagram attribute compartment is collapsed—Select the interface or class and click the little blue arrow in the compartment in the middle. That expands the attribute compartment.
> - Class diagram attribute compartment is filtered out—Right-click the interface or class and select **Filters** → **Show/Hide Compartment** → **Attribute Compartment**.

Create getter and setter methods using refactor feature

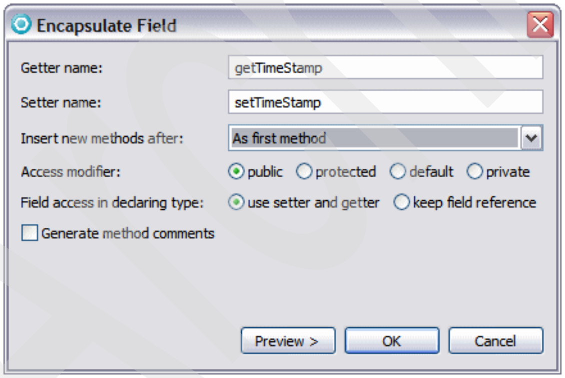
This section describes how to generate getter and setter methods for Java attributes by using the refactor feature of Application Developer.

To generate getter and setter methods for a Java attribute using the refactor feature, do these steps:

- Right-click the attribute in the diagram editor or in the Outline view and select **Refactor → Encapsulate Field**.
- In the Encapsulate Field dialog, enter the following data (Figure 7-22):
  - Getter name: `getTimeStamp` (default)
  - Setter name: `setTimeStamp` (default)
  - Insert new methods after: As first method (default)
  - Access modifier: select `public`

    You can later change the access modifier of the setter method to `private` in the source code.
  - Field access in declaring type: use getter and setter (default)

    It is good programming style when you use the getter and setter method also internally in the class to access member variables.
  - Generate method comments: clear (default)


Encapsulate Field

Getter name: getTimeStamp

Setter name: setTimeStamp

Insert new methods after: As first method

Access modifier: ● public ○ protected ○ default ○ private

Field access in declaring type: ● use setter and getter ○ keep field reference

☐ Generate method comments

Figure 7-22 Encapsulate Field dialog

- Click **OK** to generate the getter and setter methods.

Create getter and setter methods using source feature

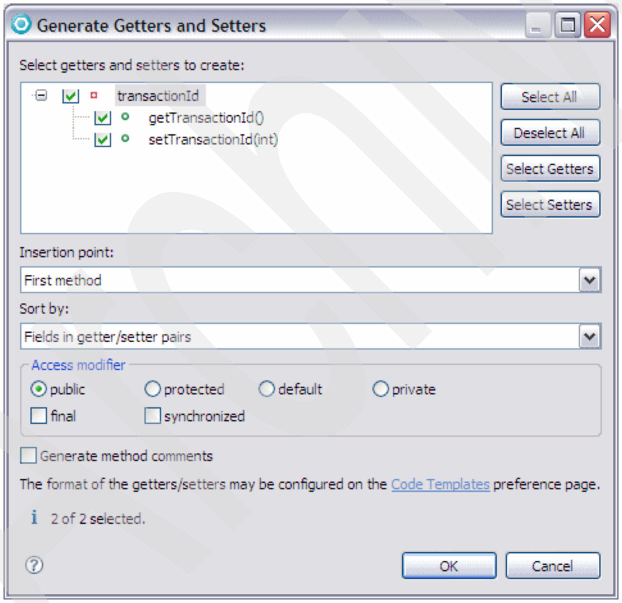
This section describes how to generate getter and setter methods for Java attributes by using the source feature of Application Developer.

To generate getter and setter methods for a Java attribute using the source feature, do these steps:

- Create a field in the `Transaction` class:
  - Name: `transactionId`
  - Type: int
  - Initial value: 0
- Right-click the attribute in the diagram editor or in the Outline view and select **Source** → **Generate Getters and Setters**.

> **Note:** If the source code is open in the Java editor, you can just right-click somewhere in the Java editor and select **Source** → **Generate Getters and Setters**, or you can select **Source** → **Generate Getters and Setters** in the menu bar.

- The Generate Getters and Setters dialog is shown in Figure 7-23. (Below this figure, we list the data for you to enter.)



Generate Getters and Setters

Select getters and setters to create:

☑ transactionId
  ☑ getTransactionId()
  ☑ setTransactionId(int)

Insertion point: First method

Sort by: Fields in getter/setter pairs

Access modifier: ● public ○ protected ○ default ○ private

□ final □ synchronized

□ Generate method comments

The format of the getters/setters may be configured on the Code Templates preference page.

2 of 2 selected.

Figure 7-23 Generate Getters and Setters dialog

- In the Generate Getters and Setters dialog, enter the following data:
  - Select getters and setters to create:

    `getTransactionId` and `setTransactionId(int)` (default)
  - Insertion point: Last method (default)
  - Sort by: First getters, then setters (default)
  - Access modifier: `public` (default)

    You can later change the access modifier of the setter method to `private` in the source code.
  - Generate method comments: clear (default)
- Click **OK** to generate the getter and setter methods.

> **Note:** It does not matter in which way you are creating the getter and setter methods. As soon as you save the changes, they are visible in the class diagram as well as in the source code.

Adding method declarations to an interface

This section describes how to add method declarations to the `Bank` interface of the ITSO Bank application. Table 7-5 lists all method declarations that must be created.

Table 7-5 Method declarations of the `Bank` interface

Method name	Return type	Parameters	Exceptions
addCustomer	void	Customer customer	CustomerAlreadyExistException
updateCustomer	void	String ssn, String title, String firstName, String lastName	InvalidCustomerException
removeCustomer	void	Customer customer	InvalidCustomerException
openAccountForCustomer	void	Customer customer, Account account	InvalidCustomerException, AccountAlreadyExistException
closeAccountOfCustomer	void	Customer customer, Account account	InvalidAccountException, InvalidCustomerException
searchCustomerBySsn	Customer	String ssn	InvalidCustomerException
searchAccountByAccountNumber	Account	String accountNumber	InvalidAccountException
getCustomers	Map<String, Customer>		

Method name	Return type	Parameters	Exceptions
getAccountsForCustomer	ArrayList <Account>	String customerSsn	InvalidCustomerException
getTransactionsForAccount	ArrayList <Transaction>	String accountNumber	InvalidAccountException
deposit	void	String accountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException
withdraw	void	String accountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException
transfer	void	String debitAccountNumber, String creditAccountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException

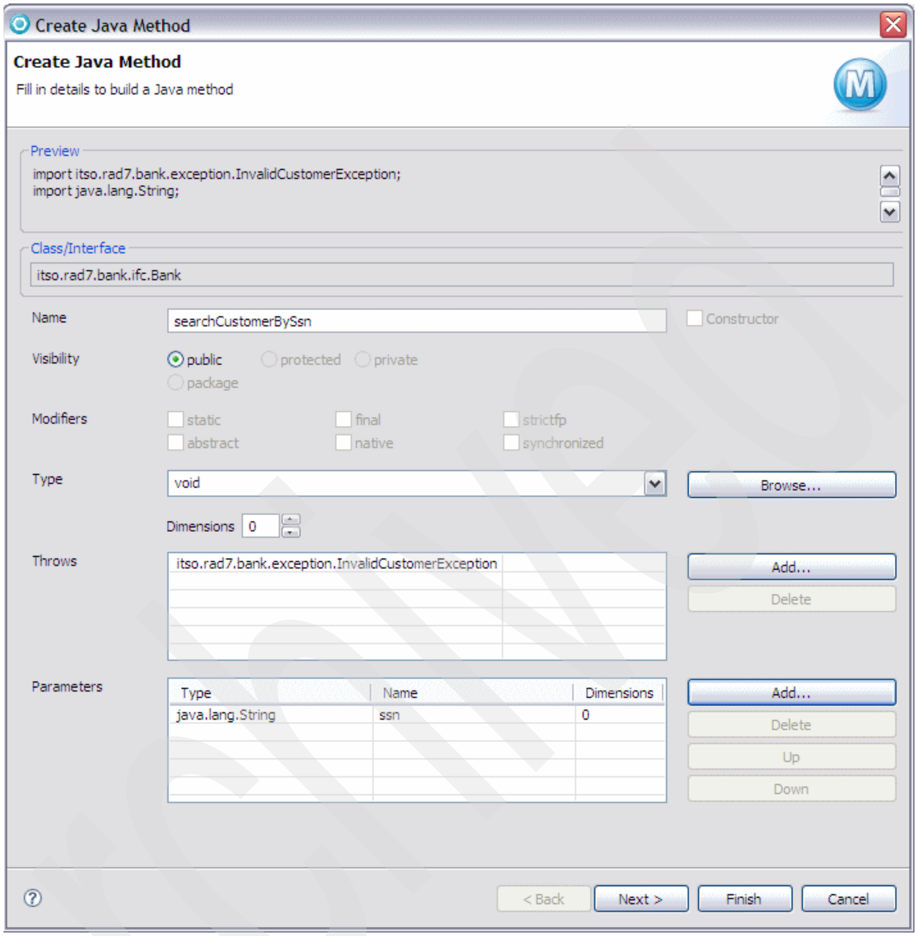
There are two ways to add a method declaration to an interface. You can write the method declaration directly into the interface body in the Java editor, or you can add it in the Visualizer Class Diagram editor. The Visualizer Class Diagram editor calls the Create Java Method wizard to complete the work.

Note: The Create Java Method wizard only gets called through the Visualizer Class Diagram editor. It is not possible to call it by yourself (for example, by clicking an icon in the toolbar).

To create a method in the Visualizer Class Diagram editor, do these steps:

- Right-click the `Bank` interface in the diagram editor and select **Add Java → Method**, or just move the mouse pointer anywhere over the interface in the diagram editor, and click ● in the action bar above the class.

- In the Create Java Method dialog, enter the following data (Figure 7-24).



Create Java Method

Fill in details to build a Java method

Preview

```
import itso.rad7.bank.exception.InvalidCustomerException;
import java.lang.String;
```

Class/Interface: itso.rad7.bank.ifc.Bank

Name: searchCustomerBySsn □ Constructor

Visibility: ● public ○ protected ○ private ○ package

Modifiers: □ static □ final □ strictfp □ abstract □ native □ synchronized

Type: void

Dimensions: 0

Throws: itso.rad7.bank.exception.InvalidCustomerException

Parameters:

Type	Name	Dimensions
java.lang.String	ssn	0

Figure 7-24 Create Java Method dialog

– Name: searchCustomerBySsn

– Visibility: public (default)

– Modifiers: clear all (default)

> **Restriction:** All methods of a Java interface are `public abstract`. Modifier `abstract` can be omitted, as it is per default `abstract`. Therefore, there is no choice by Visibility and Modifiers when you are adding a method declaration to an interface. An interface has never a constructor, so the constructor check box is never active.

  - Type: void (default)
  - Dimensions: 0 (default)
  - Throws:

    `itso.rad7.bank.exception.InvalidCustomerException`

    > **Note:** To add an exception, click **Add** and type the exception class name in the Pick one or more exception types to throw field. All matching types are listed in the Matching types field. Select the required exceptions and click **OK**.

  - Parameters: `string ssn`

    > **Note:** To add a parameter, click **Add**. Enter the name, select the type and dimensions in the Create Parameter dialog, and click **OK** to add the parameter. In the example, we do not pass any array parameters, and dimensions are always 0 (default).

- Click **Finish** to create the Java method.

> **Note:** There are two reasons why you might not see the methods in the class diagram:
>
> - Class diagram method compartment is collapsed—Select the interface or class and click the little blue arrow in the compartment in the bottom. That expands the method compartment.
> - Class diagram method compartment is filtered out—Right-click the interface or class and select **Filters** → **Show/Hide Compartment** → **Method Compartment**.

Adding constructors and Java methods to a class

The way to add constructors and methods to a class is the same as when you add a method declaration to an interface, as explained in “Adding method declarations to an interface” on page 256, except that there are no restrictions as described for the interfaces.

Table 7-6 lists all constructors and methods that must be added to the classes of the ITSO Bank application. Be aware that you have to select **Constructor** when adding a constructor to a class.

Table 7-6 Constructors and methods of the classes of the ITSO Bank application

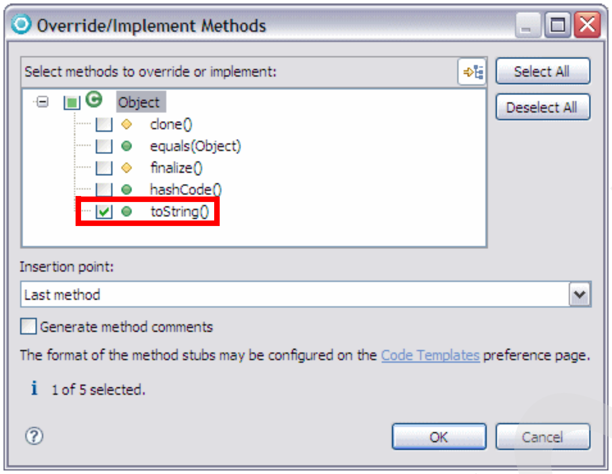
Method name	Modi-fiers	Type	Parameters	Exceptions
ITSOBank				
ITSOBank	private constructor	-		
addCustomer	public	void	Customer customer	CustomerAlreadyExistException
removeCustomer		void	Customer customer	InvalidCustomerException
openAccountForCustomer		void	Customer customer, Account account	InvalidCustomerException, AccountAlreadyExistException
closeAccountOfCustomer		void	Customer customer, Account account	InvalidAccountException, InvalidCustomerException
searchAccountByAccountNumber		Account	String accountNumber	InvalidAccountException
searchCustomerBySsn		Customer	String ssn	InvalidCustomerException
processTransaction	private	void	String accountNumber, BigDecimal amount, String transactionType	InvalidAccountException, InvalidTransactionException
getAccountsForCustomer	public	ArrayList<Account>	String customerSsn	InvalidCustomerException
getTransactionsForAccount		ArrayList<Transaction>	String accountNumber	InvalidAccountException
updateCustomer		void	String ssn, String title, String firstName, String lastName	InvalidCustomerException
deposit		void	String accountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException
withdraw		void	String accountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException
transfer		void	String debitAccountNumber, String creditAccountNumber, BigDecimal amount	InvalidAccountException, InvalidTransactionException

Method name	Modi-fiers	Type	Parameters	Exceptions
initializeBank	private	void		
Account				
Account	public constructor	-	String accountNumber, BigDecimal balance	
processTransaction	public	void	BigDecimal amount, String transactionType	InvalidTransactionException
toString		String		
Customer				
Customer	public constructor	-	String ssn, String title, String firstName, String lastName	
updateCustomer	public	void	String title, String firstName, String lastName	
addAccount		void	Account account	AccountAlreadyExistException
removeAccount		void	Account account	InvalidAccountException
toString		String		
Transaction				
Transaction	public constructor	-	BigDecimal amount	
getTransactionType	public abstract	String		
process		BigDecimal	BigDecimal accountBalance	InvalidTransactionException
Credit				
Credit	public constructor	-	BigDecimal amount	

Method name	Modi-fiers	Type	Parameters	Exceptions
getTransactionType	public	String		
process		BigDecimal	BigDecimal accountBalance	InvalidTransactionException
toString		String		
Debit				
Debit	public constructor	-	BigDecimal amount	
getTransactionType	public	String		
process		BigDecimal	BigDecimal accountBalance	InvalidTransactionException
toString		String		
BankClient				
main	public static	void	String[] args	

Tip: If you want to add a method to a class that implements or overrides an existing method in an interface or a superclass, there is a much faster way to add it than by using the Create Java Method wizard. Use the source feature Override/Implement Methods:

- Right-click the class in the diagram editor or in the Package Explorer and select **Source** → **Override/Implement Methods**.
- In the Override/Implement Methods dialog, all methods that can be implemented or overridden by this class are listed. Select the methods you want to override or implement and click **OK** to add the method stubs to the selected class.
- For example, you can add the `toString` method to the Transaction class (Figure 7-25).



Override/Implement Methods

Select methods to override or implement:

- Object
  - ☐ clone()
  - ☐ equals(Object)
  - ☐ finalize()
  - ☐ hashCode()
  - ☑ toString()

Insertion point: Last method

☐ Generate method comments

The format of the method stubs may be configured on the Code Templates preference page.

1 of 5 selected.

Figure 7-25 Override/Implement Methods dialog

Implementing the classes and methods

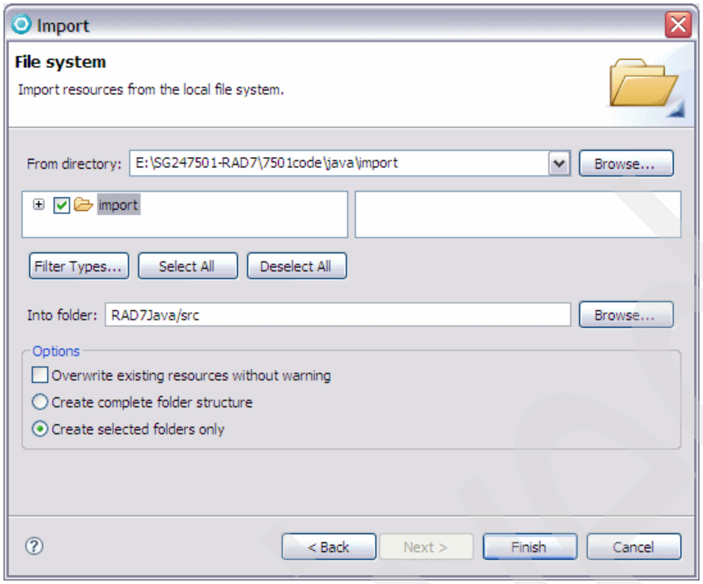
In the previous sections of the ITSO Bank application example, we have included step-by-step approaches with the objective of demonstrating the Application Developer tooling and the logical process of developing a Java application. In this section we import all the classes with the method code.

The Java editor of the Application Developer provides a set of useful features to develop the code. These features are explained in “Additional features used for Java applications” on page 282 and “Java editor and rapid application development” on page 300.

Import the classes

to import the classes:

- Right-click the `src` folder and select **Import**.
- Select **General File System**, then click **Browse** and navigate to the `C:\7501code\java\import` folder. Refer to the note on page 236 for downloading the sample code.
- Select the **import** folder and click **Finish** (Figure 7-26).



File system

Import resources from the local file system.

From directory: E:\SG247501-RAD7\7501code\java\import

☑ import

Into folder: RAD7Java/src

Options

□ Overwrite existing resources without warning

○ Create complete folder structure

● Create selected folders only

Figure 7-26 Importing the classes

- You can add all the classes to the diagram manually, or import the diagram into the `diagram` folder from:

```
C:\7501code\java\diagram\ITSOBank-Diagram.dnx
```

- To change the appearance of the diagram, right-click in the diagram and select **Filters** → **Show/Hide Connector Labels** → **All** or **No connector Labels**, or **Filters** → **Show/Hide Relationships** and select the relationships to be displayed or hidden. For example, you can hide the many `<<use>>` relationships.

Creating relationships between Java types

In Application Developer it is possible to model the relationships between Java types in the Visualizer Class Diagram editor. This section includes the following topics:

- Extends relationship
- Implements relationship
- Association relationship

Extends relationship

To create an *extends* relationship between existing classes, select Extends in the Java Drawer and drag the mouse with the left mouse button down from any point on the child class to the parent class.

There are already extends relationships in the class diagram: The `Transaction` class is specified as the superclass of the `Credit` and `Debit` classes. Extends relationships are displayed using a solid line with a triangular arrow pointing to the superclass.

Implements relationship

To create an *implements* relationship between an existing class and an interface select Implements in the Java Drawer and drag the mouse with the left mouse button down from any point in the implementation class to the interface.

There is already an implements relationship in the class diagram: The `ITSOBank` class implements the `Bank` interface. The implements relationship is displayed using a dashed line with a triangular arrow pointing to the interface.

Association relationship

The classes in the ITSO Bank application have the following relationships:

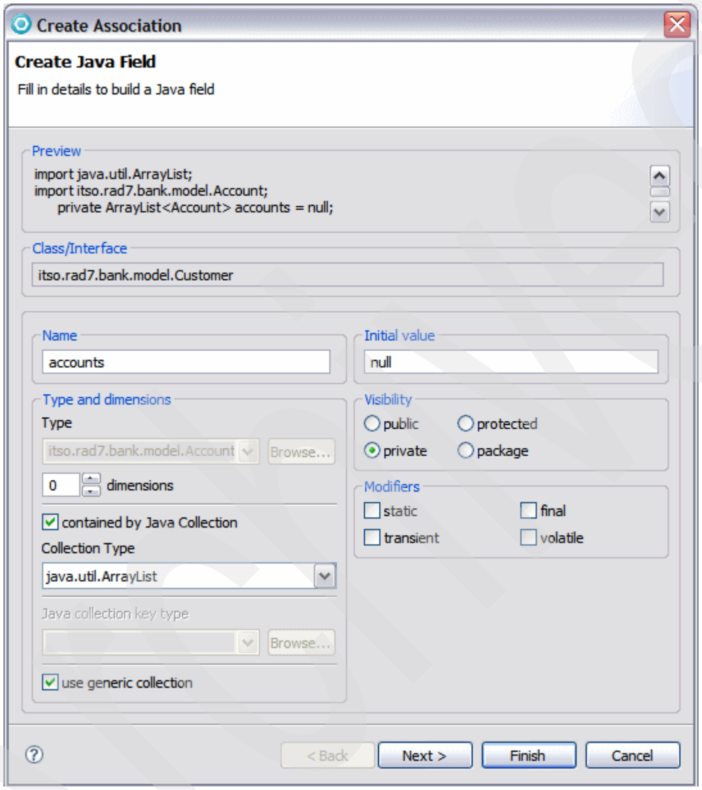
- `ITSOBank` remembers the customers and accounts.
- A customer knows his or her accounts.
- An account logs all the transactions for logging and querying purposes.

To create an association relationship between to classes, do these steps:

> **Note:** The following step-by-step guide explains how to create the association between `Customer` class and the `Account` class. The associations are listed in Table 7-4 on page 250.
>
> This is an example of how to define an association manually. All the associations are already defined after importing the classes.

- Select Association in the Java drawer.
- Drag the mouse with the left mouse button down from any point on the `Customer` class to the `Account` class.
- In the Create Association dialog, enter the following data (Figure 7-27).
  - Name: `accounts`
  - Type: - (not active)
  - Dimensions: 0 (default)

- Contained by Java Collection: select
- Collection type: `java.util.ArrayList`
- Java Collection key type: - (not active)
- Use generic collection: select
- Initial value: `null`
- Visibility: `private` (default)
- Modifiers: clear all (default)
- Click **Finish** to create the association.



Create Association

Create Java Field

Fill in details to build a Java field

Preview

```
import java.util.ArrayList;
import itso.rad7.bank.model.Account;
    private ArrayList<Account> accounts = null;
```

Class/Interface: itso.rad7.bank.model.Customer

Name: accounts

Initial value: null

Type and dimensions

Type: itso.rad7.bank.model.Account

0 dimensions

☑ contained by Java Collection

Collection Type: java.util.ArrayList

Java collection key type:

☑ use generic collection

Visibility: ○ public ○ protected ● private ○ package

Modifiers: □ static □ final □ transient □ volatile

Figure 7-27 Create Association dialog

Note: An association can be displayed as an arrow or an attribute:

- Show as **attribute**—To display the association as an attribute, right-click the association arrow and select **Filters** → **Show As Attribute**.
- Show as **association arrow**—To display the attribute as an association, right-click the attribute and select **Filters** → **Show As Association**.

Understanding the sample code

In this section we show an extract of the sample code. You can study all the sample code after importing it into the project.

> **Note:** We do not show the generated getter and setter methods because they are trivial.

Customer class

Example 7-1 shows the Java source code of the Customer class.

Example 7-1 Customer class

```
package itso.rad7.bank.model;

import itso.rad7.bank.exception.AccountAlreadyExistException;
import itso.rad7.bank.exception.InvalidAccountException;
import java.io.Serializable;
import java.util.ArrayList;

public class Customer implements Serializable {

   private static final long serialVersionUID = 1871790137891426588L;
   private String ssn;
   private String title;
   private String firstName;
   private String lastName;
   private ArrayList<Account> accounts;

   public Customer(String ssn, String title, String firstName,
                                                String lastName) {
      this.setSsn(ssn);
      this.setTitle(title);
      this.setFirstName(firstName);
      this.setLastName(lastName);
      this.setAccounts(new ArrayList<Account>());
   }

   public void addAccount(Account account) throws AccountAlreadyExistException
   {
      if (!this.getAccounts().contains(account)) {
         this.getAccounts().add(account);
      } else {
         throw new AccountAlreadyExistException(account.getAccountNumber());
      }
   }
   public void removeAccount(Account account) throws InvalidAccountException {
      if (this.getAccounts().contains(account)) {
         this.getAccounts().remove(account);
      } else {
         throw new InvalidAccountException(account.getAccountNumber());
      }
   }

   public void updateCustomer(String title, String firstName, String lastName)
   {
      this.setTitle(title);
      this.setFirstName(firstName);
      this.setLastName(lastName);
   }

   // Getter and setter methods are omitted

   public String toString() {
      return this.getSsn() + " " + this.getTitle() + " " + this.getLastName()
             + " " + this.getFirstName();
   }
}
```

Account class

Example 7-2 shows the Java source code of the `Account` class.

Example 7-2 Account class

```
package itso.rad7.bank.model;

import itso.rad7.bank.exception.InvalidTransactionException;
import itso.rad7.bank.ifc.TransactionType;
import java.io.Serializable;
import java.math.BigDecimal;
import java.util.ArrayList;

public class Account implements Serializable {

   private static final long serialVersionUID = -1682730430512221805L;
   private String accountNumber;
   private BigDecimal balance;
   private ArrayList<Transaction> transactions;

   public Account(String accountNumber, BigDecimal balance) {
      this.setAccountNumber(accountNumber);
      this.setBalance(balance);
      this.setTransactions(new ArrayList<Transaction>());
   }
   // Getter and setter methods are omitted

   public void processTransaction(BigDecimal amount, String transactionType)
         throws InvalidTransactionException {

      BigDecimal newBalance = null;
      Transaction transaction = null;

      if (TransactionType.CREDIT.equals(transactionType)) {
         transaction = new Credit(amount);
      } else if (TransactionType.DEBIT.equals(transactionType)) {
         transaction = new Debit(amount);
      } else {
         throw new InvalidTransactionException(
               "Invalid transaction type: Please use Debit or Credit. "
                     + "No other transactions are currently supported.");
      }

      newBalance = transaction.process(this.getBalance());
      if (newBalance.doubleValue() < 0) {
         throw new InvalidTransactionException(this, transactionType, amount);
      } else {
         this.setBalance(newBalance);
         this.getTransactions().add(transaction);
      }
   }

   public String toString() {
      StringBuffer accountInfo = new StringBuffer();

      accountInfo.append("Account " + this.getAccountNumber()
            + ": --> Current balance: $"
            + this.getBalance().setScale(2, BigDecimal.ROUND_HALF_EVEN));
      if (this.getTransactions().size() > 0) {
         accountInfo.append(System.getProperty("line.separator"));
         accountInfo.append("\tTransactions: ");
         accountInfo.append(System.getProperty("line.separator"));
         for (int i = 0; i < this.getTransactions().size(); i++) {
            accountInfo.append("\t" + (i + 1) + ". "
                  + this.getTransactions().get(i) + "\n");
         }
      }
      return accountInfo.toString();
   }
}
```

Transaction class

Example 7-3 shows the Java source code of the Transaction class.

Example 7-3 Transaction class

```
package itso.rad7.bank.model;

import itso.rad7.bank.exception.InvalidTransactionException;
import java.io.Serializable;
import java.math.BigDecimal;
import java.sql.Timestamp;

public abstract class Transaction implements Serializable {

   private Timestamp timeStamp = null;
   private BigDecimal amount = null;
   private int transactionId = 0;

   public Transaction(BigDecimal amount) {
      this.setTimeStamp(new Timestamp(System.currentTimeMillis()));
      this.setAmount(amount);
      this.setTransactionId();
   }

   // Getter and setter methods are omitted

   private void setTransactionId() {
      this.transactionId += 1;
   }

   public abstract BigDecimal process(BigDecimal accountBalance)
         throws InvalidTransactionException;

   public abstract String getTransactionType();
}
```

Credit class

Example 7-4 shows the Java source code of the Credit class.

Example 7-4 Credit class

```
package itso.rad7.bank.model;

import itso.rad7.bank.exception.InvalidTransactionException;
import itso.rad7.bank.ifc.TransactionType;
import java.math.BigDecimal;

public class Credit extends Transaction {
   private static final long serialVersionUID = 5827376979569343797L;

   public Credit(BigDecimal amount) {
      super(amount);
   }

   public BigDecimal process(BigDecimal accountBalance)
         throws InvalidTransactionException {
      if ((this.getAmount() != null)
            && (this.getAmount().compareTo(new BigDecimal(0.00D)) > 0)) {
         return accountBalance.add(this.getAmount());
      } else {
         if (this.getAmount() != null) {
            throw new InvalidTransactionException(
                  "Credit transaction could not proceed: "
                        + "Negative or zero credit amount has dedected. "
                        + "Credit amount is: $"
                        + this.getAmount().setScale(2,
                              BigDecimal.ROUND_HALF_EVEN) + ".");
         } else {
            throw new InvalidTransactionException(
                  "Credit transaction could not proceed: "
                        + "Credit amount is not set.");
         }
      }
   }

   public String getTransactionType() {
      return TransactionType.CREDIT;
   }

   public String toString() {
      if (this.getAmount() != null) {
         if (this.getTimeStamp() != null) {
            return "Credit: --> Amount $"
                  + this.getAmount().setScale(2,
                        BigDecimal.ROUND_HALF_EVEN) + " on "
                  + this.getTimeStamp();
         } else {
            return "Amount that will be credited to the account is $"
                  + this.getAmount().setScale(2,
                        BigDecimal.ROUND_HALF_EVEN);
         }
      } else {
         return "Credit amount is not set. Transaction has failed.";
      }
   }
}
```

Debit class

The `Debit` class is similar to the `Credit` class, but subtracts the amount to the balance.

ITSOBank class

Example 7-5 shows the Java source code of the `ITSOBank` class.

Example 7-5 ITSOBank class

```
package itso.rad7.bank.impl;

import itso.rad7.bank.exception.AccountAlreadyExistException;
import itso.rad7.bank.exception.CustomerAlreadyExistException;
import itso.rad7.bank.exception.ITSOBankException;
import itso.rad7.bank.exception.InvalidAccountException;
import itso.rad7.bank.exception.InvalidCustomerException;
import itso.rad7.bank.exception.InvalidTransactionException;
import itso.rad7.bank.ifc.Bank;
import itso.rad7.bank.ifc.TransactionType;
import itso.rad7.bank.model.Account;
import itso.rad7.bank.model.Customer;
import itso.rad7.bank.model.Transaction;
import java.math.BigDecimal;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.Iterator;
import java.util.Map;

public class ITSOBank implements Bank {

   private Map<String, Customer> customers = null;
   private Map<String, Account> accounts = null;
   private Map<String, ArrayList<Account>> customerAccounts = null;
   private static Bank bank = new ITSOBank();

   private ITSOBank() {
      this.setCustomers(new HashMap<String, Customer>());
      this.setAccounts(new HashMap<String, Account>());
      this.setCustomerAccounts(new HashMap<String, ArrayList<Account>>());
      this.initializeBank();
   }

   public static Bank getBank() {
      return ITSOBank.bank;
   }

   public void addCustomer(Customer customer)
         throws CustomerAlreadyExistException {
```

```
   if (this.getCustomers().get(customer.getSsn()) == null) {
      this.getCustomers().put(customer.getSsn(), customer);
      this.getCustomerAccounts().put(customer.getSsn(),
            new ArrayList<Account>());
   } else {
      throw new CustomerAlreadyExistException(customer.getSsn());
   }
}

public void removeCustomer(Customer customer)
      throws InvalidCustomerException {
   String customerSsn = customer.getSsn();
   if (this.getCustomers().get(customerSsn) != null) {
      ArrayList<Account> accountList = this
            .getAccountsForCustomer(customerSsn);
      for (Iterator iter = accountList.iterator(); iter.hasNext();) {
         try {
            this
                  .closeAccountOfCustomer(customer, (Account) iter
                        .next());
         } catch (InvalidAccountException iae) {
            iae.printStackTrace();
         }
      }
      this.getCustomerAccounts().remove(customerSsn);
      this.getCustomers().remove(customerSsn);
   } else {
      throw new InvalidCustomerException(customerSsn);
   }
}

public void openAccountForCustomer(Customer customer, Account account)
      throws InvalidCustomerException, AccountAlreadyExistException {
   String customerSsn = customer.getSsn();
   String accountNumber = account.getAccountNumber();

   if (this.getCustomers().get(customerSsn) != null) {
      if (this.getAccounts().get(accountNumber) == null) {
         this.getAccounts().put(account.getAccountNumber(), account);
         this.getCustomerAccounts().get(customerSsn).add(account);
         customer.addAccount(account);
      } else {
         throw new AccountAlreadyExistException(accountNumber);
      }
   } else {
      throw new InvalidCustomerException(customerSsn);
   }
}
```

```
public void closeAccountOfCustomer(Customer customer, Account account)
      throws InvalidAccountException, InvalidCustomerException {
   String customerSsn = customer.getSsn();
   String accountNumber = account.getAccountNumber();

   if (this.getCustomers().get(customerSsn) != null) {
      if (this.getAccounts().get(accountNumber) != null) {
         this.getAccounts().remove(accountNumber);
         this.getCustomerAccounts().get(customerSsn).remove(
               accountNumber);
         customer.removeAccount(account);
      } else {
         throw new InvalidAccountException(accountNumber);
      }
   } else {
      throw new InvalidCustomerException(customerSsn);
   }
}

public Account searchAccountByAccountNumber(String accountNumber)
      throws InvalidAccountException {
   Account account = this.getAccounts().get(accountNumber);
   if (account != null) {
      return account;
   } else {
      throw new InvalidAccountException(accountNumber);
   }
}

public Customer searchCustomerBySsn(String ssn)
      throws InvalidCustomerException {
   Customer customer = this.getCustomers().get(ssn);
   if (customer != null) {
      return customer;
   } else {
      throw new InvalidCustomerException(ssn);
   }
}

private void processTransaction(String accountNumber, BigDecimal amount,
      String transactionType) throws InvalidAccountException,
      InvalidTransactionException {
   Account account = this.searchAccountByAccountNumber(accountNumber);
   account.processTransaction(amount, transactionType);
}

public ArrayList<Account> getAccountsForCustomer(String customerSsn)
      throws InvalidCustomerException {
   if (this.getCustomers().get(customerSsn) != null) {
      return this.getCustomerAccounts().get(customerSsn);
   } else {
      throw new InvalidCustomerException(customerSsn);
   }
}

public ArrayList<Transaction> getTransactionsForAccount
                                     (String accountNumber)
      throws InvalidAccountException {
   Account account = this.getAccounts().get(accountNumber);
   if (account != null) {
      return account.getTransactions();
   } else {
      throw new InvalidAccountException(accountNumber);
   }
}

public void updateCustomer(String ssn, String title, String firstName,
      String lastName) throws InvalidCustomerException {
   this.searchCustomerBySsn(ssn)
         .updateCustomer(title, firstName, lastName);
}

public void deposit(String accountNumber, BigDecimal amount)
      throws InvalidAccountException, InvalidTransactionException {
   this.processTransaction(accountNumber, amount, TransactionType.CREDIT);
}

public void withdraw(String accountNumber, BigDecimal amount)
      throws InvalidAccountException, InvalidTransactionException {
   this.processTransaction(accountNumber, amount, TransactionType.DEBIT);
}

public void transfer(String debitAccountNumber, String creditAccountNumber,
      BigDecimal amount) throws InvalidAccountException,
      InvalidTransactionException {
   this.processTransaction(debitAccountNumber, amount,
         TransactionType.DEBIT);
   this.processTransaction(creditAccountNumber, amount,
         TransactionType.CREDIT);
}

private void initializeBank() {
   try {
      Customer customer1 = new Customer("111-11-1111", "MR", "Henry",
            "Cui");
      this.addCustomer(customer1);
      Customer customer2 = new Customer("222-22-2222", "MS", "Pinar",
            "Ugurlu");
      this.addCustomer(customer2);
      Customer customer3 = new Customer("333-33-3333", "MR", "Marco",
            "Rohr");
      this.addCustomer(customer3);
      Customer customer4 = new Customer("444-44-4444", "MR", "Craig",
            "Fleming");
      this.addCustomer(customer4);

      Account account11 = new Account("001-999000777", new BigDecimal(
            1234567.89D));
      this.openAccountForCustomer(customer1, account11);
      Account account12 = new Account("001-999000888", new BigDecimal(
            6543.21D));
      this.openAccountForCustomer(customer1, account12);
      Account account13 = new Account("001-999000999", new BigDecimal(
            98.76D));
      this.openAccountForCustomer(customer1, account13);

      Account account21 = new Account("002-999000777", new BigDecimal(
            65484.23D));
      this.openAccountForCustomer(customer2, account21);
      Account account22 = new Account("002-999000888", new BigDecimal(
            87.96D));
      this.openAccountForCustomer(customer2, account22);
      Account account23 = new Account("002-999000999", new BigDecimal(
            654.65D));
      this.openAccountForCustomer(customer2, account23);

      Account account31 = new Account("003-999000777", new BigDecimal(
            9876.52D));
      this.openAccountForCustomer(customer3, account31);
      Account account32 = new Account("003-999000888", new BigDecimal(
            568.79D));
      this.openAccountForCustomer(customer3, account32);
      Account account33 = new Account("003-999000999", new BigDecimal(
            21.56D));
      this.openAccountForCustomer(customer3, account33);

      Account account41 = new Account("004-999000777", new BigDecimal(
            987.65D));
      this.openAccountForCustomer(customer4, account41);
      Account account42 = new Account("004-999000888", new BigDecimal(
            1456456.46D));
      this.openAccountForCustomer(customer4, account42);
      Account account43 = new Account("004-999000999", new BigDecimal(
            23156.46D));
      this.openAccountForCustomer(customer4, account43);
   } catch (ITSOBankException e) {
      e.printStackTrace();
      }
   }

   // Getter and setter methods are omitted
}
```

ITSOBankException class

Example 7-6 shows the Java source code of the ITSOBankException class.

Example 7-6 ITSOBankException class

```
package itso.rad7.bank.exception;

public class ITSOBankException extends Exception {

   private static final long serialVersionUID = -5758729545681152548L;

   public ITSOBankException(String message) {
      super(message);
   }
}
```

InvalidTransactionException class

Example 7-7 shows the Java source code of the InvalidTransactionException class.

Example 7-7 InvalidTransactionException class

```
package itso.rad7.bank.exception;

import itso.rad7.bank.model.Account;
import java.math.BigDecimal;

public class InvalidTransactionException extends ITSOBankException {

   private static final long serialVersionUID = -4756665135978155919L;
   private Account account;
   private String transactionType;
   private BigDecimal amount;

   public InvalidTransactionException(String message) {
      super(message);
      this.setTransactionType(null);
      this.setAmount(null);
      this.setAccount(null);
   }
   public InvalidTransactionException(Account account, String transactionType,
          BigDecimal amount) {
      super(transactionType.toLowerCase() + " transaction on account "
             + account.getAccountNumber() + " failed: " + "Could not "
             + transactionType.toLowerCase() + " $"
             + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
             + ", because current balance is $"
             + account.getBalance().setScale(2, BigDecimal.ROUND_HALF_EVEN)
             + " and negative balances are not allowed.");
      this.setTransactionType(transactionType);
      this.setAmount(amount);
      this.setAccount(account);
   }

   // Getter and setter methods are omitted
}
```

AccountAlreadyExistException class

Example 7-8 shows the Java source code of the `AccountAlreadyExistException` class. The other exceptions have similar code and are not listed.

Example 7-8 AccountAlreadyExistException class

```
package itso.rad7.bank.exception;

public class AccountAlreadyExistException extends Exception {

   private static final long serialVersionUID = -5804089601272l0114L;
   private String accountNumber;

   public AccountAlreadyExistException(String accountNumber) {
      super("Invalid account: Account with number " + accountNumber
             + " does already exist!");
      this.setAccountNumber(accountNumber);
   }

   // Getter and setter methods are omitted
}
```

BankClient class

Example 7-9 shows the Java source code of the `BankClient` class.

Example 7-9 BankClient class

```
package itso.rad7.bank.client;

import itso.rad7.bank.exception.ITSOBankException;
import itso.rad7.bank.ifc.Bank;
import itso.rad7.bank.impl.ITSOBank;
import itso.rad7.bank.model.Account;
import itso.rad7.bank.model.Customer;
import java.math.BigDecimal;

public class BankClient {
   public static void main(String[] args) {
      try {
         Bank bank = ITSOBank.getBank();

         System.out.println("System is going to add new customer...");
         Customer customer1 = new Customer("xxx-xx-xxxx", "Mrs", "Julia",
               "Johnson");
         bank.addCustomer(customer1);
         System.out.println(customer1 + " has been successfully added.\n");

         System.out.println("System is going to open two new accounts "
                     + "for the customer "
                     + customer1 + "...");
         Account account11 = new Account("11", new BigDecimal(10000.00D));
         bank.openAccountForCustomer(customer1, account11);
         Account account12 = new Account("12", new BigDecimal(11234.23D));
         bank.openAccountForCustomer(customer1, account12);
         System.out.println("Account " + account11.getAccountNumber()
                     + " and account " + account12.getAccountNumber()
                     + " have been successfully opened for " + customer1
                     + ".\n");

         System.out.println("System is listing all account information of "
               + customer1 + "...");
         System.out.println(bank.getAccountsForCustomer(customer1.getSsn()));

         BigDecimal amount = new BigDecimal(2500.00D);
         System.out.println("\nSystem is going to make credit of $"
               + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
               + " to account " + account11.getAccountNumber() + "...");
         bank.deposit(account11.getAccountNumber(), amount);
         System.out.println("Account " + account11.getAccountNumber()
               + " has sucessfully credited by $"
               + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN) + ".\n");

         amount = new BigDecimal(1234.23D);
         System.out.println("System is going to make debit of $"
               + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
               + " to account " + account12.getAccountNumber() + "...");
         bank.withdraw(account12.getAccountNumber(), amount);
         System.out.println("Account " + account12.getAccountNumber()
               + " has sucessfully debited by $"
               + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN) + ".\n");

         System.out.println("System is listing all account information of "
               + customer1 + "...");
         System.out.println(bank.getAccountsForCustomer(customer1.getSsn()));

         System.out.println("\nSystem is going to close account "
               + account11.getAccountNumber() + " of customer "
               + customer1 + "...");
         bank.closeAccountOfCustomer(customer1, account11);
         System.out.println("Account " + account11.getAccountNumber()
               + " has been successfully closed for " + customer1 + ".\n");

         System.out.println("System is listing all account information of "
               + customer1 + "...");
         System.out.println(bank.getAccountsForCustomer(customer1.getSsn()));

         amount = new BigDecimal(5000.00D);
         System.out.println("\nSystem is going to make credit of $"
               + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
               + " to account " + account12.getAccountNumber() + "...");
         bank.deposit(account12.getAccountNumber(), amount);
         System.out.println("Account " + account12.getAccountNumber()
               + " has sucessfully credited by $"
               + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN) + ".\n");

         System.out.println("System is listing all account information of "
               + customer1 + "...");
         System.out.println(bank.getAccountsForCustomer(customer1.getSsn()));

         System.out.println("\nSystem is listing all customers ...");
         System.out.println(bank.getCustomers() + "\n");
         java.util.Iterator iter = bank.getCustomers().values().iterator();
         while (iter.hasNext()) {
            Customer customer = (Customer) iter.next();
            System.out.println("Customer: "
                  + customer + "\n"
                  + bank.getAccountsForCustomer(customer.getSsn()) + "\n");
         }
      } catch (ITSOBankException e) {
         e.printStackTrace();
      }
   }
}
```

Running the ITSO Bank application

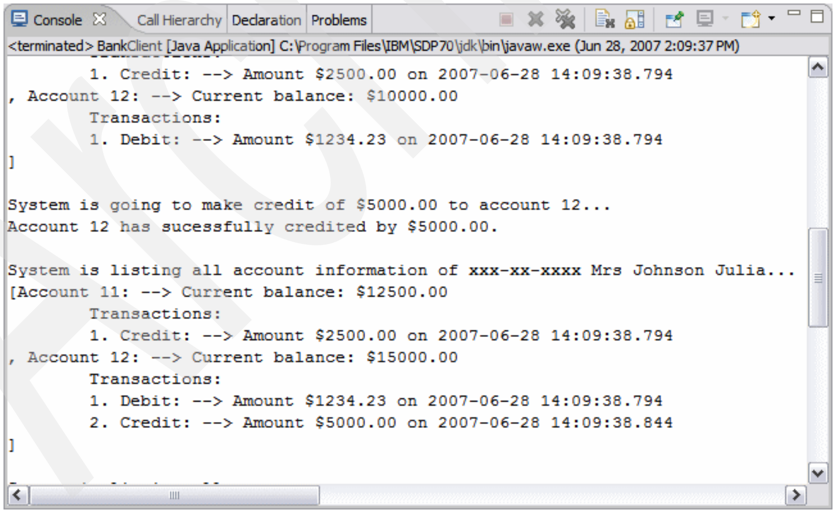
Once you have completed the ITSO Bank application and resolved any outstanding errors, you are ready to test the Bank application.

To run the ITSO Bank application, do these steps:

- Right-click the `BankClient` class in the Package Explorer and select **Run As → Java Application** or click the arrow of ▶ ▾ in the toolbar and select **Run As → Java Application**.

> **Note:** The selected class must be executable—containing a `public static void main(String[] args)` method—otherwise, the application cannot run.
>
> By executing the steps above, you have created a run configuration. As long as you have not started another application or run another configuration, this configuration is used every time, you click ▶ in the toolbar. This might be a little bit confusing, especially when you have opened or selected another executable Java class, but still the previous application is launched. To execute the new selected Java class, first you have to make a new run configuration. How to create a run configuration is explained in “Creating a run configuration” on page 285.

- You can see the output in the Console view, similar to Figure 7-28.



```
<terminated> BankClient [Java Application] C:\Program Files\IBM\SDP70\jdk\bin\javaw.exe (Jun 28, 2007 2:09:37 PM)
        1. Credit: --> Amount $2500.00 on 2007-06-28 14:09:38.794
, Account 12: --> Current balance: $10000.00
        Transactions:
        1. Debit: --> Amount $1234.23 on 2007-06-28 14:09:38.794
]

System is going to make credit of $5000.00 to account 12...
Account 12 has sucessfully credited by $5000.00.

System is listing all account information of xxx-xx-xxxx Mrs Johnson Julia...
[Account 11: --> Current balance: $12500.00
        Transactions:
        1. Credit: --> Amount $2500.00 on 2007-06-28 14:09:38.794
, Account 12: --> Current balance: $15000.00
        Transactions:
        1. Debit: --> Amount $1234.23 on 2007-06-28 14:09:38.794
        2. Credit: --> Amount $5000.00 on 2007-06-28 14:09:38.844
]
```

Figure 7-28 Console view with output of the ITSO Bank application

Additional features used for Java applications

In this section we are highlighting some key features of Application Developer when working on a Java project:

- Analyzing source code
- Creating a run configuration
- Debugging a Java application
- Using the Java scrapbook
- Plugable Java Runtime Environment (JRE)
- Exporting Java applications to a JAR file
- Running Java applications external to Application Developer
- Importing Java resources from a JAR file into a project

Analyzing source code

This section describes how to work with the Java code review provider:

- Creating and editing a static analysis configuration
- Running a static analysis

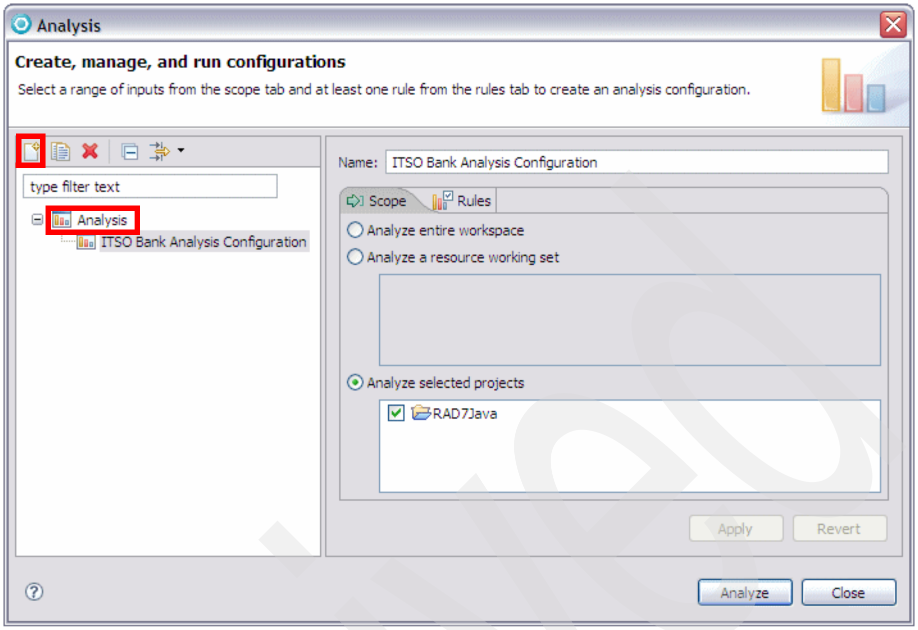
Creating and editing a static analysis configuration

For each resource, you can create an analysis configuration that specifies the rules and rule categories that are used when analyzing the resource. A static analysis code review, for example, detects violations of specific programming rules and rule categories and generates a report in the Analysis Results view, as shown in “Analysis Results view” on page 234.

To create an analysis configuration, you must be working in a perspective that supports analysis capabilities. The Java and Debug perspectives support analysis capabilities by default. In all other perspectives, you can add it. Select **Window** → **Customize Perspective**, select the **Commands** tab and **Analysis**.

To create an analysis configuration, do these steps:

- Click **Run** → **Analysis**, or right-click a project in the Package Explorer and select **Analysis** → **Analysis**, or click the arrow of in the toolbar and select **Analysis**.
- In the Analysis dialog, select Analysis and click to create a new configuration (Figure 7-29).



Analysis

Create, manage, and run configurations

Select a range of inputs from the scope tab and at least one rule from the rules tab to create an analysis configuration.

type filter text

- Analysis
  - ITSO Bank Analysis Configuration

Name: ITSO Bank Analysis Configuration

Scope | Rules

○ Analyze entire workspace

○ Analyze a resource working set

● Analyze selected projects

☑ RAD7Java

Figure 7-29   Analysis: Create, manage, and run configurations

- Type a name for the analysis configuration in the Name field and set the scope of your analysis:
  - Analyze entire workspace: The rules that you select on the Rules tab are applied to all the resources in your workspace.
  - Analyze a resource working set: The rules that you select on the Rules tab are applied to a specific set of projects, folders, or files in your workspace.
  - Analyze selected projects: The rules that you select on the Rules tab are applied to the resources in the project you select.
- Select the **Rules** tab to specify the rule categories, rules, or rule sets to apply during the analysis:
  - Analysis Domains and Rules: Expand the tree and select providers, rule categories, and rules. For example, select **Code review for Java**.
  - Rule Sets: Select a defined rule set and click **Set** to configure the providers, rule categories, and rules. For example, select **Java Quick Code Review**.

  Note that setting a rule set selects a subset of domains and rules.
- Click **Apply**.

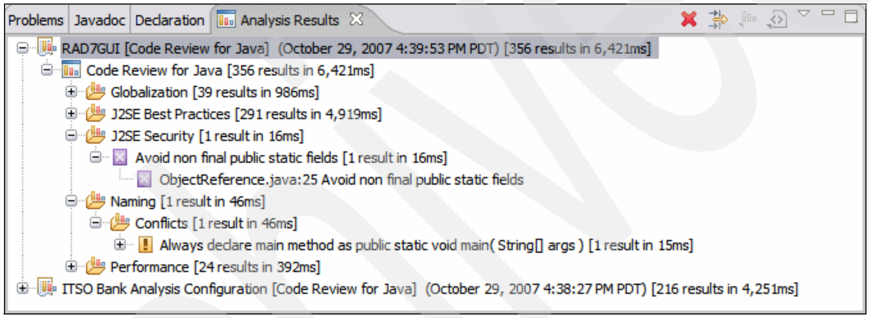
Running a static analysis

You can analyze your source code using the analysis configurations that you created.

To run a static analysis select an existing configuration or create a new configuration and click **Analyze**.

While the analysis runs, the Analysis Results view opens and, if your source code does not conform to the rules in the analysis configuration, the view populates with results. The results are listed in chronological order and are grouped into the same categories that you specified in the analysis configuration.

If you run the analysis for the RAD7Java project, no problems are reported. If running the analysis for the RAD7GUI project developed in Chapter 10, “Develop GUI applications” on page 403, then many problems are reported (Figure 7-30).



```
RAD7GUI [Code Review for Java]  (October 29, 2007 4:39:53 PM PDT) [356 results in 6,421ms]
  Code Review for Java [356 results in 6,421ms]
    Globalization [39 results in 986ms]
    J2SE Best Practices [291 results in 4,919ms]
    J2SE Security [1 result in 16ms]
      Avoid non final public static fields [1 result in 16ms]
        ObjectReference.java:25 Avoid non final public static fields
    Naming [1 result in 46ms]
      Conflicts [1 result in 46ms]
        Always declare main method as public static void main( String[] args ) [1 result in 15ms]
    Performance [24 results in 392ms]
ITSO Bank Analysis Configuration [Code Review for Java]  (October 29, 2007 4:38:27 PM PDT) [216 results in 4,251ms]
```

Figure 7-30 Analysis report for a Java project

Static analysis results

A static analysis result is a concise explanation of a rule violation. The result is a line item in the Analysis Results view that shows that the resource does not comply with the rules that you applied.

A result is not necessarily a problem, mistake, or bug, but you have to evaluate each result in the list to determine what action, if any, you have to take. If the result is a problem that has a trivial solution, the author of the rule might have provided a quick fix that automatically corrects the resource. How to process a quick fix is described in “Quick fix” on page 310.

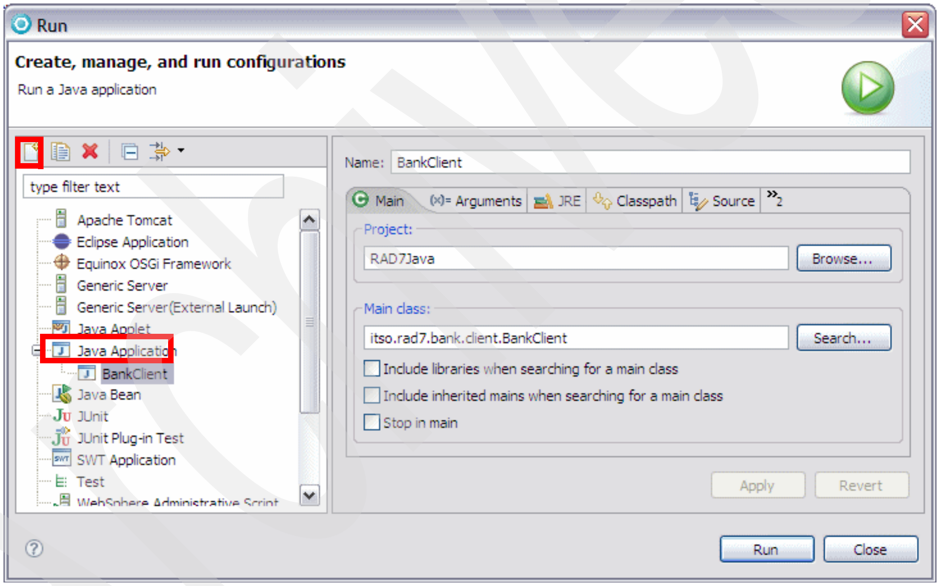
To locate a problem, right-click an entry in the Analysis Results view and select **View Result**. This action opens the Java source file with the problem code highlighted.

Creating a run configuration

When you launch the application, as shown in “Running the ITSO Bank application” on page 281, you are running the class using a generic Java Application launch configuration that derives most of the launch parameters from the Java project and the workbench preferences. In some cases, you might want to override the derived parameters or specify additional arguments.

To create a run configuration, do these steps:

- Select **Run** → **Run**, or right-click **int** the Package Explorer and select **Run As** → **Run,** or click the arrow of the **Run** icon in the toolbar and select **Run**.
- In the Run dialog, select Java Application and then click to create a new configuration (Figure 7-31). Notice that we already have a configuration from running the `BankClient` application.



Name: BankClient

Project: RAD7Java

Main class: itso.rad7.bank.client.BankClient

☐ Include libraries when searching for a main class

☐ Include inherited mains when searching for a main class

☐ Stop in main

Figure 7-31 Run configuration dialog

- The Main tab defines the class to be launched:
  - Project: Select the project containing the class to launch.
  - Main class: Click **Search** to get a list of all executable main classes in the project. Select the main class to be launched.

> **Note:** With the check boxes **Include libraries when searching for a main class** and **Include inherited mains when searching for main class**, you can expand the area where Application Developer is searching for an executable class.

  - Stop in main: The program stops in the main method whenever it is launched in debug mode.

> **Note:** You do not have to specify a project, but doing so allows a default classpath, source lookup path, and JRE to be chosen.

- The Arguments tab defines the arguments to be passed to the application and to the virtual machine. To add a program argument, do these steps:
  - Click **Variables** below the Program arguments field.
  - Select one of the predefined variables or create your own variable by clicking **Edit Variables** and then **New**.
    - Enter the name and the value for the variable and click **OK** to add it.
    - Click **OK** again to return to the Select a variable dialog. The new variable is now available in the list. Select it and click **OK** to return to the Arguments tab.
  - In the same way, you can also add VM arguments.
  - You can also specify the working directory to be used by the launched application.
- The JRE tab defines the JRE used to run or debug the application. You can select a JRE from the already defined JREs, or define a new JRE.
- The Classpath tab defines the location of class files used when running or debugging an application. By default, the user and bootstrap class locations are derived from the associated project's build path. You can override these settings here.
- The Source tab defines the location of source files used to display source when debugging a Java application. By default, these settings are derived from the associated project's build path. You can override these settings here.
- The Environment tab defines the environment variable values to use when running or debugging a Java application. By default, the environment is inherited from the Eclipse runtime. You can override or append to the inherited environment.

- The Common tab defines general information about the launch configuration. You can select to store the launch configuration in a specific file and specify which perspectives become active when the launch configuration is launched.
- Click **Run** to launch the class.

Debugging a Java application

For details debugging an application, refer to Chapter 22, “Debug local and remote applications” on page 1041.

Using the Java scrapbook

The scrapbook feature can be used to quickly run and test Java code without having to create an executable testing class. Snippets of Java code can be entered in a scrapbook page and evaluated by simply selecting the code and running it.

A scrapbook page can be added to any project and package. The extension of a scrapbook page is `.jpage`, to distinguish it from normal Java source file.

To create and run a scrapbook page, do these steps:

- Right-click a package (`itso.rad7.bank.client`) in the Package Explorer and select **New** → **Other** → **Java** → **Java Run/Debug** → **Scrapbook Page**.
- Enter a file name (`CodeSnippet`) and click **Finish**.
- The scrapbook page is opened in the Java editor and you can enter the snippet Java code.

  Example 7-10 contains two little snippets. The first one is related to the ITSO Bank application, and the second one is a simple code snippet to produce a times (multiplication) table.

Example 7-10 Java scrapbook examples

```
// ITSO Bank Snippet
Bank bank = ITSOBank.getBank();

System.out.println("System is going to add new customer...");
Customer customer = new Customer("xxx-xx-xxx", "Mrs", "Julia", "Johnson");
bank.addCustomer(customer);
System.out.println(customer + " has been successfully added.\n");

System.out.println("System is going to open an account for the customer "
                   + customer + "...");
```

```
Account account = new Account("A1", new BigDecimal(10000.00D));
bank.openAccountForCustomer(customer, account);
System.out.println("Account " + account.getAccountNumber()
                   + " has been successfully opened for " + customer
                   + ".\n");

BigDecimal amount = new BigDecimal(2500.00D);
System.out.println("System is going to make credit of $"
                   + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
                   + " to account " + account.getAccountNumber() + "...");
bank.deposit(account.getAccountNumber(), amount);
System.out.println("Account " + account.getAccountNumber()
                   + " has sucessfully credited by $"
                   + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
                   + ".\n");

bank.searchCustomerBySsn("xxx-xx-xxxx");


// Multiplication Table Snippet
String line;
int result;

for (int i = 1; i <= 10; i++) {
   line ="row " + i + ": ";

   // begin inner for-loop
   for (int j = 1; j <= 10; j++) {
      result = i*j;
      line += result + " ";
   } // end inner for-loop
   System.out.println(line);
}
```

Important: All classes that are not from the `java.lang` package must be fully qualified, or you have to set import statements:

- Right-click anywhere in the scrapbook page editor and select **Set Imports**.
- For the example, add the following types and packages:

```
itso.rad7.bank.model.*
itso.rad7.bank.ifc
itso.rad7.bank.impl.ITSOBank
java.math.BigDecimal
```

- You can execute, display, or inspect a snippet:
  - Select the code of the `Multiplication Table Snippet`, right-click, and select **Execute**, or press **Ctrl+U**, or click ▶ in the toolbar. All output is displayed in the Console view.
  - Select the `ITSO Bank Snippet`, right-click, and select **Display**, or click in the toolbar. Again, all output is displayed in the Console view. But in addition, the (uncaught) return value of the call `bank.searchCustomerBySsn("111-11-1111")` is written directly to the scrapbook page after the call.
  - Select the `ITSO Bank Snippet`, right-click, and select **Inspect**, or press **Ctrl+Shift+I**, or click 🔍 in the toolbar. Again, all output is displayed in the Console view. But in addition, an expression box opens, which allows you to inspect the current variables. Pressing **Ctrl+Shift+I** again opens the Expression view, which you can find in the Debug perspective, as described in Chapter 22, “Debug local and remote applications” on page 1041.

> **Note:** You cannot execute, display, or inspect a snippet in the scrapbook page, unless you have selected the code.

- Click ■ in the Console view to end the scrapbook evaluation.

Plugable Java Runtime Environment (JRE)

Application Developer supports Java projects to run under different versions of the Java Runtime Environment. New JREs can be added to the workspace, and projects can be configured to use any of the JREs available. By default, the Application Developer V7 uses and provides projects with support for IBM Java Runtime Environment V5.0.

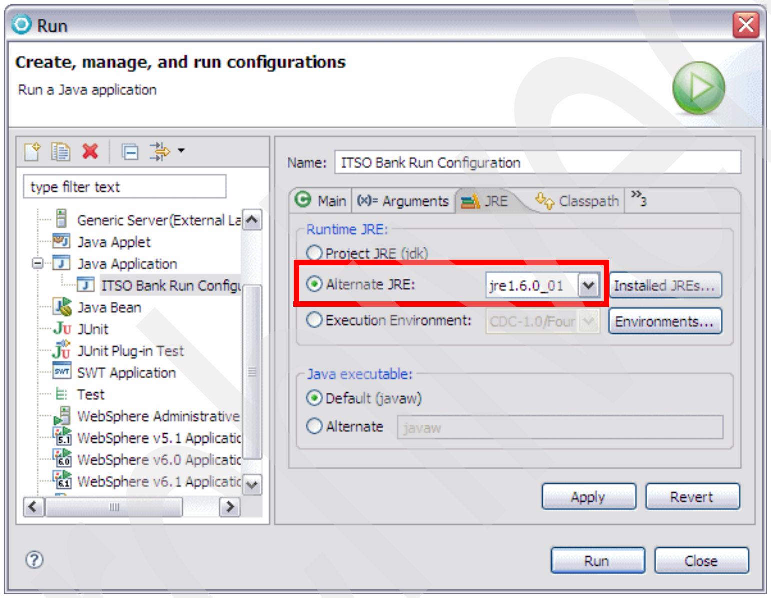
To add another JRE to the workspace, do these steps:

- Select **Window** → **Preferences** and in the Preferences dialog, select **Java** → **Installed JREs**.
- Click **Add** to add a new JRE to the workspace.
- Click **Browse** and select the home directory of the JRE you want to use.
- Click **OK**. The new added JRE is now available in the list. By default, the selected JRE is added to the build path of newly created Java projects.

The JRE that is used to run a program can also be chosen in the Run configurations dialog (Figure 7-32).

To change the JRE in a run configuration, do these steps:

- Select **Run** → **Run or** click the **Run** icon in the toolbar and then select **Run**.
- Select an existing Java application run configuration or create a new configuration as described in “Creating a run configuration” on page 285.
- Select the JRE tab, select **Alternate JRE**, and select the JRE you want to use.


Run

Create, manage, and run configurations

Run a Java application

Name: ITSO Bank Run Configuration

Runtime JRE:

○ Project JRE (jdk)

● Alternate JRE: jre1.6.0_01

○ Execution Environment: CDC-1.0/Four

Java executable:

● Default (javaw)

○ Alternate javaw

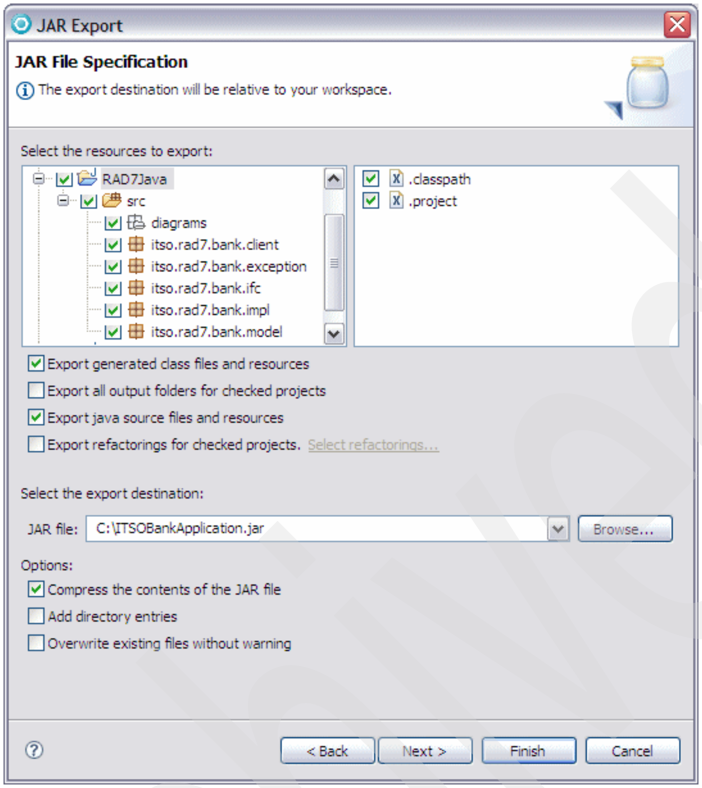
Figure 7-32 Run JRE configuration tab

Exporting Java applications to a JAR file

This section describes how to export a Java application to a JAR file that can be run outside Application Developer using a JRE in a Windows Command Prompt. We demonstrate how to export and run the ITSO Bank application.

To export the ITSO Bank application code to a JAR file, do these steps:

- Right-click the `RAD7Java` project and select **Export**.
- In the Export dialog, select **Java** → **JAR file** and click **Next**.
- In the JAR Export dialog, enter the following data (Figure 7-33):



JAR Export

JAR File Specification

ⓘ The export destination will be relative to your workspace.

Select the resources to export:

- ☑ RAD7Java
  - ☑ src
    - ☑ diagrams
    - ☑ itso.rad7.bank.client
    - ☑ itso.rad7.bank.exception
    - ☑ itso.rad7.bank.ifc
    - ☑ itso.rad7.bank.impl
    - ☑ itso.rad7.bank.model

☑ .classpath
☑ .project

☑ Export generated class files and resources
☐ Export all output folders for checked projects
☑ Export java source files and resources
☐ Export refactorings for checked projects. Select refactorings...

Select the export destination:

JAR file: C:\ITSOBankApplication.jar

Options:

☑ Compress the contents of the JAR file
☐ Add directory entries
☐ Overwrite existing files without warning

Figure 7-33 JAR Export

- – Select the `RAD7Java` project.
- – Select **Export generated class files and resources**.
- – Select **Export java source files and resources**.

> **Note:** We select to include the source to demonstrate later how to import a JAR file into a project. It is not necessary or desirable to include Java sources in a JAR file for execution.

- – JAR file: `C:\ITSOBankApplication.jar`
- – Select **Compress the contents of the JAR file**.
- – Clear other check boxes.

- ► In the JAR Packaging Options dialog accept the defaults and click **Next**.
- ► In the JAR Manifest Specification dialog click **Browse** for the Main class field and select the `BankClient` class.

- Click **Finish** to export the entire Java project as a JAR file.

Running Java applications external to Application Developer

Once you have exported the Java application as a JAR file you can run the Java application on any installed JRE on your system (at least as long as there are no version conflicts).

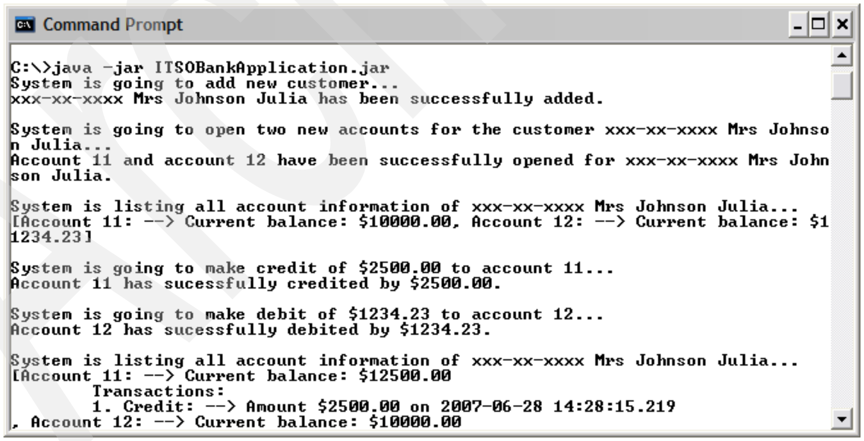
> **Note:** Ensure that the JRE is set in the Windows environment variable called `PATH`. You can add the JRE to the path with the following command in the Windows Command Prompt:
>
> ```
> set path=%path%;{JREInstallDirectory}\bin
> set path=%path%;C:\Program Files\IBM\SDP70\jdk\jre\bin
> ```

To run a Java application external to Application Developer on a Windows system, do these steps:

- Open a Command Prompt and navigate to the directory to which you have exported the JAR file, for example `C:\`.
- Enter the following command to run the ITSO Bank application:

  ```
  java -jar ITSOBankApplication.jar
  ```

  This results in the `main` method of `BankClient` being executed, and the results are shown in Figure 7-34.



```
C:\>java -jar ITSOBankApplication.jar
System is going to add new customer...
xxx-xx-xxxx Mrs Johnson Julia has been successfully added.

System is going to open two new accounts for the customer xxx-xx-xxxx Mrs Johnso
n Julia...
Account 11 and account 12 have been successfully opened for xxx-xx-xxxx Mrs John
son Julia.

System is listing all account information of xxx-xx-xxxx Mrs Johnson Julia...
[Account 11: --> Current balance: $10000.00, Account 12: --> Current balance: $1
1234.23]

System is going to make credit of $2500.00 to account 11...
Account 11 has sucessfully credited by $2500.00.

System is going to make debit of $1234.23 to account 12...
Account 12 has sucessfully debited by $1234.23.

System is listing all account information of xxx-xx-xxxx Mrs Johnson Julia...
[Account 11: --> Current balance: $12500.00
        Transactions:
        1. Credit: --> Amount $2500.00 on 2007-06-28 14:28:15.219
, Account 12: --> Current balance: $10000.00
```

Figure 7-34 Output from running ITSOBankApplication.jar in Command Prompt

Importing Java resources from a JAR file into a project

This section describes how to import Java resources from a JAR file into an existing Java project in the workspace.

We use the `ITSOBankApplication.jar` file which we have created in “Exporting Java applications to a JAR file” on page 290. Alternatively, you can use the `ITSOBankApplication.jar` provided in the `c:\7501code\java\jar` directory included with the Redbooks publication sample code.

- Create a Java project called **`RAD7JavaImport`** with the default options.
- Right-click the `RAD7JavaImport` project in the Package Explorer and select **Import**.
- In the Import dialog, select **General** → **Archive File** and click **Next**.
- In the Import - Archive File dialog click **Browse** and locate the JAR file (for example, `c:\ITSOBankApplication.jar`).
- Clear the files `.classpath` and `.project` and the folder `META-INF`. These file are created when required.
- Click **Finish**. Notice that by default there is no `src` folder and the packages are directly under the project.

To test the imported Java project, select and run the `BankClient` class from the Package Explorer.

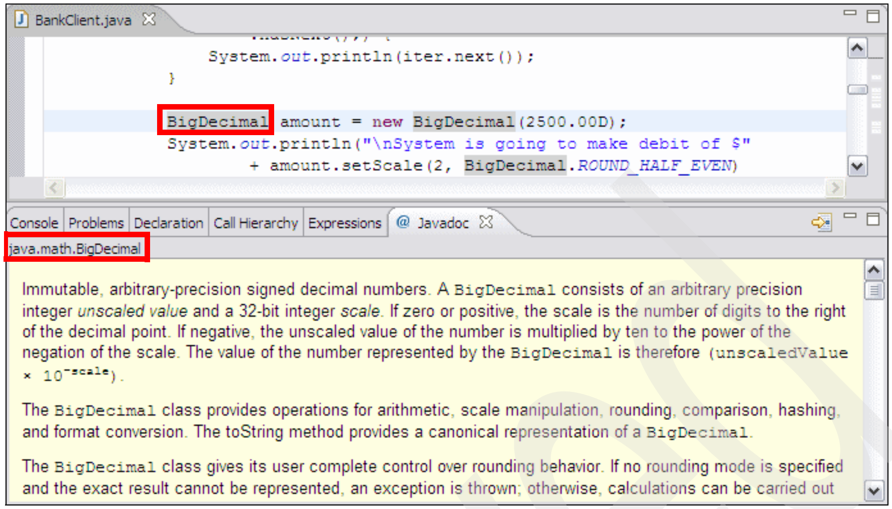
Javadoc tooling

Javadoc is a very useful tool in the Java Development Kit used to document Java code. It generates a Web-based (html files) documentation of the packages, interfaces, classes, methods and fields.

Application Developer has a Javadoc view, which is implemented using a SWT browser widget to display HTML. In the Java perspective, the Javadoc view is context sensitive. It only displays the Javadoc associated with the Java element where the cursor is currently located within the Java editor.

To demonstrate the use of Javadoc, we use the `RAD7JavaImport` project that we imported.

- Open the Javadoc view in the Java perspective if it is not already open.
- Open the `BankClient` class in the Java editor.
- You will notice that when the cursor selects a type, its Javadoc is shown in the Javadoc view. Select the `BigDecimal` type, and the Javadoc view changes to the documentation associated with `BigDecimal`, as shown in Figure 7-35.



```
                System.out.println(iter.next());
            }

        BigDecimal amount = new BigDecimal(2500.00D);
        System.out.println("\nSystem is going to make debit of $"
                + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
```

Immutable, arbitrary-precision signed decimal numbers. A `BigDecimal` consists of an arbitrary precision integer *unscaled value* and a 32-bit integer *scale*. If zero or positive, the scale is the number of digits to the right of the decimal point. If negative, the unscaled value of the number is multiplied by ten to the power of the negation of the scale. The value of the number represented by the `BigDecimal` is therefore (`unscaledValue × 10`$^{-scale}$).

The `BigDecimal` class provides operations for arithmetic, scale manipulation, rounding, comparison, hashing, and format conversion. The toString method provides a canonical representation of a `BigDecimal`.

The `BigDecimal` class gives its user complete control over rounding behavior. If no rounding mode is specified and the exact result cannot be represented, an exception is thrown; otherwise, calculations can be carried out

Figure 7-35 Javadoc view: Context sensitive (BigDecimal)

Generating Javadoc

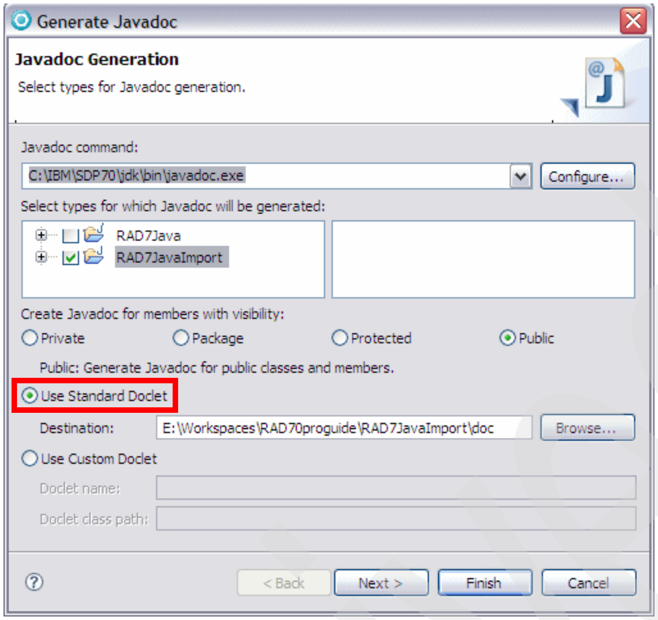
This section explains how to generate Javadoc from an existing Java project. Application Developer supports the following types of Javadoc generation:

- Generate Javadoc
- Generate Javadoc with diagrams from existing tags
- Generate Javadoc with diagrams automatically
- Generate Javadoc from an Ant script

Generate Javadoc

To generate Javadoc from an existing Java project, do these steps:

- Right-click the `RAD7JavaImport` project in the Package Explorer and select **Export** → **Java** → **Javadoc**, or select **Project** → **Generate Javadoc.**
- In the Javadoc Generation dialog, enter the following data (Figure 7-36):
  - The Javadoc command is predefined.
  - Select **Public** for Create Javadoc for members with visibility (default).
  - Select **Use Standard Doclet**. Alternatively, you can specify a custom doclet with the name of the doclet and the classpath to the doclet implementation.
  - Destination: `{workspaceDirectory}\RAD7Java\doc` (default)
    Generates Javadoc in the `doc` directory of current project



Generate Javadoc

Javadoc Generation

Select types for Javadoc generation.

Javadoc command: C:\IBM\SDP70\jdk\bin\javadoc.exe

Select types for which Javadoc will be generated:

☐ RAD7Java

☑ RAD7JavaImport

Create Javadoc for members with visibility:

○ Private ○ Package ○ Protected ● Public

Public: Generate Javadoc for public classes and members.

● Use Standard Doclet

Destination: E:\Workspaces\RAD70proguide\RAD7JavaImport\doc

○ Use Custom Doclet

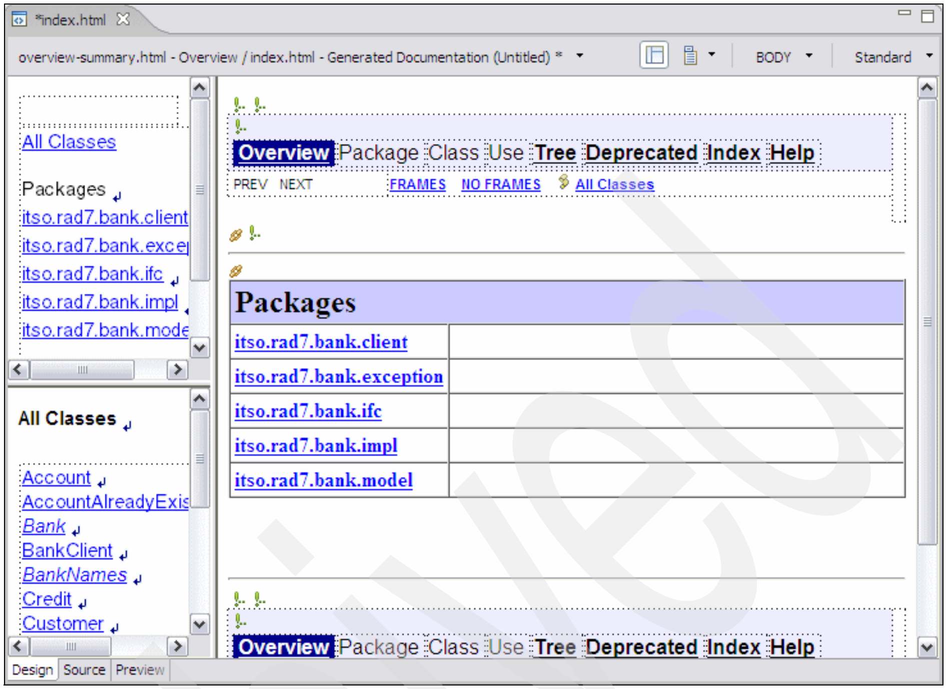
Doclet name:

Doclet class path:

Figure 7-36 Javadoc Generation dialog

- In the Configure Javadoc arguments for standard doclets dialog accept the default settings and click **Next**.
- In the Configure Javadoc arguments dialog, enter the following data:
  - Select **1.5** for JRE source compatibility because we use generic types in the project, which are only supported from JDK version 1.5 (5.0) or higher.
  - Select **Save the settings for this Javadoc export as an Ant script** and accept the destination:

    `{workspaceDirectory}\RAD7JavaImport\javadoc.xml`
- Click **Finish** to generate the Javadoc.
- When prompted to update the Javadoc location, click **Yes to all**.
- When prompted that the Ant file will be created, click **OK**.
- Open the Javadoc in a browser and explore the generated Javadoc for the `RAD7JavaImport` project (Figure 7-37).
  - Expand **RAD7JavaImport** → **doc** in the Package Explorer.
  - Double-click **index.html** to display the Javadoc contents for the project.



Packages	
itso.rad7.bank.client	
itso.rad7.bank.exception	
itso.rad7.bank.ifc	
itso.rad7.bank.impl	
itso.rad7.bank.model	

Figure 7-37 Javadoc output generated from the Javadoc wizard

Generate Javadoc from an Ant script

In “Generate Javadoc” on page 294, we selected **Save Settings for this Javadoc export as an Ant script**. This generated the `javadoc.xml` ant script, which can be used to invoke the Javadoc command.

To generate Javadoc from an Ant script, do these steps:

- Right-click `javadoc.xml` in the Package Explorer and select **Run As** → **Ant Build**.
- The Javadoc generation process starts. If you cannot see the new generated `doc` folder in the project, select the project and press **F5** to refresh the view.

Generate Javadoc with diagrams from existing tags

Application Developer enables you to embed a *@viz.diagram* tag into Javadoc on the class, interface, or package level. This tag must refer to an existing diagram which is located in the same package as the interface or class itself, and the wizard will then export that diagram into a GIF, JPG, or BMP, and embed it to the generated Javadoc for the class, interface, or package.

Example 7-11 shows the use of the `@viz.diagram` tag in the `BankClient` class.

Example 7-11 BankClient class with a @viz.diagram tag

```
package itso.rad7.bank.client;
...
/**
 * @viz.diagram ITSOBank-ClassDiagram.dnx
 */
public class BankClient {
   public static void main(String[] args) {
      try {
...
```

Note that the `ITSOBank-Diagram.dnx` file must be in the same package as the class. Copy the diagram from the `diagram` folder to the `itso.rad7.bank.client` package.

To generate Javadoc with diagrams from existing tags, do these steps:

- Add the `@viz.diagram` tag to the source code, as shown in Example 7-11, and copy the diagram to the package.

> **Restriction:** The `@viz.diagram` tag assumed that the diagram being referenced is placed in the same folder as the Java file containing the `@viz.diagram` tag.
>
> You have to place your diagrams along with the Java source code. For Web applications, this has the side effect of the class diagrams being packaged into the WAR file with the compiled Java code.
>
> We found two possible work-arounds:
>
> - You can manually remove these diagrams from the WAR file after exporting.
> - Filter the contents by configuring an exclusion filter for the EAR export feature.
>
> Refer to “Filtering the content of an EAR” on page 1128 for information on techniques for filtering files (include and exclude) when exporting the EAR.

- Right-click the project in the Package Explorer and select **Project** → **Generate Javadoc with Diagrams** → **From Existing tags**.
- In the Javadoc Generation dialog, use the same options as in Figure 7-36 on page 295.
- Click **Next** in the next dialog panel.

- In the Configure Javadoc arguments dialog, select **1.5** for JRE source compatibility.

Important: There is an identified product defect in this feature (APAR PK41008). While running this wizard, you get errors such as the following examples, and the Javadoc will be generated without diagrams:

- warning - `@viz.diagram` is an unknown tag
- javadoc: Error - Exception `java.lang.ClassNotFoundException` thrown while trying to register Taglet `com.ibm.xtools.viz.j2se.ui.javadoc.internal.taglet.DiagramTaglet`

The `ClassNotFoundException` can be eliminated by upgrading Application Developer to version 7.0.0.2, which upgrades the version of the IBM JVM from 5 SR3 to 5 SR4a.

After upgrading to 7.0.0.2, there will still be an error:

- warning - `@viz.diagram` is an unknown tag

This remaining error can be resolved with an additional input in the Extra Javadoc options field of the Javadoc Generation - Configure Javadoc arguments dialog:

```
-tagletpath "C:\IBM\SDP70Shared\plugins\
   com.ibm.xtools.viz.j2se.ui.javadoc_7.0.100.v20061213_1732.jar"
```

The path above is the default installation path. You might have to verify what the actual location is as well as the version number of the above plug-in on your installation.

- In the Choose diagram image generation options dialog, accept the default settings and click **Finish**.
- When prompted to update the Javadoc location, click **Yes to all**.
- Open the Javadoc in a browser and verify that a diagram has been added to the generated Javadoc for the `BankClient` class.
  - Expand **RAD7JavaImport** → **doc** in the Package Explorer.
  - Double-click **index.html** to display the Javadoc contents.
  - Select `BankClient` class in the All Classes pane and verify that the diagram has been imported.

Generate Javadoc with diagrams automatically

If you do not have diagrams that you want to embed to the generated Javadoc, you can let Application Developer to generate diagrams for you and embed them to the Javadoc.

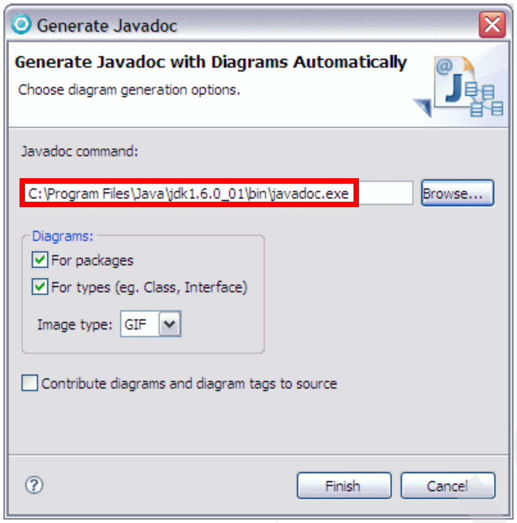
To generate Javadoc with diagrams automatically, do these steps:

- Select the project in the Package Explorer, and select **Project → Generate Javadoc with Diagrams → Automatically**.
- In the Generate Javadoc with diagrams automatically dialog, enter the following data (Figure 7-38).
  - Javadoc command: path to javadoc.exe

    ```
    {JDKInstallDirectory}\bin\javadoc.exe
    ```

> **Important:** In the ITSO Bank application we are using generic types that are supported by JDK version 1.5 (5.0) or higher. In the previous Javadoc wizards, we could select JRE source compatibility. But this wizard does not let us choose it.
>
> We found that when we used the default JDK (`C:\IBM\SDP70\jdk\bin\javadoc.exe`), it automatically added the argument `-source 1.3` to the javadoc command. Therefore, running this wizard with the default JDK on the RAD7Java project ends in an error, because the generic types are not supported.
>
> We found two work-arounds to solve the problem:
>
> - We downloaded and installed the newest Sun™ JDK version (in our case version 1.6.0_01) and used the javadoc command from this JDK.
> - We first generate Javadoc, as it is described in “Generate Javadoc” on page 294 and then run the **Generate Javadoc with diagrams automatically** again. There are still errors shown in the Console view—in fact, the Javadoc has not generated—but the diagrams were added to the existing Javadoc.

  - Keep the defaults for Diagrams.
  - Optionally select **Contribute diagrams and diagrams tags to source** if you want the `@viz.diagram` tags to be stored in the Java sources and the generated diagrams to be stored in the packages of the Java sources.



Generate Javadoc with Diagrams Automatically

Choose diagram generation options.

Javadoc command:

C:\Program Files\Java\jdk1.6.0_01\bin\javadoc.exe

Diagrams:

☑ For packages

☑ For types (eg. Class, Interface)

Image type: GIF

□ Contribute diagrams and diagram tags to source

Figure 7-38 Generate Javadoc with Diagrams Automatically dialog

- Click **Finish** to generate the Javadoc, then open the Javadoc and browse the classes with the generated diagrams.

Java editor and rapid application development

Application Developer contains a number of features that ease and expedite the code development process. These features are designed to make life easier for both experienced and novice Java programmers by simplifying or automating many common tasks.

This section is organized into the following topics:

- Navigating through the code
- Source folding
- Type hierarchy
- Smart insert
- Marking occurrences
- Smart compilation
- Java and file search
- Working sets
- Quick fix
- Quick assist
- Content assist
- Import generation

- Adding constructors
- Delegate method generator
- Refactoring

Navigating through the code

This section highlights the use of the Outline view, Package Explorer, and bookmarks to navigate through the code.

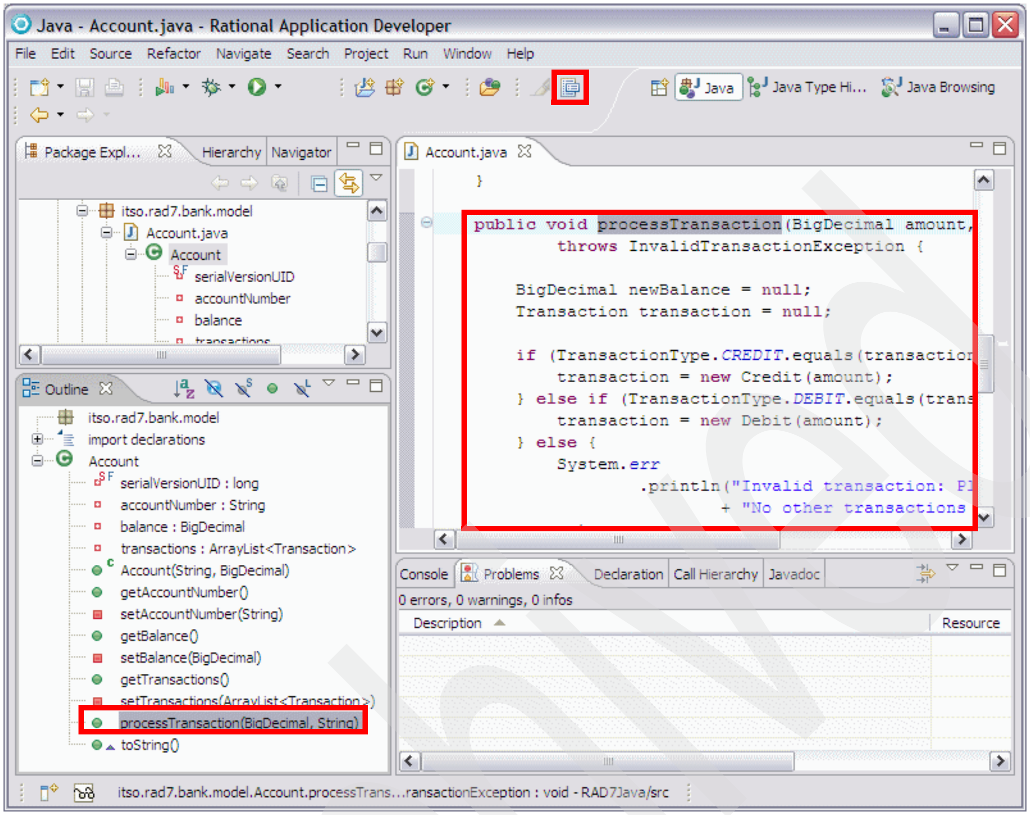
Using the Outline view to navigate the code

The Outline view displays an outline of a structured file that is currently open in the editor area, and lists structural elements. The contents of the Outline view are editor-specific.

For example, in a Java source file, the structural elements are package name, import declarations, class, fields, and methods. We use the `RAD7Java` project to demonstrate the use of the Outline view to navigate through the code:

- Select and expand the **RAD7Java** → **src** → **itso.rad7.bank.model** from the Package Explorer.
- Double-click **Account.java** to open the file in the Java editor.
- By selecting elements in the Outline view, you can navigate to the corresponding point in your code. This allows you to easily find method and field definitions without scrolling through the Java editor, as shown in Figure 7-39.

> **Note:** If you have a source file with many fields and methods, you can use the **Show Source of Selected Element Only** feature. Select the open source file in the Java editor, and click ▤ in the toolbar to limit the edit view to the element that is currently selected in the Outline view.



```
    }

    public void processTransaction(BigDecimal amount,
            throws InvalidTransactionException {

        BigDecimal newBalance = null;
        Transaction transaction = null;

        if (TransactionType.CREDIT.equals(transactio
            transaction = new Credit(amount);
        } else if (TransactionType.DEBIT.equals(trans
            transaction = new Debit(amount);
        } else {
            System.err
                    .println("Invalid transaction: Pl
                            + "No other transactions
```

0 errors, 0 warnings, 0 infos

itso.rad7.bank.model.Account.processTrans...ransactionException : void - RAD7Java/src

Figure 7-39 Java editor - Outline view for navigation

Using the Package Explorer to navigate the code

The Package Explorer, which is available by default in the Java perspective, can also be used for navigation. The Package Explorer provides you with a Java-specific view of the resources shown in the Project Explorer view. The element hierarchy is derived from the project's build paths.

Using bookmarks to navigate the code

Bookmarks are another simple way to navigate to resources that you frequently use. The Bookmarks view displays all bookmarks in the workspace.

Show bookmarks

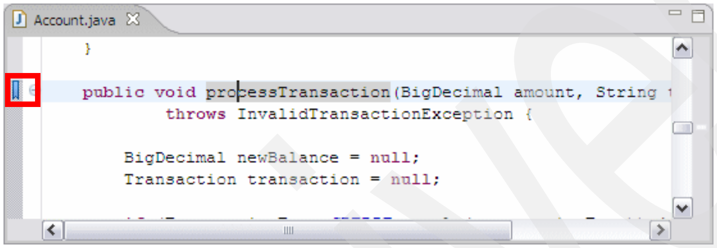
To show the Bookmarks view select **Window** → **Show View** → **General** → **Bookmarks**.

Set bookmark

To set a bookmark in the code, right-click in the gray sidebar to the left of the code in the Java editor and select **Add Bookmark** or select **Edit** → **Add Bookmark**. In the Add Bookmark dialog, enter the name of the bookmark and click **OK**.

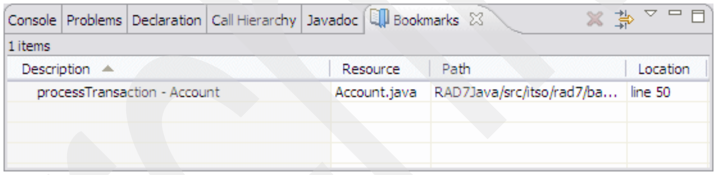
View bookmark

Bookmarks are indicated by the symbol 🔖 in the gray sidebar (Figure 7-40), and are listed in the Bookmarks view (Figure 7-41). Double-clicking the bookmark entry in the Bookmarks view opens the file and navigates to the line where the bookmark has been set.



```
    }

    public void processTransaction(BigDecimal amount, String t
            throws InvalidTransactionException {

        BigDecimal newBalance = null;
        Transaction transaction = null;
```

Figure 7-40 Java Editor with a bookmark



1 items

Description	Resource	Path	Location
processTransaction - Account	Account.java	RAD7Java/src/itso/rad7/ba...	line 50

Figure 7-41 Bookmarks view

Delete bookmarks

A bookmark can be removed by right-clicking on the bookmark symbol 🔖 in the gray sidebar and selecting **Remove Bookmark** or right-click the bookmark in the Bookmarks view and select **Delete**, or click ✖ in the Bookmark view toolbar.

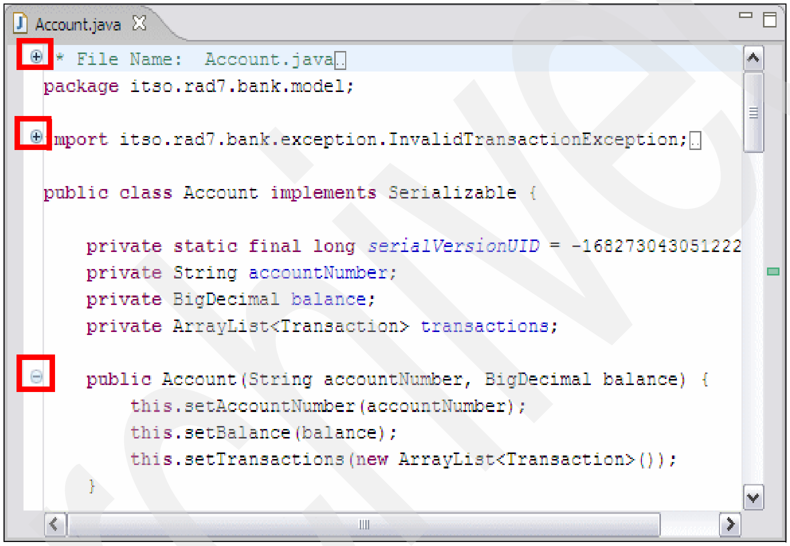
> **Note:** Bookmarks can also be given for a file. It allows you to open it quickly from the Bookmarks view later. Select the file in the Project Explorer and select **Edit** → **Add Bookmark**.
>
> Bookmarks are not specific to Java code. They can be used in any file to provide a quick way of navigating to a specific location.

Source folding

Application Developer folds the source of import statements, comments, types, and methods. Source folding can be configured through the Java editor preferences.

To configure the folding feature, select **Window** → **Preferences**. In the Preferences dialog, select **Java** → **Editor** → **Folding**.

Folded source is marked by a ⊕ symbol, expanded source by a ⊖ symbol on the left side of the source code, as shown in Figure 7-42. Click the symbol to fold or expand the source code.

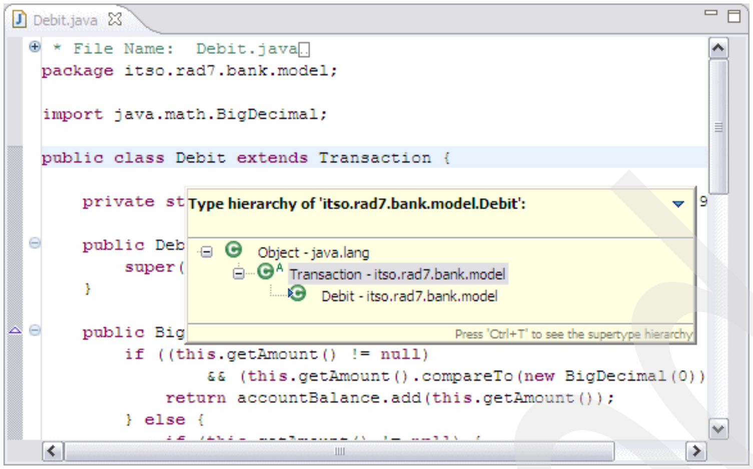


```
* File Name:  Account.java
package itso.rad7.bank.model;

import itso.rad7.bank.exception.InvalidTransactionException;

public class Account implements Serializable {

    private static final long serialVersionUID = -168273043051222
    private String accountNumber;
    private BigDecimal balance;
    private ArrayList<Transaction> transactions;

    public Account(String accountNumber, BigDecimal balance) {
        this.setAccountNumber(accountNumber);
        this.setBalance(balance);
        this.setTransactions(new ArrayList<Transaction>());
    }
```

Figure 7-42   Java Editor with source folding

Type hierarchy

The Java editor allows the quick viewing of type hierarchy of a selected type. Select a type with the cursor and press **Ctrl+T** to displays the hierarchy (Figure 7-43).



```
 * File Name:  Debit.java
package itso.rad7.bank.model;

import java.math.BigDecimal;

public class Debit extends Transaction {

    private st
    public Deb
        super(
    }

    public Big
        if ((this.getAmount() != null)
                && (this.getAmount().compareTo(new BigDecimal(0))
            return accountBalance.add(this.getAmount());
        } else {
```

Type hierarchy of 'itso.rad7.bank.model.Debit':

```
Object - java.lang
    Transaction - itso.rad7.bank.model
        Debit - itso.rad7.bank.model
```

Press 'Ctrl+T' to see the supertype hierarchy

Figure 7-43   Java Editor with quick type hierarchy view

Smart insert

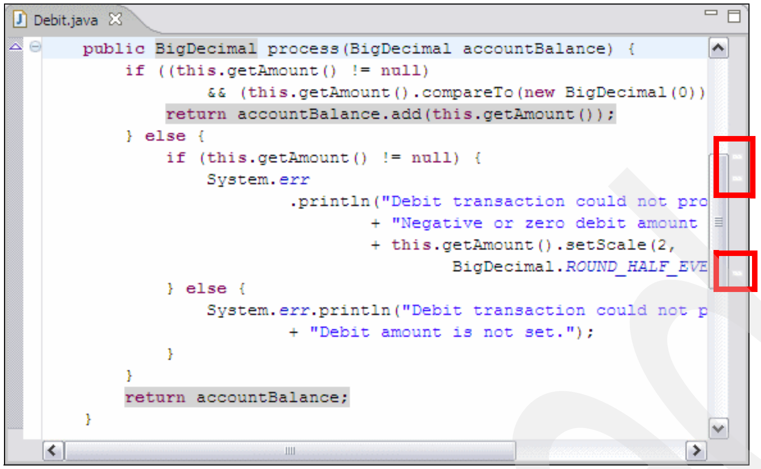
To toggle the editor between smart insert and insert modes, press **Ctrl+Shift+Insert**. When the editor is in smart insert mode, the editor provides extra features specific to Java. For example, in smart insert mode when you cut and paste code from a Java source to another Java source, all the needed imports are automatically added to the target Java file.

To configure the smart insert mode, select **Window** → **Preferences**. In the Preferences dialog, select **Java** → **Editor** → **Typing**.

Marking occurrences

When enabled, the editor highlights all occurrences of types, methods, constants, non-constant fields, local variables, expressions throwing a declared exception, method exits, methods implementing an interface, and targets of break and continue statements, depending on the current cursor position in the source code (Figure 7-44). For better orientation in large files, all occurrences are marked with a white line on the right side of the code.

The feature can be enabled and disabled by pressing **Alt+Shift+O**, or by clicking in the toolbar. To configure mark occurrences, select **Window** → **Preferences**. In the Preferences dialog, select **Java** → **Editor** → **Mark Occurrences**.



```
    public BigDecimal process(BigDecimal accountBalance) {
        if ((this.getAmount() != null)
                && (this.getAmount().compareTo(new BigDecimal(0))
            return accountBalance.add(this.getAmount());
        } else {
            if (this.getAmount() != null) {
                System.err
                        .println("Debit transaction could not pro
                                + "Negative or zero debit amount
                                + this.getAmount().setScale(2,
                                        BigDecimal.ROUND_HALF_EVE
            } else {
                System.err.println("Debit transaction could not p
                        + "Debit amount is not set.");
            }
        }
        return accountBalance;
    }
```

Figure 7-44   Java Editor with mark occurrences (methods exits)

Smart compilation

The Java builder in the Application Developer workbench incrementally compiles the Java code in the background as it is changed and displays any compilation errors automatically, unless you disable the automatic build feature. Refer to Chapter 4, “Perspectives, views, and editors” on page 117 for information on enabling and disabling automatic builds and running workbench tasks in the background.

Java and file search

Application Developer provides support for various searches. To display the Search dialog click 🔦 in the toolbar, or press **Ctrl+H**. The Search dialog can be configured to display different searches by clicking **Customize**. In Figure 7-45 the search dialog has been customized to display only the Java and File Search tabs.

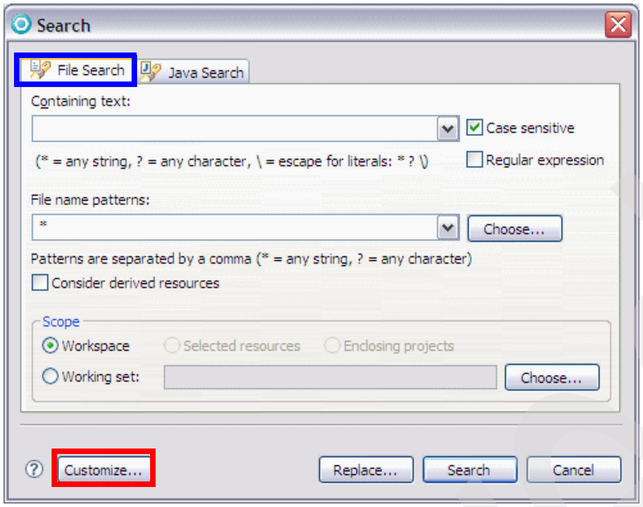


Figure 7-45 Search dialog [customized]

In the Search dialog, you can perform file, text, or Java searches:

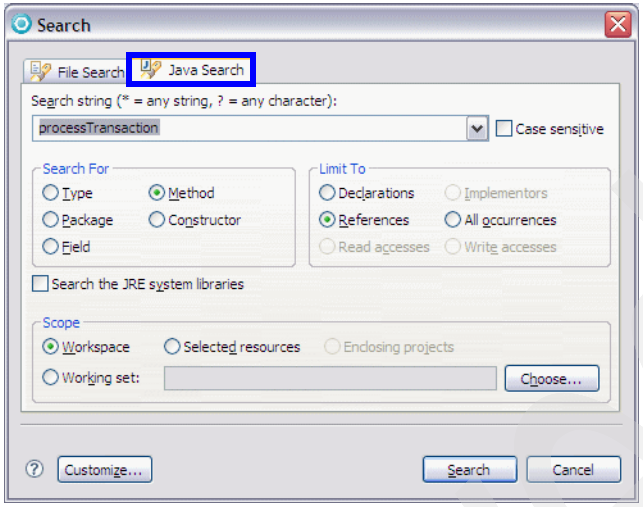
- Java searches operate on the structure of the code.
- File searches operate on the files by name and/or text content.
- Text searches allow you to find matches inside comments and strings.

Java searches are faster, because there is an underlying indexing structure for the code.

Performing a Java search from the workbench (example)

To perform a Java search from the workbench, do these steps:

- In the Java perspective click in the toolbar, or select **Search** → **Java**, or press **Ctrl+H**.
  - Select the Java Search tab and enter the following data (Figure 7-46).
  - Search string: `processTransaction`
  - Select **Method**.
  - Select **References**.
  - Select **Workspace**.



Search string (* = any string, ? = any character): processTransaction

□ Case sensitive

Search For: ○ Type ● Method ○ Package ○ Constructor ○ Field

Limit To: ○ Declarations ○ Implementors ● References ○ All occurrences ○ Read accesses ○ Write accesses

□ Search the JRE system libraries

Scope: ● Workspace ○ Selected resources ○ Enclosing projects ○ Working set:

Figure 7-46 Java Search dialog

- Click **Search**. While searching, you can click **Cancel** at any time to stop the search. Partial results will be shown. The Search view shows the search results.
- Click ⇩ or ⇧ in the toolbar of the Search view to navigate to the next or previous match. If the file in which the match was found is not currently open, it is opened in the Java editor at the position of the match. Search matches are tagged with a ⇨ symbol on the left side of the source code line.

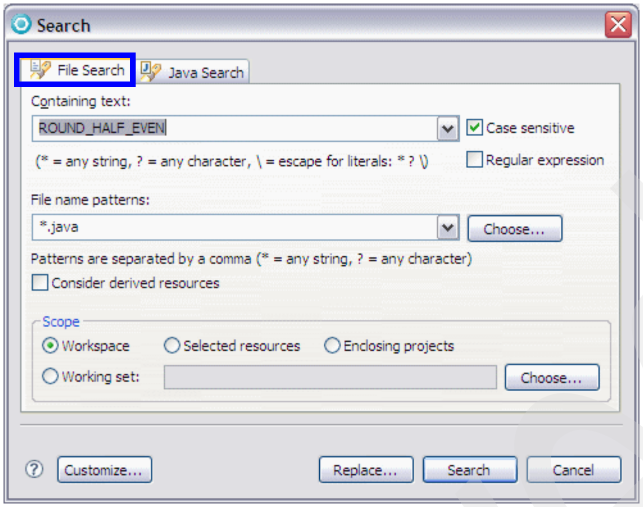
Searching from a Java view or editor

Java searches can also be performed from specific views, including the Outline view, Hierarchy view, Package Explorer, or even the Search view, or from the Java editor.

Right-click the resource you are looking for in the view or editor and select **References** → **Workspace**, or press **Ctrl+Shift+G**.

Performing a file search (example)

- In the Java perspective click in the toolbar, or select **Search** → **File**, or press **Ctrl+H**.
  - Select the File Search tab and enter the following data (Figure 7-47).
  - Containing text: `ROUND_HALF_EVEN`
  - File name patterns: `*.java` (default)
  - Select **Workspace**.



Search

Containing text: ROUND_HALF_EVEN

☑ Case sensitive

(* = any string, ? = any character, \ = escape for literals: * ? \)

□ Regular expression

File name patterns: *.java

Patterns are separated by a comma (* = any string, ? = any character)

□ Consider derived resources

Scope

● Workspace ○ Selected resources ○ Enclosing projects

○ Working set:

Figure 7-47 File Search dialog

- Click **Search**. While searching, you can click **Cancel** at any time to stop the search. Partial results will be shown. The Search view shows the search results.

> **Note:** To find all files of a given file name pattern, leave the Containing text field empty.

Viewing previous search results

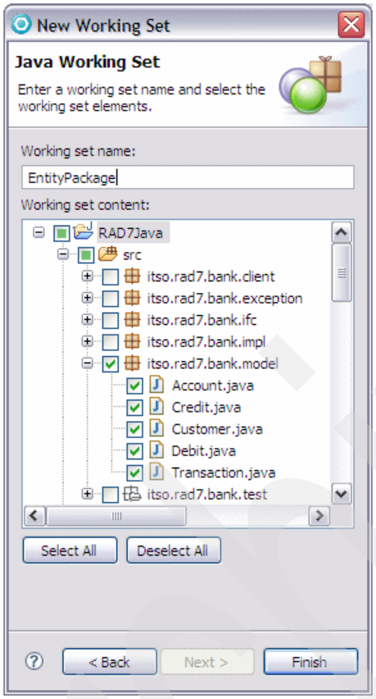
Click first ▾ on the right side of in the Search view toolbar, and select one of the previous searches. The list can be cleared by selecting **Clear History**.

Working sets

Working sets are used to filter resources by only including the specified resources. They are selected and defined using the view's filter selection dialog. We use an example to demonstrate the creation and use of a working set as follows:

- Click in the toolbar, or select **Search** → **Java**, or press **Ctrl+H** to open the Java Search Dialog.
- Enter `itso.rad7.bank.model.Credit` in the Search string field, select **Working set** under Scope, and then click **Choose**.
- In the Select Working Set dialog click **New** to create a new working set.

- In the New Working Set dialog, select **Java** to indicate that the working set includes only Java resources and then click **Next**.
- In the Java Working Set dialog, select only **RAD7Java** → **src** → **itso.rad7.bank.model**, type `EntityPackage` in the Working set name field, and click **Finish** (Figure 7-48).



Enter a working set name and select the working set elements.

Working set name: EntityPackage

Working set content:

- ■ RAD7Java
  - ■ src
    - □ itso.rad7.bank.client
    - □ itso.rad7.bank.exception
    - □ itso.rad7.bank.ifc
    - □ itso.rad7.bank.impl
    - ☑ itso.rad7.bank.model
      - ☑ Account.java
      - ☑ Credit.java
      - ☑ Customer.java
      - ☑ Debit.java
      - ☑ Transaction.java
    - □ itso.rad7.bank.test

Figure 7-48 New Java Working Set dialog

- Select the new `EntityPackage` working set in the Select Working Sets dialog, and click **OK**.
- We have now created a working set named `EntityPackage` containing Java resources comprised of all the Java files in the `itso.rad7.bank.model` package.
- Click **Search** to start the search process.

Quick fix

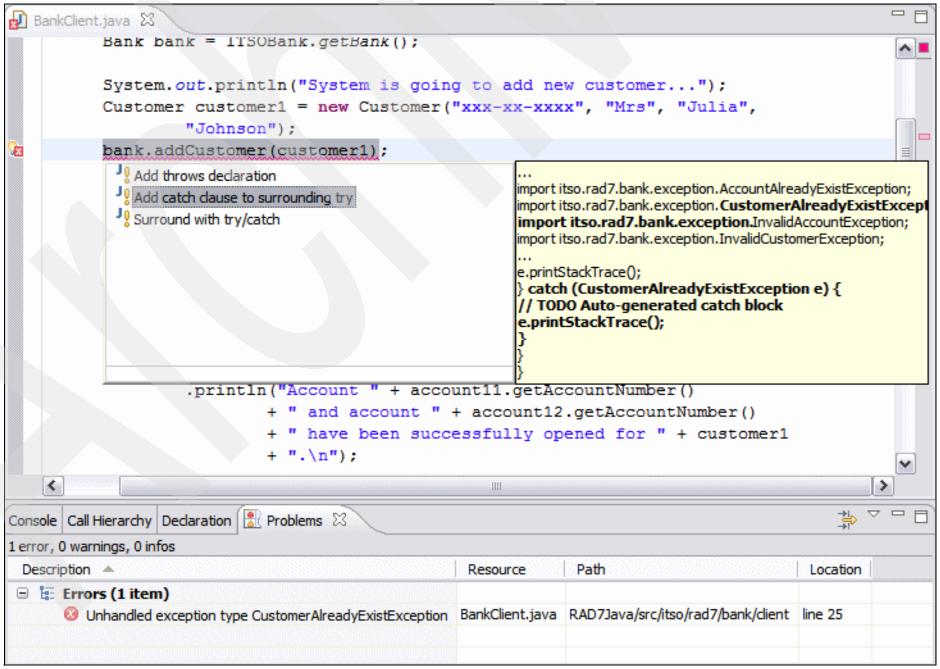
Application Developer offers a quick fix for some kind of problems, which were detected during the code compilation or static code analysis. The developer can process the quick fix to correct his code.

The following symbols show the developer that a quick fix is available:

- Static code analysis:
  - Quick fix for a result with a severity level of recommendation
  - Quick fix for a result with a severity level of warning
  - Quick fix for a result with a severity level of severe
- Code compilation:
  - Quick fix for a warning
  - Quick fix for an error

To process the quick fix, click on the symbol. All suggestions to correct the problem are displayed in an overlaid window. As soon a suggestion is selected, a code preview is shown, so that the developer can see, what will be changed (Figure 7-49). Double-click one of the suggestion to process the quick fix.

Note: The problem symbol is grayed out but still visible. Save the source and the problem symbol disappears.



```
Bank bank = ITSOBank.getBank();

System.out.println("System is going to add new customer...");
Customer customer1 = new Customer("xxx-xx-xxxx", "Mrs", "Julia",
        "Johnson");
bank.addCustomer(customer1);
```

```
...
import itso.rad7.bank.exception.AccountAlreadyExistException;
import itso.rad7.bank.exception.CustomerAlreadyExistExcept
import itso.rad7.bank.exception.InvalidAccountException;
import itso.rad7.bank.exception.InvalidCustomerException;
...
e.printStackTrace();
} catch (CustomerAlreadyExistException e) {
// TODO Auto-generated catch block
e.printStackTrace();
}
}
}
```

```
        .println("Account " + account11.getAccountNumber()
                + " and account " + account12.getAccountNumber()
                + " have been successfully opened for " + customer1
                + ".\n");
```

1 error, 0 warnings, 0 infos

Description	Resource	Path	Location
Errors (1 item)			
Unhandled exception type CustomerAlreadyExistException	BankClient.java	RAD7Java/src/itso/rad7/bank/client	line 25

Figure 7-49   Java Editor with quick fix

Quick assist

Application Developer supports quick assists in the Java editor to provide suggestions to complete tasks quickly. Quick assist depends on the current curser position. When there is a suggestion and quick assist highlighting is enabled, a green light bulb 💡 is displayed on the left side of the code line.

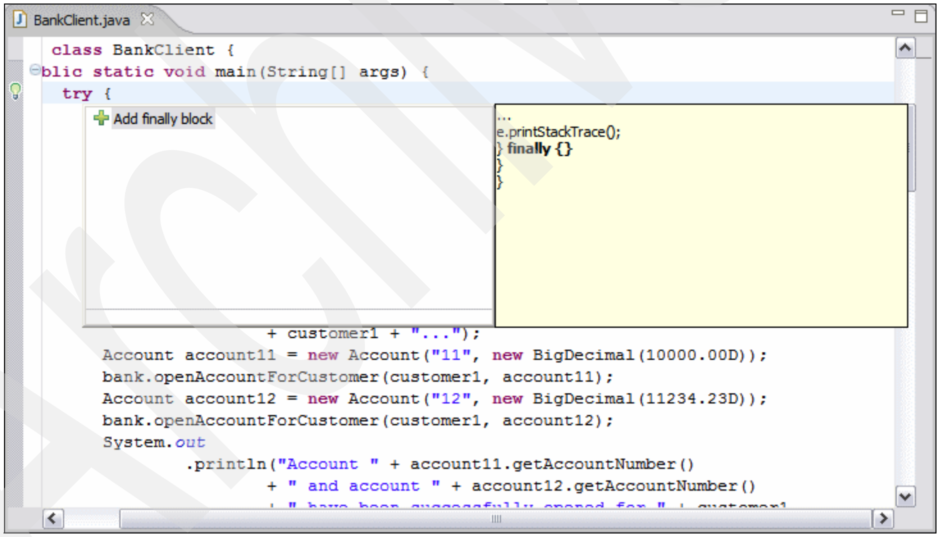
Enable quick assist highlighting

By default, the display of the quick assist light bulb is disabled. To enable it, do these steps:

- Select **Window** → **Preferences**.
- In the Preferences dialog, select **Java** → **Editor**.
- Select **Light bulb for quick assists**.

Invoking quick assist

To use the quick assist feature, double-click 💡 or press **Ctrl+1** to provide a list of intelligent suggestions. Select one to complete the task (Figure 7-50).



```
 class BankClient {
blic static void main(String[] args) {
  try {
```

```
...
e.printStackTrace();
} finally {}
}
}
```

```
                + customer1 + "...");
    Account account11 = new Account("11", new BigDecimal(10000.00D));
    bank.openAccountForCustomer(customer1, account11);
    Account account12 = new Account("12", new BigDecimal(11234.23D));
    bank.openAccountForCustomer(customer1, account12);
    System.out
            .println("Account " + account11.getAccountNumber()
                    + " and account " + account12.getAccountNumber()
```

Figure 7-50   Java Editor with quick assist

Content assist

This feature displays possible code completions that are valid with the current context.

Content assist preferences

To configure the Content Assist preferences, do these steps:

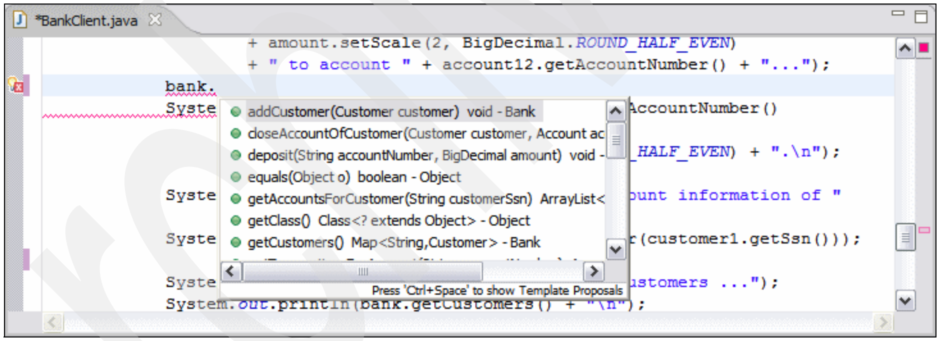
- Select **Window** → **Preferences**.
- In the Preferences dialog, select **Java** → **Editor** → **Content Assist**.
- Modify the settings as desired and click **Apply** and **OK**.

Invoke content assist

Press **Ctrl+Spacebar** at any point in the Java editor, to invoke the content assist.

The content assist provides the all possible code completions that are valid for the current context in a overlaid window (Figure 7-51). Double-click the desired completion, or use the arrow keys to select it, and press **Enter**.

> **Tip:** If there are still too many possible completions, just continue to write the code yourself and the amount of suggestions becomes smaller.
>
> Content assist can also be invoked to insert or to complete Javadoc tags.



```
                    + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
                    + " to account " + account12.getAccountNumber() + "...");
            bank.
            Syste                                        AccountNumber()
                                                         _HALF_EVEN) + ".\n");
            Syste                                        ount information of "
            Syste                                        r(customer1.getSsn()));
            Syste                                        ustomers ...");
            System.out.println(bank.getCustomers() + "\n");
```

```
addCustomer(Customer customer)  void - Bank
closeAccountOfCustomer(Customer customer, Account ac
deposit(String accountNumber, BigDecimal amount)  void -
equals(Object o)  boolean - Object
getAccountsForCustomer(String customerSsn)  ArrayList<
getClass()  Class<? extends Object> - Object
getCustomers()  Map<String,Customer> - Bank
```

Press 'Ctrl+Space' to show Template Proposals

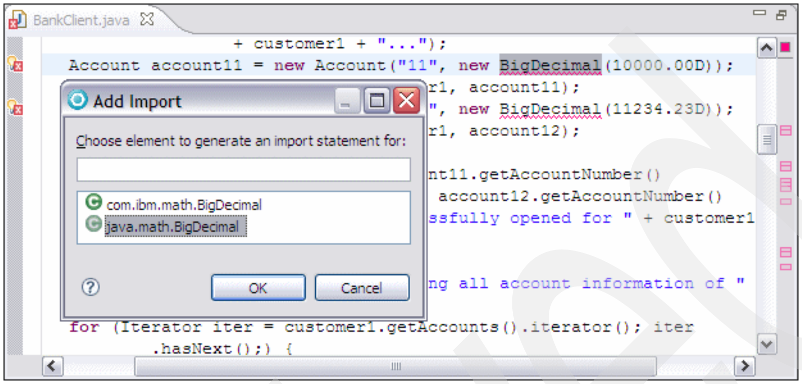
Figure 7-51 Java Editor with content assist

Import generation

The Java editor simplifies the task of finding the correct import statements to use in the Java code.

Simply right-click the unknown type in the code and select **Source** → **Add Import**, or select the type and press **Ctrl+Shift+M**. If the type is unambiguous, the import statement is directly added. If the type exists in more than one package, a window with all the types is displayed and you can select the correct type for the import statement.

Figure 7-52 shows an example where the selected type (`BigDecimal`) exists in several packages. Once you have determined that the `java.math` package is what you want, double-click the entry in the list, or select it and click **OK**, and the import statement is generated in the code.



```
                + customer1 + "...");
    Account account11 = new Account("11", new BigDecimal(10000.00D));
                                           r1, account11);
                                           ", new BigDecimal(11234.23D));
                                           r1, account12);

                                           nt11.getAccountNumber()
                                            account12.getAccountNumber()
                                           ssfully opened for " + customer1


                                           ng all account information of "

    for (Iterator iter = customer1.getAccounts().iterator(); iter
            .hasNext();) {
```

Choose element to generate an import statement for:

- com.ibm.math.BigDecimal
- java.math.BigDecimal

Figure 7-52 Java Editor with import generation

You can also add the required import statements for the whole compilation unit. Right-click a project, package, or Java type in the Package Explorer and select **Source** → **Organize Imports**, or select the project, package, or Java type and press **Ctrl+Shift+O**. The code in the compilation unit is analyzed and the appropriate import statements are added.

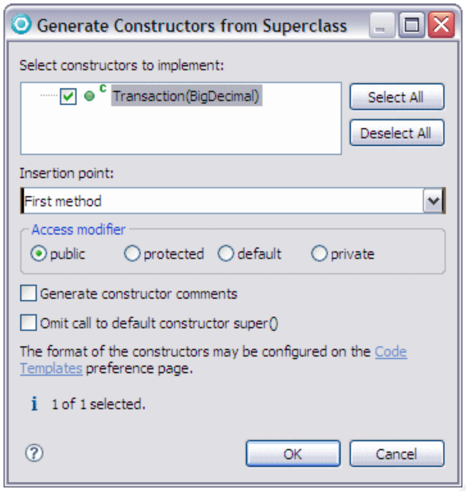
Adding constructors

This feature allows you to automatically add constructors to the open type. The following constructors can be added:

- Constructors from superclass
- Constructor using fields

Constructors from superclass

Add any or all of the constructors defined in the superclass for the currently opened type. Right-click anywhere in the Java editor and select **Source** → **Add Constructors from Superclass**. Select the constructors which you want to add to the current opened type, and click **OK** (Figure 7-53).



Generate Constructors from Superclass

Select constructors to implement:

☑ Transaction(BigDecimal)

Insertion point: First method

Access modifier: ● public ○ protected ○ default ○ private

☐ Generate constructor comments

☐ Omit call to default constructor super()

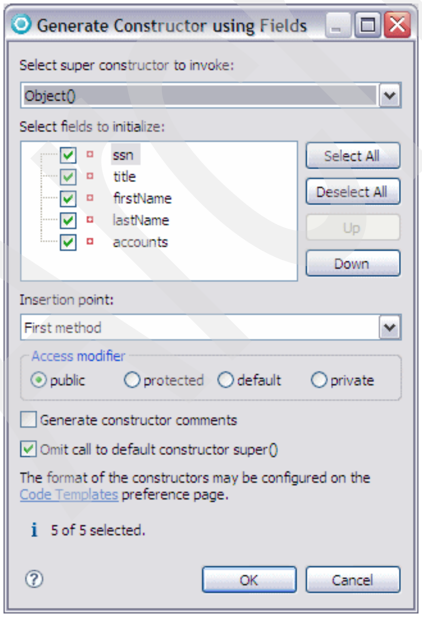
The format of the constructors may be configured on the Code Templates preference page.

1 of 1 selected.

Figure 7-53 Generate Constructors from Superclass dialog

Constructor using fields

This option adds a constructor which initialized any or all of the defined fields of the currently opened type. Right-click anywhere in the Java editor and select **Source** → **Add Constructors using Fields**. Select the fields which you want to initialize with the constructor and click **OK** (Figure 7-54).



Generate Constructor using Fields

Select super constructor to invoke: Object()

Select fields to initialize:

☑ ssn
☑ title
☑ firstName
☑ lastName
☑ accounts

Insertion point: First method

Access modifier: ● public ○ protected ○ default ○ private

☐ Generate constructor comments

☑ Omit call to default constructor super()

The format of the constructors may be configured on the Code Templates preference page.

5 of 5 selected.

Figure 7-54 Generate Constructor using Fields dialog

Example 7-12 shows the constructor code which would be generate with the settings shown in Figure 7-54.

Example 7-12 Generated constructor code

```
public Customer(String ssn, String title, String firstName, String lastName,
                   ArrayList<Account> accounts) {
   this.ssn = ssn;
   this.title = title;
   this.firstName = firstName;
   this.lastName = lastName;
   this.accounts = accounts;
}
```

Delegate method generator

The delegate method generator feature allows you to delegate methods from one class to another for better encapsulation. We use a simple example to explain this feature. A car has an engine, and a driver wants to start his car. Figure 7-55 shows that the engine is not encapsulated, the `PoorDriver` has to use the `Car` and the `Engine` class to start his car.

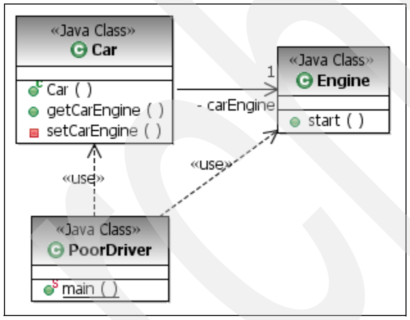


Figure 7-55 Simple car example class diagram (before method delegation)

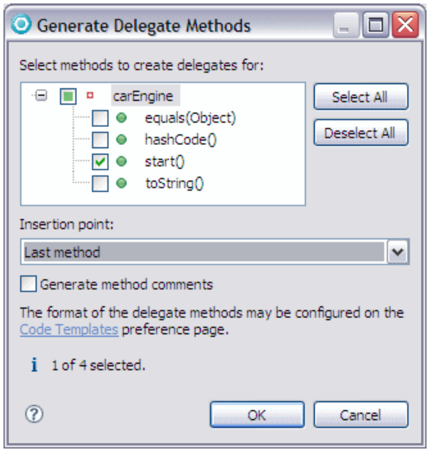
Example 7-13 shows how the poor driver has to start his car.

Example 7-13 Car, Engine, and PoorDriver classes (compressed)

```
// Car class
package itso.rad7.example;
import itso.rad7.example.Engine;
public class Car {
   private Engine carEngine = null;
   public Car(Engine carEgine) {
      this.setCarEngine(carEngine);
   }
   public Engine getCarEngine() {
      return carEngine;
   }
   private void setCarEngine(Engine carEngine) {
      if (carEngine != null) {
         this.carEngine = carEngine;
      } else {
         this.carEngine = new Engine();
      }
   }
}

// Engine class
package itso.rad7.example;
public class Engine {
   public void start() {
   // code to start the engine
   }
}

// PoorDriver class
package itso.rad7.example;
import itso.rad7.example.Car;
public class PoorDriver {
   public static void main(String[] args) {
      Car myCar = new Car(null);
      /* How can I start my car?
       * Do I really have to touch the engine?
       * - Yes, there is no other way at the moment.
       */
      myCar.getCarEngine().start();
   }
}
```

To make the driver happy, we delegate the `start` method from the `Engine` class to the `Car` class. To delegate a method, do these steps:

- Right-click the `carEngine` field in the Car class and select **Source** → **Generate Delegate Methods**.
- In the Generate Delegate Methods dialog, select only the `start` method and click **OK** (Figure 7-56).



Generate Delegate Methods

Select methods to create delegates for:

- ▣ carEngine
  - □ equals(Object)
  - □ hashCode()
  - ☑ start()
  - □ toString()

Select All

Deselect All

Insertion point:

Last method

□ Generate method comments

The format of the delegate methods may be configured on the Code Templates preference page.

1 of 4 selected.

OK

Cancel

Figure 7-56   Generate Delegate Method dialog

- This action adds the `start` method to the `Car` class, and code is added in the body of the method to delegate the method call to the `Engine` class through the `carEngine` attribute.
- Figure 7-57 and Example 7-14 shows how the `HappyDriver` can start the car.

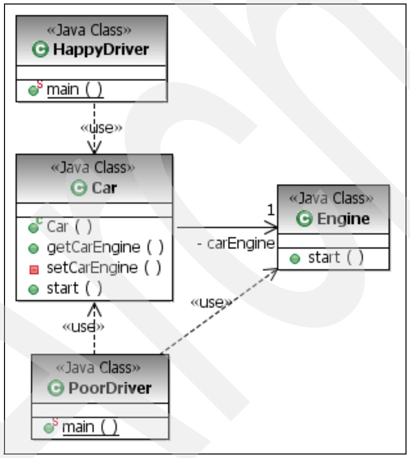


Figure 7-57   Simple car example class diagram (after method delegation

Example 7-14   Car and HappyDriver class

```
// Car class
package itso.rad7.example;
import itso.rad7.example.Engine;
public class Car {
   private Engine carEngine = null;
   public Car(Engine carEgine) {
      this.setCarEngine(carEngine);
   }
   public Engine getCarEngine() {
      return carEngine;
   }
   private void setCarEngine(Engine carEngine) {
      if (carEngine != null) {
         this.carEngine = carEngine;
      } else {
         this.carEngine = new Engine();
      }
   }
   public void start() {
      carEngine.start();
   }
}

// HappyDriver class
package itso.rad7.example;
public class HappyDriver {
   public static void main(String[] args) {
      Car myAdvancedCar = new Car(null);
      // Start the car - I don't care about technical details
      myAdvancedCar.start();
   }
}
```

Refactoring

During the development of a Java application, it might be necessary to perform tasks, such as renaming classes, moving classes between packages, and breaking out code into separate methods. Such tasks are both time consuming and error prone, because it is up to the programmer to find and update each and every reference throughout the project code. Application Developer provides a list of refactor actions to automate the this process.

The Java development tools (JDT) of Application Developer provide assistance for managing refactor. In the each refactor wizard you can select:

- **Refactor with preview**: Click **Next** in the dialog to bring up a second dialog panel where you are notified of potential problems and are given a detailed preview of what the refactor action will do.
- **Refactor without preview**: Click **Finish** in the dialog and have the refactor performed. If a stop problem is detected, refactor cancels and a list of problems is displayed.

Table 7-7 provides a summary of common refactor actions.

Table 7-7 Refactor actions

Name	Function
Rename	Starts the Rename Compilation Unit wizard. Renames the selected element and (if enabled) corrects all references to the elements (also in other files). It is available on methods, fields, local variables, method parameters, types, compilation units, packages, source folders, projects, and on a text selection resolving to one of these element types. Right-click the element and select **Refactor** → **Rename**, or select the element and press **Alt+Shift+R**, or select **Refactor** → **Rename** from the menu bar.
Move	Starts the Move wizard. Moves the selected elements and (if enabled) corrects all references to the elements (also in other files). Can be applied on one or more static methods, static fields, types, compilation units, packages, source folders and projects, and on a text selection resolving to one of these element types. Right-click the method signature and select **Refactor** → **Move**, or select the method signature and press **Alt+Shift+V**, or select **Refactor** → **Move** from the menu bar.
Change Method Signature	Starts the Change Method Signature wizard. You can change the visibility of the method, change parameter names, parameter order, parameter types, add parameters, and change return types. The wizard updates all references to the changed method. Right-click the element and select **Refactor** → **Change Method Signature**, or select the element and press **Alt+Shift+C**, or select **Refactor** → **Change Method Signature from** the menu bar.
Extract Interface	Starts the Extract Interface wizard. You can create an interface from a set of methods and make the selected class implements the newly created interface. Right-click the class and select **Refactor** → **Extract Interface**, or select the element and select **Refactor** → **Extract Interface** from the menu bar.
Push Down	Starts the Push Down wizard. Moves a field or method to its subclasses. Can be applied to one or more methods from the same type or on a text selection resolving to a field or method. Right-click the type and select **Refactor** → **Push Down**, or select the element and select **Refactor** → **Push Down** from the menu bar.

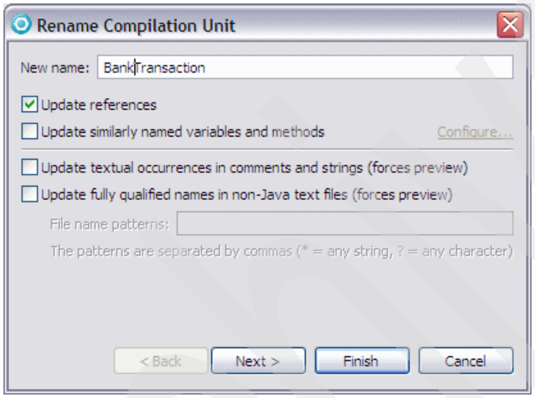
Name	Function
Pull Up	Starts the Pull Up wizard. Moves a field or method to its superclass. Can be applied on one or more methods and fields from the same type or on a text selection resolving to a field or method. Right-click the type and select **Refactor** → **Push Up**, or select the element and select **Refactor** → **Push Up** from the menu bar.
Extract Method	Starts the Extract Method wizard. Creates a new method containing the statements or expressions currently selected, and replaces the selection with a reference to the new method. Right-click the statement or expression and select **Refactor** → **Extract Method**, or select it and press **Alt+Shift+M**, or select **Refactor** → **Extract Method** from the menu bar.
Extract Local Variable	Starts the Extract Local Variable wizard. Creates a new variable assigned to the expression currently selected and replaces the selection with a reference to the new variable. Right-click the expression and select **Refactor** → **Extract Local Variable**, or select it and press **Alt+Shift+L**, or select **Refactor** → **Extract Local Variable** from the menu bar.
Extract Constant	Starts the Extract Constant wizard. Creates a static final field from the selected expression and substitutes a field reference, and optionally replaces all other places where the same expression occurs. Right-click the expression and select **Refactor** → **Extract Constant**, or select **Refactor** → **Extract Constant** from the menu bar.
Inline	Starts the Inline Method wizard. Inlines local variables, non abstract methods, or static final fields. Right-click the element and select **Refactor** → **Inline**, or select the element and press **Alt+Shift+I**, or select **Refactor** → **Inline** from the menu bar.
Encapsulate Field	Starts the Encapsulate Field wizard. Replaces all references to a field with getter and setter methods. Is applicable to a selected field or a text selection resolving to a field. Right-click the field, and select **Refactor** → **Encapsulate Field...**, or select the element and select **Refactor** → **Encapsulate Field...** from the menu bar.

Refactor example (rename a class)

The following example of a refactor operation assumes that you want to rename the class `Transaction` to `BankTransaction` in the `RAD7Java` project.

To rename the `Transaction` class to `BankTransaction`, do these steps:

- Right-click the `Transaction` class in the Package Explorer and select **Refactor** → **Rename**.
- In the Rename Compilation Unit wizard, enter the following data (Figure 7-58):
  - New name: `BankTransition`
  - Select **Update references** (default).
  - Clear other check boxes.



Rename Compilation Unit

New name: BankTransaction

☑ Update references

☐ Update similarly named variables and methods

☐ Update textual occurrences in comments and strings (forces preview)

☐ Update fully qualified names in non-Java text files (forces preview)

File name patterns:

The patterns are separated by commas (* = any string, ? = any character)

Figure 7-58 Refactor - Rename Compilation Unit wizard

- Click **Next** to see the preview and potential problems.
- Click **Finish** to process the rename task.

> **Note:** If there are any files with unsaved changes in the workspace and you have not indicated in the preferences that the save has to be done automatically, you are prompted to save these files before continuing the refactor operation.
>
> If there are problems or warnings, the wizard displays the **Found Problems** window. If the problems are severe, the **Continue** button is disabled and the refactor must be aborted until the problems have been corrected.

More information

We highly recommend the help feature provided by Application Developer. Click **Help** → **Help Contents** → **Developing Java applications**. There is also a Watch and Learn tutorial available, called Create a Hello World Java Application. To start the tutorial click **Help** → **Welcome** → **Tutorials** → **Create a Hello World Java application**.

The following URLs provide further information about Eclipse and Java technology:

- **Sun Java™ SE Technology Home page**—Contains links to specifications, API Javadoc, and articles about Java SE:

  `http://java.sun.com/javase/index.jsp`

- **IBM Developerworks Java Technology**—Java news, downloads and CDs, and learning resources:

  `http://www.ibm.com/developerworks/java/`

- **Eclipse Open Source Community**—Official home page of the eclipse open source community:

  `http://www.eclipse.org/`

8

Accelerate development using patterns

This chapter introduces the concept of pattern implementation, how it relates to pattern specification, and the benefits that pattern implementation can bring to software application development.

The focus of the chapter is to demonstrate by example how to use the Rational Application Developer patterns tooling to develop and use patterns.

The chapter is organized into the following sections:

- Introduction to pattern implementation
- Creating a pattern implementation
- Applying the pattern

Introduction to pattern implementation

As a developer, you are probably familiar with the concept of pattern as it relates to software development, where a pattern is a proven solution to a common problem within a given context. However, you might be less familiar with the concept of pattern implementation and how it would help you accelerate your development.

Pattern specification and pattern implementation

As do many developers, you probably read patterns-related books, where a pattern is described along with its benefits, the problem it solves, and the context it should be applied. This description is what is commonly referred as the *pattern specification*, and its intent is to help you understand how and when to use the pattern.

The idea of *pattern implementation* came from the desire to go further than just using the pattern specification as a blueprint—to try to automate as much as possible the application of the pattern. The result is to codify the pattern specification into a pattern implementation, allowing you to automatically apply it into a given environment.

The core idea behind pattern implementation is that as we have codified and automated best practices, applying them is now faster than doing it manually, and more consistent, because it involves less manual modifications.

Pattern implementation and Application Developer

In the context of Application Developer, the pattern implementation—rather than being derived by a community design patterns—comes from your best practices and existing successful implementations you would like to apply to other projects. This approach is called *exemplar authoring* in Application Developer.

Exemplar authoring consists of using an *exemplar* to build the pattern implementation. An exemplar is a representative output of the pattern and contains an instance of each artifact you expect the pattern implementation to generate. A good examplar should be built following the applicable best practices and must allow you to define the points of variability of the pattern and the input model. All the purpose of the exemplar authoring is to leverage your best practices and existing assets to increase the development efficiency and productivity. To do so, you codify the best practices to make their application automatic.

The examplar authoring tool leverages the Eclipse's Java Emitter Templates (JET), which is part of the Eclipse Modeling Framework (EMF). JET is similar to JavaServer Pages syntax and is powerful to generate Java, SQL, and any other text based files.

JET allow you to customize the output artifacts into an XML input file and a set of templates. The structure of this input file, as well as how it impacts the different artifacts, is based on what you are doing when creating your transformation using the Application Developer Exemplar Authoring tools. Because the tools produce JET, exemplar authoring transformations are also sometimes referred as JET transformations.

Exemplar authoring is not only a set of tools, but more importantly, it is a process to create pattern based into exemplars.

Exemplar authoring process

A key aspect to succeeding with creating patterns is that there is a proven repeatable set of steps that you can follow in building your own patterns. This approach to building patterns is known as *exemplar analysis*. In this process we take steps to identify what artifacts have to be generated by the pattern and determine which elements of the pattern are dynamic versus static. We use those elements that are dynamic as the basis for the input model for the pattern. Within the input model we have to identify the roles, their cardinality, and their attributes.

At a high level, these are the steps we follow as part of exemplar analysis:

1. **Identify artifact roles**. When examining the exemplar, we often find that there are multiple elements that are based on the same abstraction. For instance, in the case of JavaBeans we often see that there is a pair of elements: An interface and a class that adheres to the interface. Our exemplar might contain multiple JavaBeans, but if the intent of our pattern is to generate JavaBeans, we just have to pick out the most representative pair and can ignore the other repeating cases. In this case, although the exemplar contains multiple beans, we have just a single role for the class and one for the interface. An important aspect of this identification is that a JET template is associated with each role.
2. **Create role groups**. Using the exemplar authoring tooling, we then have to group the roles based on their cardinality. One time roles appear at the highest level, and then we create subgroups for roles that repeat. Many subgroups can exist with all elements within a subgroup sharing the same cardinality. For instance, an Eclipse project can contain a number of artifacts such as a `.project` file and a `.classpath` file, which occur only once per project. However, we might have multiple JavaBeans within a project. As such

we would place the `.project` and `.classpath` roles into a group at the top level, and then place the JavaBeans role into subgroup.

At this point we have to keep in mind that each role will have a JET template associated with it. When we run the pattern, the JET template is used to generate instance of artifacts based on the role. Also, the input model, which by default is an XML file, contains a set of elements based on the groups defined. However, at this point, our elements do not provide any useful information. Therefore, we now have to focus on the information that is assigned to the element.

3. **Identify attributes**. At this point we know the type of dynamic information that we have to pass into the pattern. However, we have to be more specific in terms of the information required. For example, in the case of a JavaBean we likely have to know what name to provide, a package name, and a directory location. As such, we create a set of attributes that we assign to the role groups, which in turn become attributes on the XML elements. As part of this effort we have to perform a normalization step, whereby we review the attributes to determine which attributes are atomic and which can be derived. A derived attribute is one that can be calculated using information known about the pattern and details provided by atomic attributes.
4. **Customize templates.** Once we have completed the previous steps, we have to go through each of the JET templates and replace the static text with tags that allow us to access and process the dynamic information provided by the input model.

Prepare for the sample

We use the ITSO Bank sample application created in Chapter 7, “Develop Java applications” on page 227, with same small changes as the examplar for our pattern implementation.

Import the `c:\7501code\patterns\RAD7Patterns.zip` project interchange file into Application Developer. Refer to Appendix B, “Additional material” on page 1307.

After importing the ITSO Bank sample, verify that it runs properly in the Java perspective (`BankClient`). For more information refer to “Running the ITSO Bank application” on page 281.

Note: The **BankClient.java** is now in the **RAD7PatternsClient** project because we split the initial **RAD7Java** project into 2 different projects:

- **RAD7Patterns:** Model and implementation
- **RAD7PatternsClient:** Client application and solution files (**assets**)

Creating a pattern implementation

In this section we demonstrate how Application Developer examplar authoring tools can be used to develop a pattern implementation. The pattern we decide to implement allows us to easily create different test clients for our bank application. We decide to use the BankClient application as our exemplar because it is our best implementation of a test client for this bank application.

Note: The example we will be walking you through in the rest of the chapter is probably not the best representation of a pattern implementation as its scope of application is small (we will generate bank test clients). We selected this example because it was providing enough variability to allow you to understand how the exemplar authoring works, but not too much to not overwhelm you by language and command details.

A more interesting example is described in "Facade pattern" on page 351.

Here are the tasks we perform to develop our pattern implementation:

- Create a new JET Transform project
- Populating the transformation model
- Adding and deriving attributes
- Generate and edit templates

Create a new JET Transform project

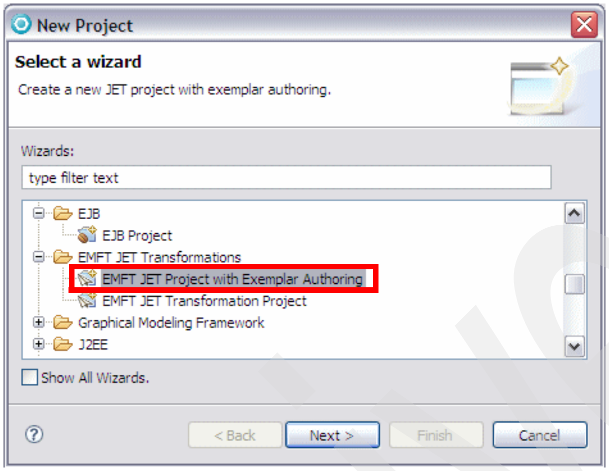
A JET Transform project contains all the elements required for JET transformations, including the transformation model. Table 8-1 provides an overview of the specific files contained into this kind of project.

Table 8-1 Files specific to a JET transform project

File(s)	Description
`*.tma`	The transformation model
`*.jet`	JET templates files, initially generated from the transformation model.
`input.ecore`	EMF input model generated from the transformation model
`schema.xsd`	XML schema corresponding to the input model
`sample.xml`	Sample of input file that would be used by the transformation

To create a new JET Transform project, do these steps:

- From the workbench, select **File** → **New** → **Other**.
- In the **New** dialog, select **EMF JET Transformations** → **EMF JET Project with Exemplar Authoring** (Figure 8-1), and click **Next**.



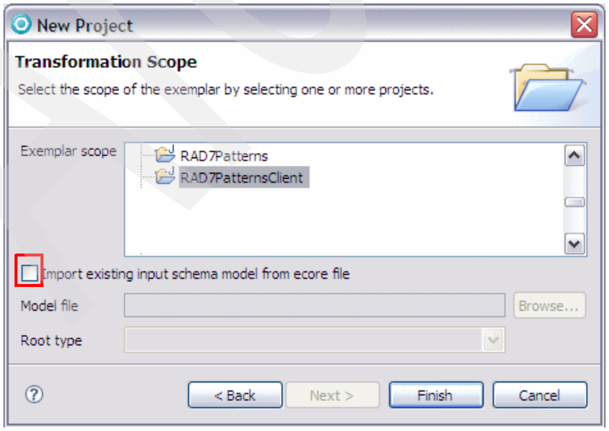
New Project

Select a wizard

Create a new JET project with exemplar authoring.

Figure 8-1 Create an EMF JET Transform project

- In the Create Jet Project with Exemplar Authoring dialog, enter **RAD7PatternsTransform** as the project name. Leave the **Use default location** checked, and click **Next**.
- In the Transformation Scope dialog, select the **RAD7PatternsClient** project, leave the **Import existing input schema model from ecore file** cleared (Figure 8-2).



New Project

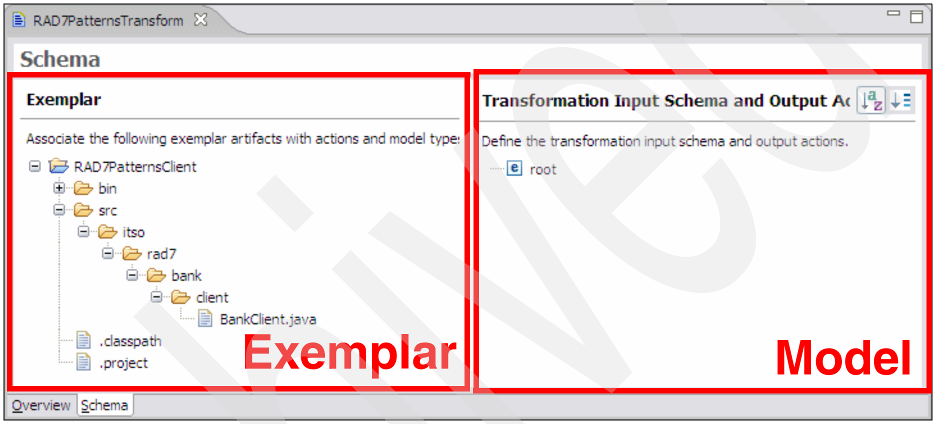
Transformation Scope

Select the scope of the exemplar by selecting one or more projects.

Figure 8-2 Transformation Scope

Note: The **Import existing input schema model from ecore file** allows you to seed your transformation model from an existing ecore file, allowing you to extend an existing transformation without reentering all the information provided in the base one.

- Click **Finish** and the EMF JET Transformation project is created. The **Exemplar Authoring** editor now displays the **RAD7JavaClient** exemplar and an empty model (Figure 8-3). The underlying file of this transformation is called `transform.tma` (in case you have to reopen the transformation).



Associate the following exemplar artifacts with actions and model type:

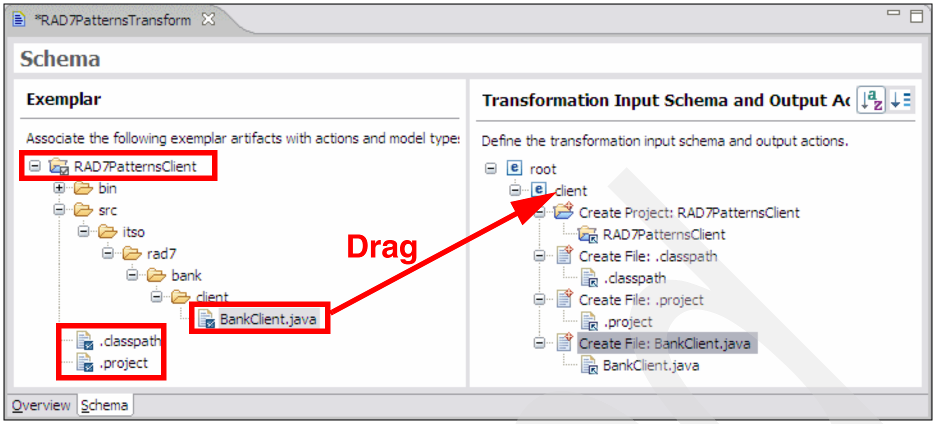
Define the transformation input schema and output actions.

Figure 8-3 Exemplar Authoring editor

Populating the transformation model

Now that the project is created, we can start populate the model with the different elements we want the transformation to create.

- The first thing to do is to create a new type to contain our transformation. To do so, right-click on the **root** element and select **Type**. Type **client** for the new type.
- The analysis of the examplar lets us identify the files as the artifacts we want the transform to generate:
  - The Java project **RAD7PatternsClient**
  - The project metadata files **.classpath** and **.project**
  - The main class **itso.rad7.bank.client.BankClient.java**
- We add these artifacts to the transformation model by dragging them from the left pane (exemplar) to the right pane (model) onto the **client** type we just created (Figure 8-4).



Associate the following exemplar artifacts with actions and model type:

Define the transformation input schema and output actions.

*Figure 8-4   Artifacts added to the transformation model under the **client** type*

> **Note:** As you can see in Figure 8-4, each time you drag an artifact from the exemplar to the model:
>
> - A create action (create project or create file) is created into the model.
> - The corresponding artifact into the exemplar is marked by a blue check mark.

- Each of the create actions will create the corresponding Eclipse resource. Below are the names and paths of the associated exemplar artifacts:

  – `RAD7PatternsClient`
  – `RAD7PatternsClient/.classpath`
  – `RAD7PatternsClient/.project`
  – `RAD7PatternsClient/src/itso/rad7/bank/client/BankClient.java`

  Some of the components of the names and paths that are likely to vary from one test client to another:

  – `RAD7PatternsClient` (project name)
  – `itso/rad7/bank/client` (client directory corresponding to the client package)

  According to JET transformation best practices, these variable names have to be stored in attributes and derived attributes.

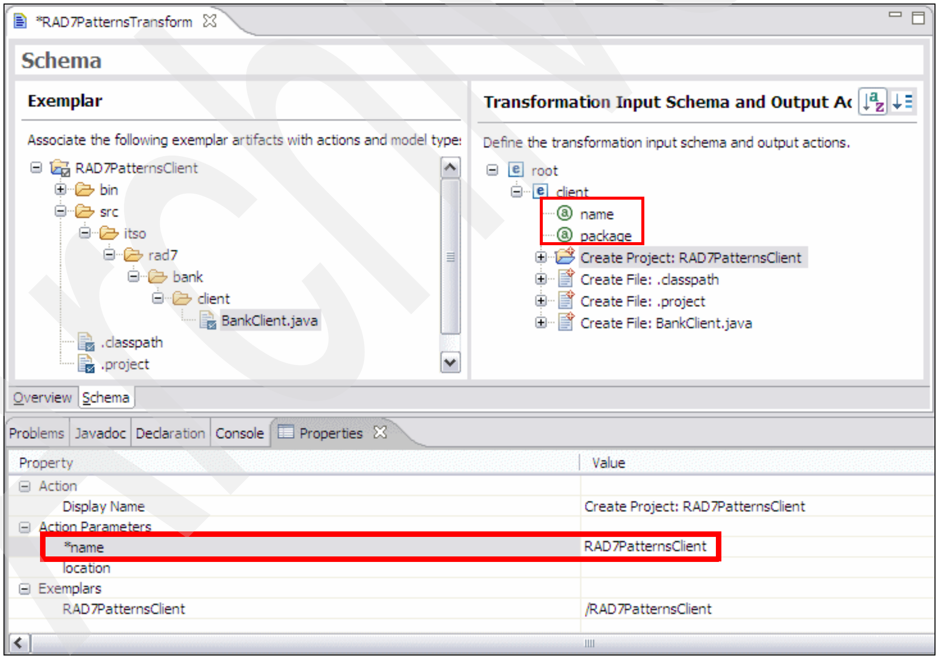
Adding and deriving attributes

In JET transformations, variable information is stored into attributes. The exemplar authoring tool identifies two different types of attributes:

- **Attribute**: Input attribute that has to be provided by the input model
- **Derived attribute:** Attribute derived from an input attribute, usually used to satisfy a naming convention (such as artifacts, Java variables)

Let us now add the necessary attributes and derived attributes:

- Select the **client** type, right-click, and select **New** → **Attribute**. Call this new attribute **name**.
- Repeat the previous operation to add the **package** attribute.
- Select the **Create Project: RAD7PatternsClient** action and view its properties into the Properties view (Figure 8-5). The **name** action parameter is used by the transformation to name the test client project when it is first created. As we said above, this name must be variable and related to the **name** attribute we just created. This is what we do in the next steps.



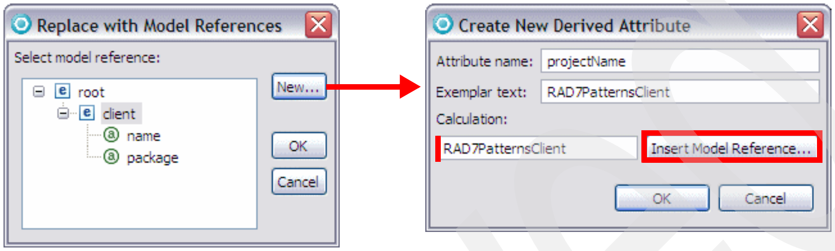
Associate the following exemplar artifacts with actions and model type:

Define the transformation input schema and output actions.

Property	Value
Action	
Display Name	Create Project: RAD7PatternsClient
Action Parameters	
*name	RAD7PatternsClient
location	
Exemplars	
RAD7PatternsClient	/RAD7PatternsClient

Figure 8-5 Create Project Properties

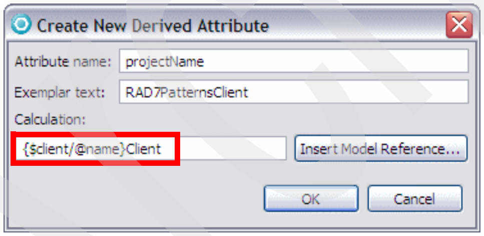
- Select the entire text of the action parameters name, right-click, and select **Replace with Model Reference**.
- In the Replace with Model References dialog, select the **client** type and click **New**, because we want to add the **Client** string to the name attribute.
- In the Create New Derived Attribute dialog, enter **projectName** as Attribute Name. Point the cursor to the start of the Calculation field and click **Insert Model Reference** (Figure 8-6).


Replace with Model References
Select model reference:
root
client
name
package
New...
OK
Cancel
Create New Derived Attribute
Insert Model Reference...
OK
Cancel

Attribute name: projectName
Exemplar text: RAD7PatternsClient
Calculation: RAD7PatternsClient

Figure 8-6 Insert Model Reference

- In the Select Model Reference dialog, select the **name** attribute of the client type and click **OK**.
- Note that a query expression for the **name** attribute as been inserted as `{$client/@name}RAD7PatternsClient`. Change the expression to `{$client/@name}Client` to define the calculation correctly (Figure 8-7). Click **OK** to get back to **Replace with Model References** dialog.

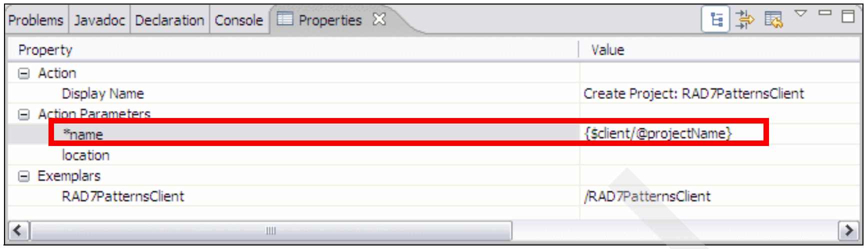


Attribute name: projectName
Exemplar text: RAD7PatternsClient
Calculation: {$client/@name}Client

Figure 8-7 Final definition of the projectName derived attribute

> **Note:** The syntax used for the calculation is related to the fact that access to the variable content is done by navigating an XML Domain Object Model (DOM) using XPath.

- In the Replace with Model References dialog, select the **projectName** attribute and click **OK** (Figure 8-8).

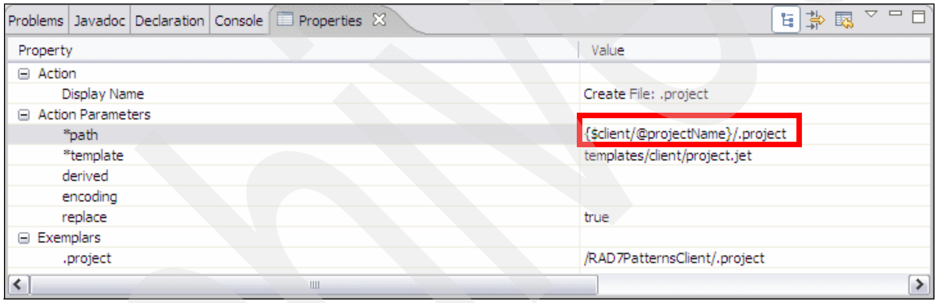


Property	Value
⊟ Action	
Display Name	Create Project: RAD7PatternsClient
⊟ Action Parameters	
*name	{$client/@projectName}
location	
⊟ Exemplars	
RAD7PatternsClient	/RAD7PatternsClient

Figure 8-8   Final project name variable

- Using the same approach to replace the **path** parameter of the **.classpath** and **.project** elements. Select **RAD7PatternsClient** and replace the text with a model reference to the **projectName** attribute (Figure 8-9):

```
{$client/$projectName}/.classpath
{$client/$projectName}/.project
```



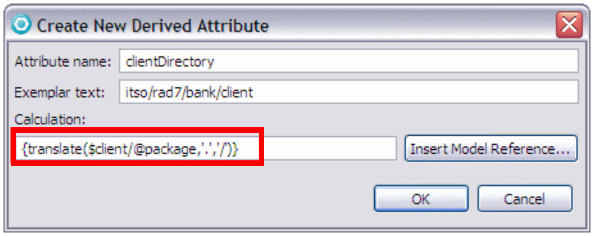
Property	Value
⊟ Action	
Display Name	Create File: .project
⊟ Action Parameters	
*path	{$client/@projectName}/.project
*template	templates/client/project.jet
derived	
encoding	
replace	true
⊟ Exemplars	
.project	/RAD7PatternsClient/.project

Figure 8-9   Adding a reference to .classpath and .project path variables

- Select the **Create File: BankClient.java** action and in the Properties view replace **RAD7PatternsClient** with a model reference to the **projectName** attribute:

```
{$client/$projectName}/src/itso/rad7/bank/client/BankClient.java
```

- We defined **package** as a variable, therefore we want to replace **itso/rad7/bank/client** by a new derived attribute. This derived attribute is called **clientDirectory** (Figure 8-10).



Attribute name: clientDirectory

Exemplar text: itso/rad7/bank/client

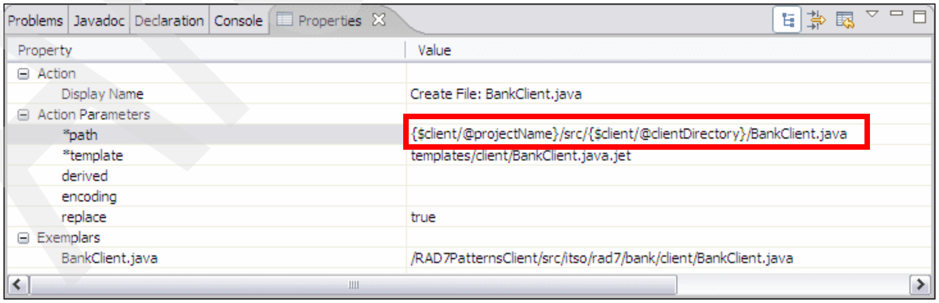
Calculation: {translate($client/@package,'.','/')}

Figure 8-10 Attribute deriving the directory path from the package attribute

> **Note:** Be careful to click on the **client** type before adding the new derived attribute. Consider the expression on the **Calculation** box:
>
> ```
> {translate($client/@package,'.','/')}
> ```
>
> This translates the **package** attribute (in the form `itso.rad7.bank.client`) into a directory path (in the form `itso/rad7/bank/client`).
>
> The `translate` function is a standard XPath function. To learn more about the standard XPath function go to `http://www.w3.org/TR/xpath`. To learn more about the Rational Application Developer additional XPath functions look into the help at **Extending Rational Application Developer functionality** → **Authoring JET transformations** → **EMFT JET Developer Guide** → **XPath Function Reference** → **Additional XPath Functions**.

- Select the **Create File: BankClient.java** action and in the Properties view replace the package directory with a reference to the **clientDirectory** attribute (Figure 8-11):

```
{$client/$projectName}/src/{$client/$clientDirectory}/BankClient.java
```



Property	Value
⊟ Action	
Display Name	Create File: BankClient.java
⊟ Action Parameters	
*path	{$client/@projectName}/src/{$client/@clientDirectory}/BankClient.java
*template	templates/client/BankClient.java.jet
derived	
encoding	
replace	true
⊟ Exemplars	
BankClient.java	/RAD7PatternsClient/src/itso/rad7/bank/client/BankClient.java

Figure 8-11 Final version of the BankClient.java path parameter

- Save the transformation. The completed model is shown in Figure 8-12.

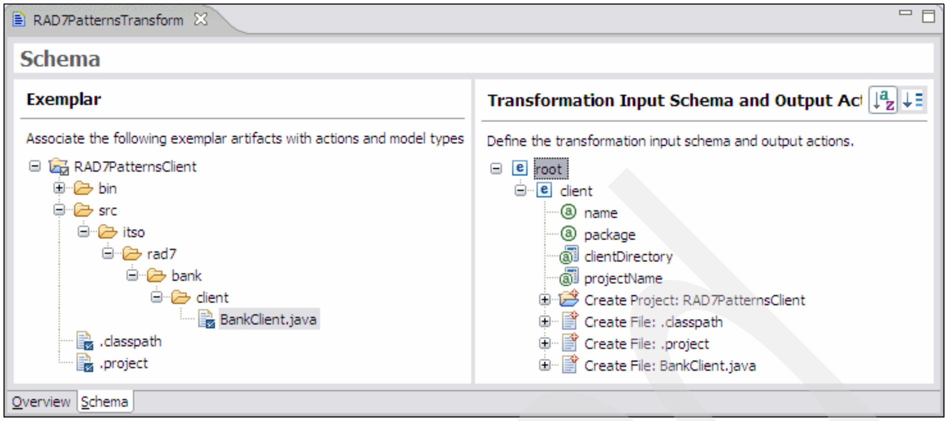

RAD7PatternsTransform
Schema
Exemplar
Associate the following exemplar artifacts with actions and model types
RAD7PatternsClient
bin
src
itso
rad7
bank
client
BankClient.java
.classpath
.project
Transformation Input Schema and Output Ac
Define the transformation input schema and output actions.
root
client
name
package
clientDirectory
projectName
Create Project: RAD7PatternsClient
Create File: .classpath
Create File: .project
Create File: BankClient.java
Overview
Schema

Figure 8-12 Final model for the JET transformation

Generate and edit templates

So far we have only made file names and paths variable; with templates we can make the file content variable. Templates are the means used by JET to allow us to modify the content of the files generated based on attributes (and derived attributes) provided by the input model.

Let us generate the templates and insert variables into the content of these templates:

- Right-click in the model transformation pane (for example near root) and select **Update Project**. New Templates are generated into the project (Figure 8-13).

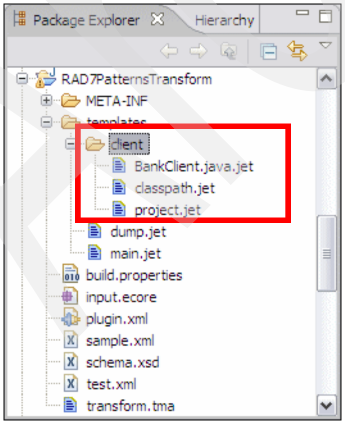


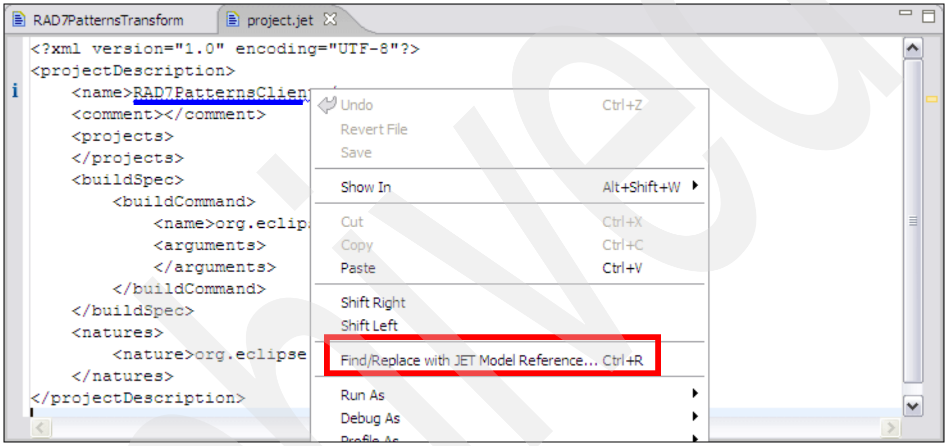
Figure 8-13 New generated templates

Updating the project.jet file

Double-click **project.jet** to edit the template.

> **Note:** There is a blue underscore under the name element, indicating that the underscored string matches the exemplar strings of one of the attributes. This indicates that the string should probably be replaced by a variable expression referencing this attribute.

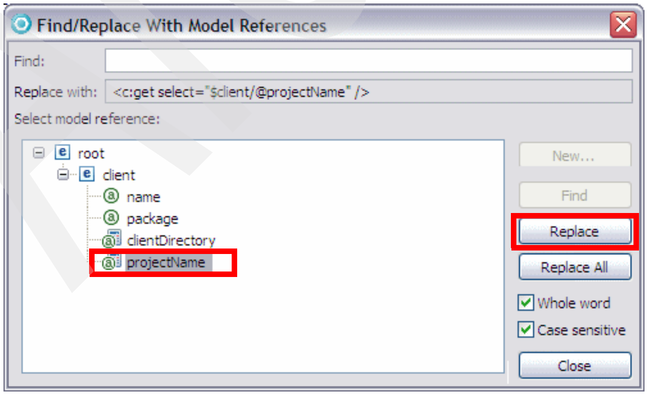
- Select the underscored text, right-click, and select **Find/Replace with JET Model Reference** (Figure 8-14)



```
<?xml version="1.0" encoding="UTF-8"?>
<projectDescription>
    <name>RAD7PatternsClien
    <comment></comment>
    <projects>
    </projects>
    <buildSpec>
        <buildCommand>
            <name>org.eclip
            <arguments>
            </arguments>
        </buildCommand>
    </buildSpec>
    <natures>
        <nature>org.eclipse
    </natures>
</projectDescription>
```

Figure 8-14 Launching Find/Replace with JET Model Reference

- In the **Find/Replace with JET Model Reference** dialog, select **projectName**, click **Replace**, and then click **Close** (Figure 8-15).



Find:

Replace with: <c:get select="$client/@projectName" />

Select model reference:

☑ Whole word

☑ Case sensitive

*Figure 8-15 Replacing the project names with the **projectName** attribute*

- The name in the template has been replaced by the correct `<c:get>` tag:

```
<name><c:get select="$client/@projectName" /></name>
```

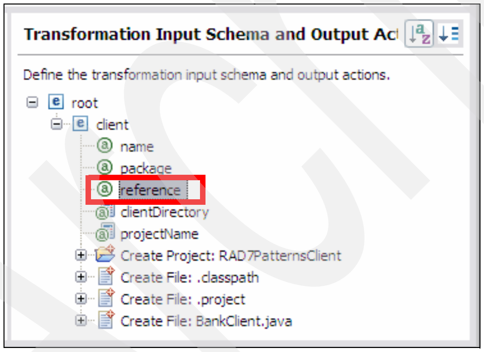
- Save and close `project.jet`.

Updating the classpath.jet file

Open **classpath.jet**. The classpath contain a reference to the **RAD7Patterns** project, that implements the bank application:

```
<?xml version="1.0" encoding="UTF-8"?>
<classpath>
   <classpathentry kind="src" path="src"/>
   <classpathentry kind="con"
                   path="org.eclipse.jdt.launching.JRE_CONTAINER"/>
   <classpathentry combineaccessrules="false" kind="src"
                                          path="/RAD7Patterns"/>
   <classpathentry kind="output" path="bin"/>
</classpath>
```

- It would be good to make the project name a variable in case we decide to rename the project or decide to use a new project. However, we do not yet have an attribute to store this value. We have to add it to the model.
- Switch back to the **RAD7PatternsTransform** model and add a new attribute called **reference** under **client** (Figure 8-16).


Transformation Input Schema and Output Actions

Define the transformation input schema and output actions.

Figure 8-16 Adding reference attribute to the model

- Switch back to **classpath.jet** and replace **RAD7Patterns** (with the **/** at the beginning) by the new created attribute called **reference**.

```
<classpathentry combineaccessrules="false" kind="src"
                path="<c:get select="$client/@reference" />"/>
```

- Save and close `classpath.jet`.

Updating the BankClient.java.jet file

We update the client program with many templates:

- Open the `BankClient.java.jet` and replace the package name with the package attribute in the same manner:

```
......
package <c:get select="$client/@package" />;

import itso.rad7.bank.exception.ITSOBankException;
import itso.rad7.bank.ifc.Bank;
```

Using a customer template

- Analyzing the code of the **BankClient**, we can see that it creates one customer (**customer1**), two accounts for this customer (**account11** and **account12**), and executes a deposit transaction on the first account and a debit transaction on the second account. We make this variable, allowing us to create as many customers and accounts as we like and perform all the transactions we want.
- A customer is defined by four parameters (SSN, title, first name, and last name). We create it as a type and add the parameters as attributes. We also have to add an **id** attribute to allow us to create multiple customers.

  Switch back to the transformation model, select the **client** type, and select **New** → **Type**. Name the new type **customer**. Add to the customer type four attributes called **ssn**, **title**, **firstName**, **lastName**, and **id**.
- When done, save all changes, right-click, and select **Update Project**, to make the model modifications available for use in the JET templates (Figure 8-17).

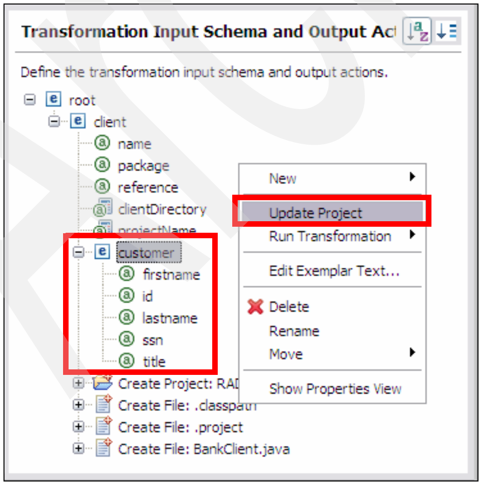
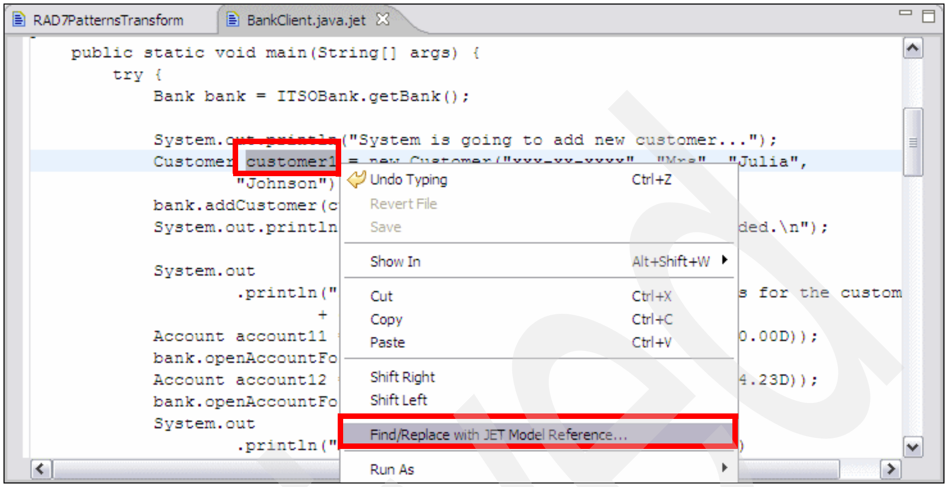


Figure 8-17 Adding the customer type to the model

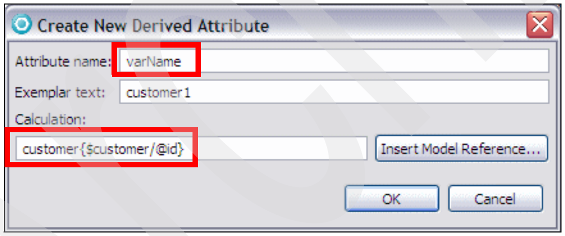
- Switch back to **BankClient.java.jet**. Select **customer1**, right-click and select **Find/Replace with JET Model Reference** (Figure 8-18).



```
    public static void main(String[] args) {
        try {
            Bank bank = ITSOBank.getBank();

            System.out.println("System is going to add new customer...");
            Customer customer1 = new Customer("xxx-xx-xxxx", "Mrs", "Julia",
                    "Johnson")
            bank.addCustomer(c
            System.out.println                                   ded.\n");

            System.out
                    .println("                                   s for the custom
                            +
            Account account11                                    0.00D));
            bank.openAccountFo
            Account account12                                    4.23D));
            bank.openAccountFo
            System.out
                    .println("                                   )
```

Figure 8-18 Replace customer1 by a variable

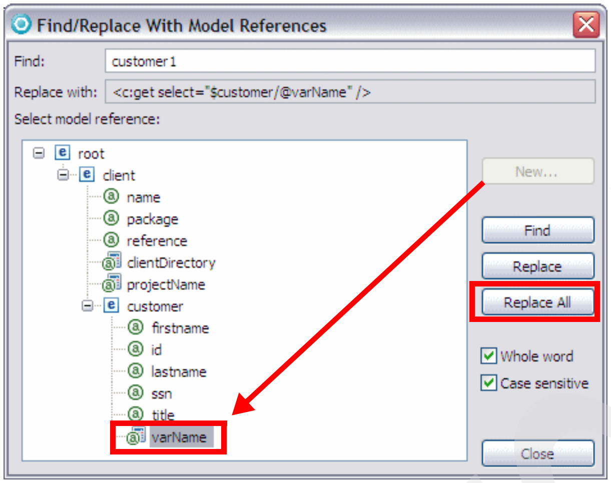
- In the Find/Replace with JET Model Reference dialog, select **customer** and click **New** to create a new derived attribute. Call the new derived attribute **varName** and type **customer{$customer/@id}** into the Calculation box (Figure 8-19). Click **OK** to go back to the previous dialog.



Attribute name: varName
Exemplar text: customer1
Calculation:
customer{$customer/@id}

Figure 8-19 Creating the varName derived attribute

- In the **Find/Replace with JET Model Reference** dialog, select **varName**, click **Replace All** to replace all the instances of **customer1**, and then click **Close** (Figure 8-20).



Find: customer1

Replace with: <c:get select="$customer/@varName" />

☑ Whole word

☑ Case sensitive

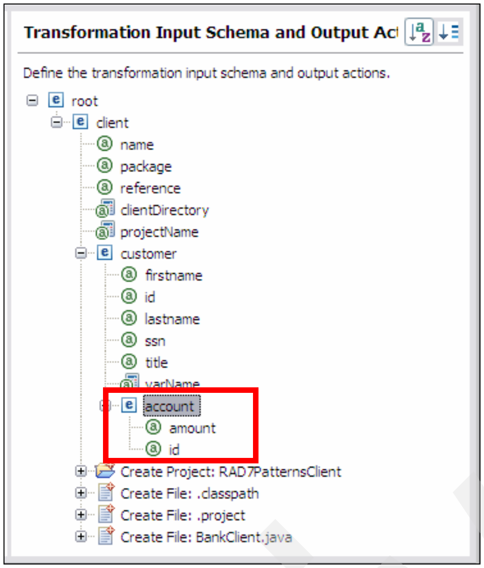
Figure 8-20 Replace all occurrences of customer1

- Replace the customer parameters by their corresponding model attribute (**xxx-xx-xxxx** by **$customer/@ssn**, **Mrs** by **$customer/@title**, **Julia** by **$customer/@firstName**, and **Johnson** by **$customer/@lastName**):

```
Customer <c:get select="$customer/@varName" /> = new Customer
   ("<c:get select="$customer/@ssn" />",
   "<c:get select="$customer/@title" />",
   "<c:get select="$customer/@firstname" />",
   "<c:get select="$customer/@lastname" />");
```

Using an account template

- As we did for customer, we have to create a new type for account. Because an account belongs to a given customer, we create this new type under **customer** and we call it **account**. It contains two attributes, **id** and **amount** (Figure 8-21).



Define the transformation input schema and output actions.

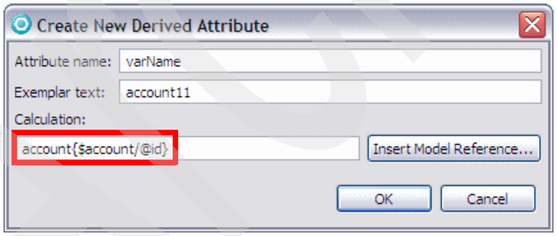
- root
  - client
    - name
    - package
    - reference
    - clientDirectory
    - projectName
    - customer
      - firstname
      - id
      - lastname
      - ssn
      - title
      - varName
      - account
        - amount
        - id
    - Create Project: RAD7PatternsClient
    - Create File: .classpath
    - Create File: .project
    - Create File: BankClient.java

Do not forget to run **Update Project**

Figure 8-21 Adding the account type

- Replace all occurrences of **account11** by a new **account** derived attribute called **varName**, with the calculation shown in Figure 8-22.

> **Note:** Do not forget to select **account** before clicking **New** in the **Find/Replace With Model Reference** dialog.



Attribute name: varName

Exemplar text: account11

Calculation:

account{$account/@id}

Use **Replace All** (see Figure 8-20 on page 342)

Figure 8-22 New account varName derived attribute

- Replace the New Account parameters by their corresponding attributes (**11** by **$account/@id**, **10000.00D** by **$account/@amount**):

```
Account <c:get select="$account/@varName" /> = new Account
    ("<c:get select="$account/@id" />",
        new BigDecimal(<c:get select="$account/@amount" />));
```

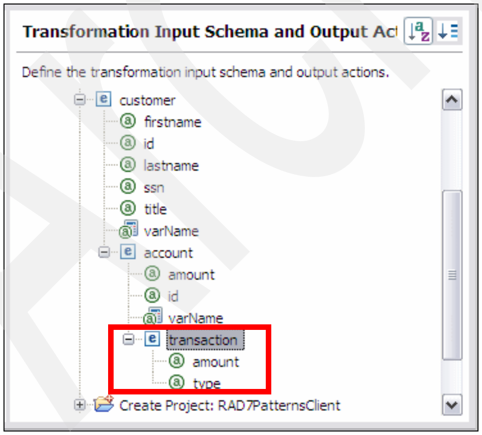
- The **account12** lines of code are similar to the **account11** lines we just replaced; they just create a second account. To reproduce that (and allow us to create more than two accounts), we delete the **account12** creation lines and embed the account lines by a **<c:iterate>** tag, creating as many blocks of lines as there are account elements declared in the input file:

```
<c:iterate select="$customer/account" var="account">
   Account <c:get select="$account/@varName" /> = new Account
      ("<c:get select="$account/@id" />",
         new BigDecimal(<c:get select="$account/@amount" />));
   bank.openAccountForCustomer(<c:get select="$customer/@varName" />,
                            <c:get select="$account/@varName" />);
   System.out.println("Account "
         + <c:get select="$account/@varName" />.getAccountNumber()
         + " has been successfully opened for "
         + <c:get select="$customer/@varName" /> + ".\n");
</c:iterate>
```

> **Note:** We also updated the output line before **</c:iterate>** to replace it with only one account reference.

Using a transaction template

- A transaction is always linked to only one account, so we create a new type called **transaction**, under the **account** type. Because there can be debit and credit transactions, we add a **type** attribute to the **account** type. We also add an **amount** attribute to contain the amount the transaction applies to. Figure 8-23 shows the resulting model.



Transformation Input Schema and Output Act

Define the transformation input schema and output actions.

Do not forget to run
Update Project

Figure 8-23 Adding the transaction type

- Replace the transaction amounts by the **$transaction/@amount** attribute. Because we want to perform more than one transaction, we also have to add a **<c:iterate>** tag on the **transaction** type. As there could be debit or credit transactions, we need to perform a test on the **type** attribute to insert the correct line of code. To do so, we use the **<c:choose>, <c:when>**, and **<c:otherwise>** tags. Example 8-1 shows the resulting template.

> **Notes:**
>
> - Because we iterate to create different amounts, the declaration of the amount variable (`BigDecimal amount;`) must be before the first **<c:iterate>** tag.
> - As the debit code is applied to **account12**, the easiest way to reuse the lines of code is to replace **account12** by **$account/@varName,** so it could be applied to any account we create.
> - The account iteration closing tag (**</c:iterate>**) has been moved after the transaction iteration closing tag because a transaction is a subtype of the **account** type.

Example 8-1 Template code containing the account and transaction parametrization

```
BigDecimal amount = 0; // top of method
<c:iterate select="$customer/account" var="account">
   Account <c:get select="$account/@varName" /> = new Account
          ("<c:get select="$account/@id" />",
             new BigDecimal(<c:get select="$account/@amount" />));
   bank.openAccountForCustomer(<c:get select="$customer/@varName" />,
                                <c:get select="$account/@varName" />);
   System.out.println("Account "
          + <c:get select="$account/@varName" />.getAccountNumber()
          + " has been successfully opened for "
          + <c:get select="$customer/@varName" /> + ".\n");
   System.out.println("System is listing all account information of "
          + <c:get select="$customer/@varName" /> + "...");
   System.out.println(bank.getAccountsForCustomer
             (<c:get select="$customer/@varName" />.getSsn()));

   <c:iterate select="$account/transaction" var="transaction">
      amount = new BigDecimal(<c:get select="$transaction/@amount" />);
      <c:choose select="$transaction/@type">
         <c:when test="'Credit'">
            System.out.println("\nSystem is going to make credit of $"
               + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
               + " to account " + <c:get select="$account/@varName" />
                                      .getAccountNumber() + "...");
            bank.deposit(<c:get select="$account/@varName" />
                                .getAccountNumber(), amount);
            System.out.println("Account "
               + <c:get select="$account/@varName" />.getAccountNumber()
               + " has sucessfully credited by $"
               + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN) + ".\n");
         </c:when>
         <c:when test="'Debit'">
            System.out.println("System is going to make debit of $"
               + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
               + " to account " + <c:get select="$account/@varName" /
                                             .getAccountNumber() + "...");
            bank.withdraw(<c:get select="$account/@varName" />
                                       .getAccountNumber(), amount);
            System.out.println("Account "
               + <c:get select="$account/@varName" />.getAccountNumber()
               + " has sucessfully debited by $"
               + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN) + ".\n");
         </c:when>
      </c:choose>
   </c:iterate>
</c:iterate>
```

- We said previously that eventually we would like to create more than one customer. To do so, we create a **<c:iterate>** tag. We also remove the remaining transactions and the account close command, as we do not expect our clients program to close any accounts. Example 8-2 shows the final **BankClient.java.jet** with all the tags.

> **Note:** The final **BankClient.java.jet** is available in the **RAD7PatternsClient** project in the **assets** directory.

Example 8-2 Final version of BankClient.java.jet

```
/*
 * File Name:  BankClient.java
 */
package <c:get select="$client/@package" />;

import itso.rad7.bank.exception.ITSOBankException;
import itso.rad7.bank.ifc.Bank;
import itso.rad7.bank.impl.ITSOBank;
import itso.rad7.bank.model.Account;
import itso.rad7.bank.model.Customer;

import java.math.BigDecimal;

public class BankClient {
   public static void main(String[] args) {
```

```
try {
    Bank bank = ITSOBank.getBank();
    BigDecimal amount = null;
<c:iterate select="$client/customer" var="customer">
    System.out.println("System is going to add new customer...");
    Customer <c:get select="$customer/@varName" /> =
        new Customer("<c:get select="$customer/@ssn" />",
                        "<c:get select="$customer/@title" />",
                        "<c:get select="$customer/@firstname" />",
                        "<c:get select="$customer/@lastname" />");
    bank.addCustomer(<c:get select="$customer/@varName" />);
    System.out.println(<c:get select="$customer/@varName" />
                            + " has been successfully added.\n");
    System.out.println
            ("System is going to open two new accounts for the customer "
                    + <c:get select="$customer/@varName" /> + "...");
  <c:iterate select="$customer/account" var="account">
    Account <c:get select="$account/@varName" /> =
        new Account("<c:get select="$account/@id" />",
                new BigDecimal(<c:get select="$account/@amount" />));
    bank.openAccountForCustomer(<c:get select="$customer/@varName" />,
                                        <c:get select="$account/@varName" />);
    System.out.println("Account "
            + <c:get select="$account/@varName" />.getAccountNumber()
            + " has been successfully opened for "
            + <c:get select="$customer/@varName" /> + ".\n");
    System.out.println("System is listing all account information of "
            + <c:get select="$customer/@varName" /> + "...");
    System.out.println(bank.getAccountsForCustomer
                            (<c:get select="$customer/@varName" />.getSsn()));
    <c:iterate select="$account/transaction" var="transaction">
        amount = new BigDecimal(<c:get select="$transaction/@amount" />);
        <c:choose select="$transaction/@type">
            <c:when test="'Credit'">
                System.out.println("\nSystem is going to make credit of $"
                    + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
                    + " to account " + <c:get select="$account/@varName" />
                                            .getAccountNumber() + "...");
                bank.deposit(<c:get select="$account/@varName" />
                                            .getAccountNumber(), amount);
                System.out.println("Account "
                + <c:get select="$account/@varName" />.getAccountNumber()
                + " has sucessfully credited by $"
                + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN) + ".\n");
            </c:when>
            <c:when test="'Debit'">
                System.out.println("System is going to make debit of $"
                    + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN)
                    + " to account " + <c:get select="$account/@varName" />
```

```
                                             .getAccountNumber() + "...");
                   bank.withdraw(<c:get select="$account/@varName" />
                                          .getAccountNumber(), amount);
                   System.out.println("Account "
                   + <c:get select="$account/@varName" />.getAccountNumber()
                   + " has sucessfully debited by $"
                   + amount.setScale(2, BigDecimal.ROUND_HALF_EVEN) + ".\n");
                </c:when>
             </c:choose>
          </c:iterate>
        </c:iterate>
          System.out.println("System is listing all account information of "
                + <c:get select="$customer/@varName" /> + "...");
          System.out.println(bank.getAccountsForCustomer
                   (<c:get select="$customer/@varName" />.getSsn()));
      </c:iterate>
          System.out.println("\nSystem is listing all customers ...");
          System.out.println(bank.getCustomers() + "\n");
          java.util.Iterator iter = bank.getCustomers().values().iterator();
          while (iter.hasNext()) {
             Customer customer = (Customer) iter.next();
             System.out.println("Customer: " + customer + "\n"
                + bank.getAccountsForCustomer(customer.getSsn()) + "\n");
          }
      } catch (ITSOBankException e) {
          e.printStackTrace();
      }
   }
}
```

Run **Update Project** before running the sample to make the model modifications available for use in the JET templates. Save all the files.

Applying the pattern

To apply the pattern, we create an XML input file for the JET transformation:

- Let us start by creating a **sample.xml** that creates the same test client we used as exemplar. Open the **sample.xml** file that is contained in the **RAD7PatternsTransform** project. Modify it to match what is in Example 8-3.

> **Note:** A version of **sample.xml** is also available into the **RAD7PatternsClient** project in the **assets** directory.

Example 8-3 Sample.xml to run the transformation

```
<root>
   <client name="RAD7Patterns2" package="itso.rad7.bank.client"
         reference="/RAD7Patterns">

      <customer ssn="xxx-xx-xxxx" lastname="Julia"
               firstname="Johnson" id="1" title="Mrs">
         <account id="11" amount="10000.00D">
            <transaction type="Credit" amount="2500.00D"></transaction>
         </account>
         <account id="12" amount="11234.23D">
            <transaction type="Debit" amount="1234.23D"></transaction>
            <transaction type="Credit" amount="5000.00D"></transaction>
         </account>
      </customer>
   </client>
</root>
```

- Right-click **sample.xml** and select **Run As** → **Input for Jet Transformation**.
- Explore the newly created **RAD7Patterns2Client** project (Figure 8-24):
  - A package `itso.rad7.bank.client` is created.
  - The client program `BankClient.java` is created.
  - The classpath has been set so that there are no errors.

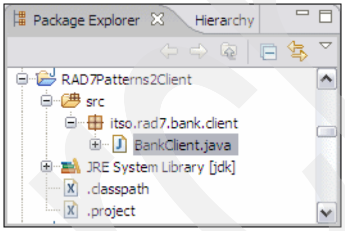


Figure 8-24 Client generated by transformation

Running the generated client

Open the generated client and study the source code.

Right-click **BankClient.java** → **Run As** → **Java Application** to verify that it produces a similar output as the examplar.

> **Note:** You can verify that both accounts have been created, that all transactions were performed, and that the balance is the same between the generated project and the exemplar project.

Running another transformation

This time we are running a different XML file, **sample2.xml**, that is in the **RAD7PatternsClient** project in the **assets** directory.

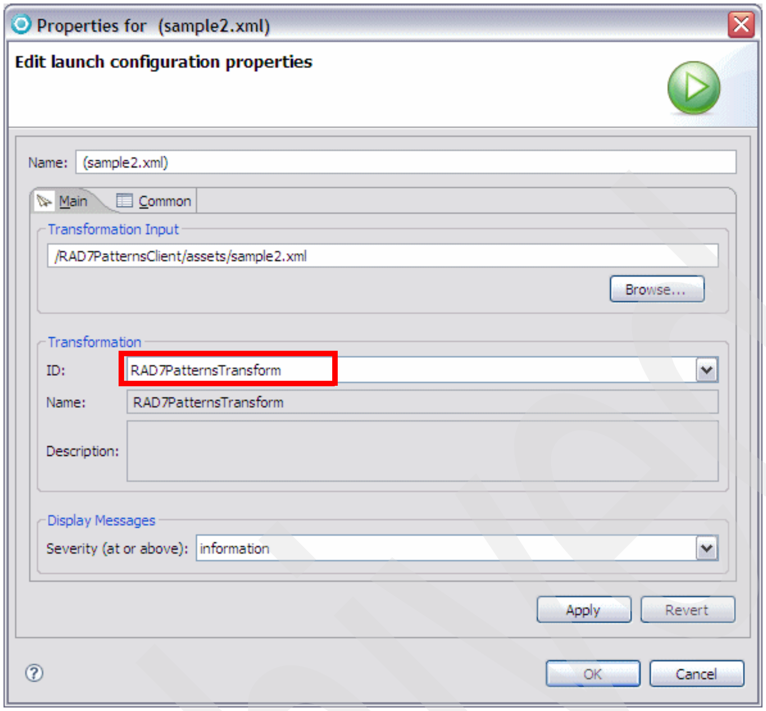
- Open **sample2.xml** and explore the content. As you can see in Example 8-4, we have two customers (Jane Doe and John Doe). Jane Doe has two accounts and we perform a debit and a credit transactions on her first account. John Doe has only one account and we do not perform any transactions. We generate the client into the same `RAD7Patterns2Client` project, but a new package (`itso.rad7.bank2.client`).

Example 8-4 Sample2.xml to run a transformation

```
<root>
   <client name="RAD7Patterns2" package="itso.rad7.bank2.client"
      reference="/RAD7Patterns">

      <customer ssn="999-99-9999" lastname="Doe"
               firstname="Jane" id="1" title="Mrs">
         <account id="11" amount="10000.00">
            <transaction type="Debit" amount="2399.99"></transaction>
            <transaction type="Credit" amount="2399.99"></transaction>
         </account>
         <account id="12" amount="11234.23">
         </account>
      </customer>
      <customer ssn="888-88-8888" lastname="Doe" firstname="John"
               id="2" title="Mr">
         <account id="21" amount="10000.00"></account>
      </customer>
   </client>
</root>
```

- To run the transform, right-click **sample2.xml** and select **Run As** → **Input for Jet Transformation**. Because the containing project is not a JET transformation project, the **Edit launch configuration properties** dialog is displayed. In this dialog, select **RAD7PatternsTransform** as transformation id from the drop-down list (Figure 8-25).

  > **Note:** After doing this initial configuration, the JET transformation can be rerun just by right-clicking **sample2.xml** and selecting **Run As** → **Input for Jet Transformation**.



Edit launch configuration properties

Name: (sample2.xml)

Transformation Input: /RAD7PatternsClient/assets/sample2.xml

Transformation
ID: RAD7PatternsTransform
Name: RAD7PatternsTransform
Description:

Display Messages
Severity (at or above): information

Figure 8-25 Creating the run configuration

- Explore the newly created **itso.rad7.bank2.client.BankClient.java** and study the generated source code. You can run the client to verify that it runs.

Facade pattern

In this section we describe a second example transformation, which generates the facade (interface and implementation) for the sample banking application.

Importing the facade example

Import the project interchange file for the facade example, by selecting **File** → **Import** → **Other** → **Project Interchange**. Click **Next**. Locate the example in:

```
c:\7501code\patterns\RAD7PatternsFacade.zip
```

Select both projects, `RAD7PatternsFacade` and `RAD7PatternsFacadeTransform`, and click **Finish**.

Study the two projects:

- RAD7PatternsFacade is the examplar project. We will generate the interface, itso.rad7.bank.ifc.Bank, and the implementation class, itso.rad7.bank.impl.ITSOBank.
- RAD7PatternsFacadeTransform is the JET Transformation project.

Facade transformation

Open the **transform.tma** file in RAD7PatternsFacadeTransform. The transformation defines:

- Attributes: name (of the interface), package (base package), systemProjName (underlying project with the model)
- Derived attributes: facadeDirectory (folders from package), facadeVarname (variable for the interface), ifcPackage (package of interface), implClass (implementation class name), implPackage (package of implementation), and projectName (generated project)
- Elements: entity (to define model objects), and operation (to define the methods of the interface)
- Creates: Project, .classpath, .project, Bank.java (interface), ITSOBank.java (implementation)

The template files (templates\facade) include:

- classpath.jet: .classpath template
- project.jet: .project template
- Bank.java.jet: Interface template
- ITSOBank.java.jet: Implementation template

Running the transformation examples

We provide two XML files to run the transformation and generate an interface and an implementation class:

- **BankSample.xml**: This example generates a **BankFacadeJava** project, with the Bank interface and the ITSOBank implementation class, matching the original classes in the RAD7Patterns projects (or RAD7Java project).
- **LibrarySample.xml**: This example generates a **LibraryFacadeJava** project, with a Library interface and an ITSOLibrary implementation class. This example demonstrates that the transformation can generate any interface and a matching implementation class. The library example is based on Book and Borrower model classes. Note that we do not provide the underlying project with the model classes (RAD7LibraryJava).

More information

The following URLs provide further information for the topics covered in this chapter:

- You can find several articles on pattern transformations on the IBM developerWorks Web site (search for JET pattern transformation):

  `http://www.ibm.com/developerworks`

- More information on using Exemplar Authoring and JET transformations can be found at IBM developerWorks in the article *Create powerful custom tools quickly with Rational Software Architect Version 7.0*:

  `http://www.ibm.com/developerworks/rational/library/07/0109_peterson/`

- For more information on patterns based development, visit the Patterns Solution area at IBM developerWorks:

  `http://www.ibm.com/developerworks/rational/products/patternsolutions/index.html?S_TACT=105AGX15&S_CMP=LP`

- More information on the XPath functions can be found at:

  `http://www.w3.org/TR/xpath`

9

Develop database applications

In an enterprise environment, applications that use databases are very common. In this chapter we explore technologies that are used in developing Java database applications. Our focus is on highlighting the database tooling provided with IBM Rational Application Developer V7.0.

This chapter is organized into the following sections:

- JDBC overview
- Setting up the sample database
- Connecting to databases
- Creating SQL statements
- Developing a Java stored procedure
- Developing SQLJ applications
- Data modeling

Introduction

Rational Application Developer provides rich features to make it easier to work with tables, views, and filters; create and work with SQL statements; create and work with database routines (such as stored procedures and user-defined functions), and create and work with SQLJ files. You can also create, modify, and generate data models. Depending on your goals, you might have to take certain steps to set up your work environment.

Depending on your goals, this chapter is written for three types of users:

- If you want to **access** databases and discover information about them, you can use the database explorer to create a connection to those databases. After you have set up connection information for a database, you can connect, refresh a connection, and browse the objects that are contained in the database.
- If you want to **develop** database related activities such as SQL queries and stored procedures, you have to create a data development project. The data development project stores your routines and other data development objects. Application developer also provides tooling to assist you to develop SQLJ applications, and offers a DB beans package to access database information without directly using the JDBC interface.
- If you want to **design** your database model, you have to create a data design project to store your objects. The modeling tool assists you to build a data model, analyze the model, perform the impact analysis, and so forth.

All examples in this chapter are demonstrated against the open source embedded Derby database server. The embedded version of Derby is bundled inside Rational Application Developer, so its availability is guaranteed. These examples can be easily applied to DB2 databases.

JDBC overview

Java DataBase Connectivity (JDBC), like Open DataBase Connectivity (ODBC), is based on the X/Open SQL call-level interface specifications; but unlike ODBC, JDBC does not rely on various C features that do not fit well with the Java language. Using JDBC, you can make dynamic calls to databases from a Java application or Java applet.

JDBC is vendor neutral and provides access to a wide range of relational databases, as well as other tabular sources of data. It can even be used to get data from flat files or spreadsheets.

JDBC is especially well suited for use in Web applications. Using the JDBC API you can connect to databases using standard network connections.

In JDBC 1.x the only way of establishing a database connection was by using the *DriverManager* interface. This was expensive in terms of performance because a connection was created each time you had to access the database from your program, thereby incurring a substantial processing overhead.

Data source

In the JDBC 2.x Standard API, you can use a data source to access the database. By using data source objects you have access to a pool of connections through the data source. Using connection pooling gives you the following advantages:

- Improves performance—Creating connections is expensive; a data source object creates a pool of connections as soon as it is instantiated.
- Simplifies resource allocation—Resources are only allocated from the data source objects, and not at arbitrary places in the code.

Data source objects work as follows:

- When a servlet or other client wants to use a connection, it looks up a data source object by name from a Java Naming and Directory Interface (JNDI) server.
- The servlet or client asks the data source object for a connection.
- If the data source object has no more connections, it can ask the database manager for more connections (as long as it has not exceeded the maximum number of connections).
- When the client has finished with the connection, the client releases the connection.
- The data source object then returns the connection to the available pool.

Setting up the sample database

We provide two implementations of the `ITSOBANK` database, Derby and DB2. Follow the instructions in “Setting up the ITSOBANK database” on page 1312 to set up the database.

The `ITSOBANK` database has four tables: `CUSTOMER`, `ACCOUNT`, `ACCOUNTS_CUSTOMERS`, and `TRANSACTIONS`.

An account can have multiple transactions and the ACCOUNTS_ID becomes the foreign key in the TRANSACTIONS table and is related to the primary key of the ACCOUNT table.

There is a many-to-many association between customers and accounts. ACCOUNTS_CUSTOMERS is the junction table to turn this many-to-many relationship into a one-to-many relationship between CUSTOMER and ACCOUNTS_CUSTOMERS and a one-to-many relationship between ACCOUNT and ACCOUNTS_CUSTOMERS.

Connecting to databases

Application Developer enables you to create a connection to the following databases:

- Cloudscape 5.1
- DB2 UDB v7.2, v8.1, 8.2, 9.1
- DB2 UDB iSeries V5R2, V5R3, V5R4
- DB2 UDB zSeries® V7, V8 New Function Mode and Compatibility Mode, V9 New Function Mode and Compatibility Mode
- Derby 10.0, 10.1
- Informix 9.2, 9.3, 9.4, 10.0
- MySQL 4.0, 4.1
- Oracle 8, 9, 10
- SQL Server 2000, 2005
- Sybase 12x, 15

Creating a database connection

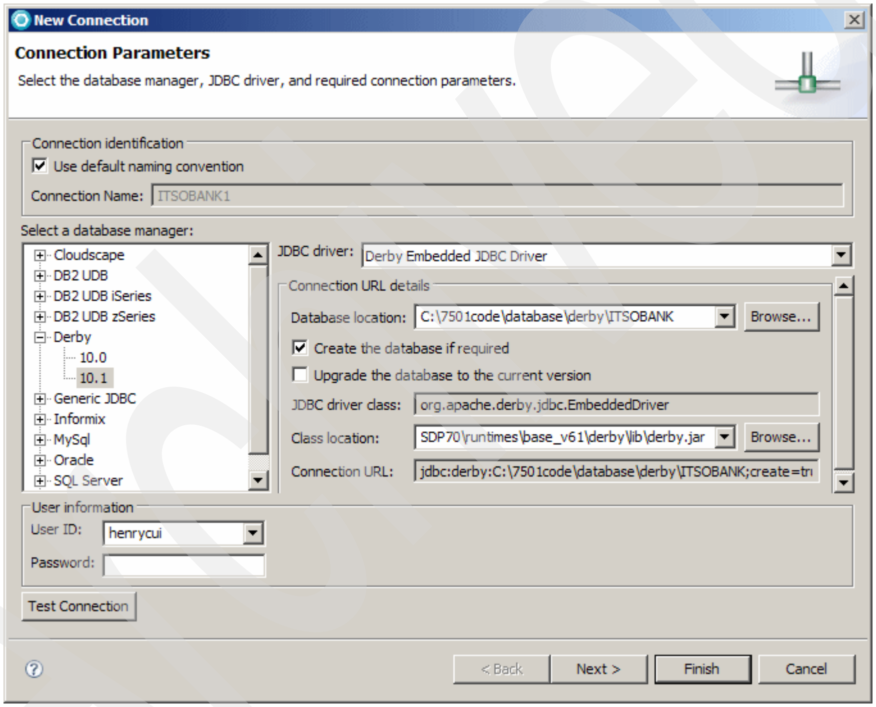
To connect to the Derby **ITSOBANK** database using the New Database Connection wizard, do these steps:

- Open the Data perspective by selecting **Window** → **Open Perspective** → **Other**. In the Open Perspective dialog, select **Data** and click **OK**. The data perspective opens.
- Locate the Database Explorer view, typically at the bottom left in the Data perspective.
- In the Database Explorer right-click **Connections** and select **New Connection**.
- In the New Connection wizard, do these steps (Figure 9-1):

- Select a database manager: Expand **Derby** and select **10.1**. This selects the Derby Embedded JDBC Driver.
- For Database location type **C:\7501code\database\derby\ITSOBANK**.
- For Class location click **Browse** to locate of the embedded Derby JDBC drive class:

```
<RAD_HOME>\runtimes\base_v61\derby\lib\derby.jar
C:\IBM\SDP70\runtimes\base_v61\derby\lib\derbyjar <=== example
```

- User ID: The Derby database does not require authentication, so you can enter any value.
- Click **Next**.



New Connection

Connection Parameters

Select the database manager, JDBC driver, and required connection parameters.

Connection identification

☑ Use default naming convention

Connection Name: ITSOBANK1

Select a database manager:

JDBC driver: Derby Embedded JDBC Driver

Connection URL details

Database location: C:\7501code\database\derby\ITSOBANK

☑ Create the database if required

☐ Upgrade the database to the current version

JDBC driver class: org.apache.derby.jdbc.EmbeddedDriver

Class location: SDP70\runtimes\base_v61\derby\lib\derby.jar

Connection URL: jdbc:derby:C:\7501code\database\derby\ITSOBANK;create=tru

User information

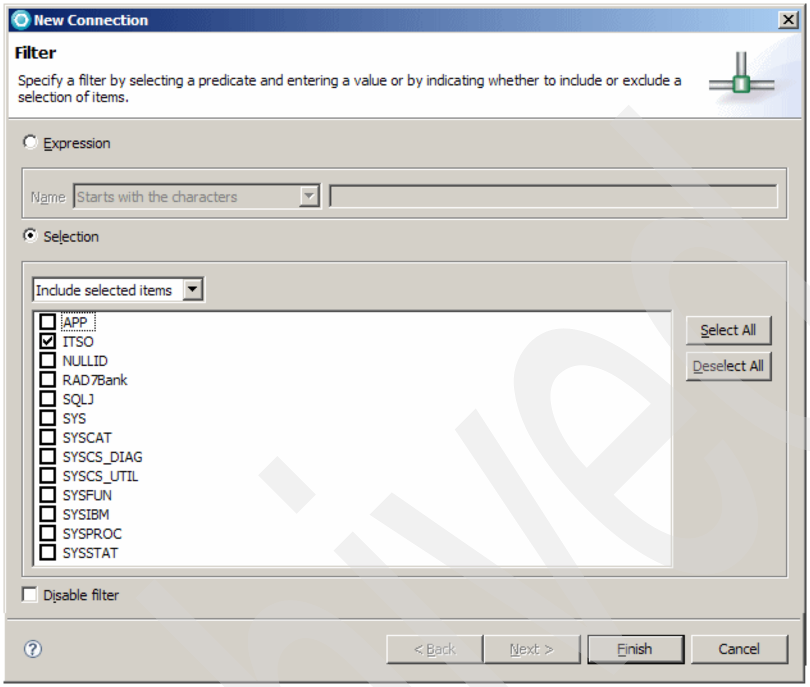
User ID: henrycui

Password:

Figure 9-1 New Connection: Connection Parameters

- You can use filters to exclude data objects (such as tables, schemas, stored procedures, and user-defined functions) from the view. Only the data objects that match the filter condition are shown. We only want to see the objects in schema `ITSO` (Figure 9-2):
  - Clear **Disable filter**.
  - Select **Selection**.
  - Select **Include selected items**.

– From the schema list, select **ITSO**.
– Click **Finish**.



New Connection

Filter

Specify a filter by selecting a predicate and entering a value or by indicating whether to include or exclude a selection of items.

○ Expression

Name: Starts with the characters

● Selection

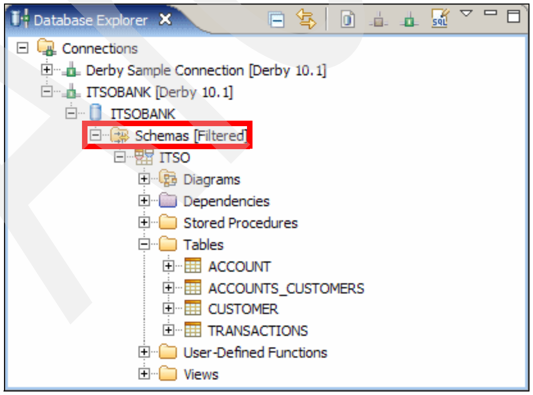
Include selected items

- □ APP
- ☑ ITSO
- □ NULLID
- □ RAD7Bank
- □ SQLJ
- □ SYS
- □ SYSCAT
- □ SYSCS_DIAG
- □ SYSCS_UTIL
- □ SYSFUN
- □ SYSIBM
- □ SYSPROC
- □ SYSSTAT

□ Disable filter

Figure 9-2 New Connection: Filter

- The connection is displayed in the Database Explorer. Expand **ITSOBANK [Derby 10.1]** → **ITSOBANK**. The Schemas folder is marked as **[Filtered]**. There is only one schema (**ITSO**) listed, the others are filtered (Figure 9-3).



Database Explorer

```
Connections
  Derby Sample Connection [Derby 10.1]
  ITSOBANK [Derby 10.1]
    ITSOBANK
      Schemas [Filtered]
        ITSO
          Diagrams
          Dependencies
          Stored Procedures
          Tables
            ACCOUNT
            ACCOUNTS_CUSTOMERS
            CUSTOMER
            TRANSACTIONS
          User-Defined Functions
          Views
```

Figure 9-3 Connection with schema and tables in Database Explorer

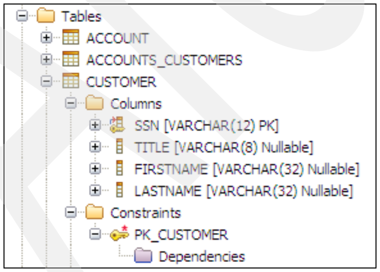
- The filter framework allows you to filter out the tables at a more granular level. Suppose we only want to see tables that start with the letter A. Expand the schema **ITSO**, right-click **Tables** and select **Filter**. In the Filter dialog, do these steps:
  - Clear **Disable filter**.
  - Select **Expression**.
  - In the Name section, select **Starts with the characters** and enter **A**.
  - Click **Finish**.
- Now you can only see two tables in the Database Explorer: `ACCOUNT` and `ACCOUNTS_CUSTOMERS`.
- We have to use the four tables in later sections. To disable the filter, right-click **Tables [Filtered]** and select **Filters,** select **Disable filter** and click **Finish**.

Browsing a database with the Database Explorer

The Database Explorer view operates much like any similar, graphical directory browsing program. The Database Explorer view has at its highest level of display, database connection resources. Here you can create and manage database connections, browse data objects in a connection, modify data objects a nd more.

Exploring the Derby database `ITSOBANK`:

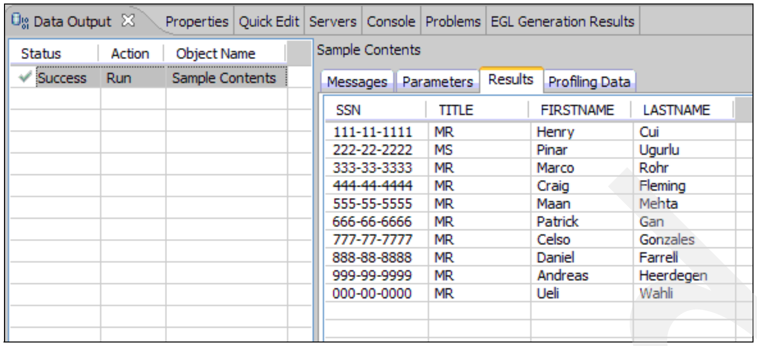
- Expand **ITSOBANK [Derby 10.1]** → **ITSOBANK** → **Schemas** → **ITSO** → **Tables** → **CUSTOMER** (Figure 9-4).
  - Expand **Columns**. All the columns in table `CUSTOMER` are listed. SSN is marked as primary key.
  - Expand **Constraints**. `PK_CUSTOMER` is listed as the primary key constraint.



```
Tables
  ACCOUNT
  ACCOUNTS_CUSTOMERS
  CUSTOMER
    Columns
      SSN [VARCHAR(12) PK]
      TITLE [VARCHAR(8) Nullable]
      FIRSTNAME [VARCHAR(32) Nullable]
      LASTNAME [VARCHAR(32) Nullable]
    Constraints
      PK_CUSTOMER
        Dependencies
```

Figure 9-4 Customer table with columns and constraints

- In the Database Explorer view right-click the **Customer** table and select **Data** → **Sample Contents**. The action opens the Data Output view (Figure 9-5).



Status	Action	Object Name
✓ Success	Run	Sample Contents

SSN	TITLE	FIRSTNAME	LASTNAME
111-11-1111	MR	Henry	Cui
222-22-2222	MS	Pinar	Ugurlu
333-33-3333	MR	Marco	Rohr
444-44-4444	MR	Craig	Fleming
555-55-5555	MR	Maan	Mehta
666-66-6666	MR	Patrick	Gan
777-77-7777	MR	Celso	Gonzales
888-88-8888	MR	Daniel	Farrell
999-99-9999	MR	Andreas	Heerdegen
000-00-0000	MR	Ueli	Wahli

Figure 9-5 Sample contents of `Customer` table via Data Output view.

Edit, extract, and load

The context menu produced by a **right-click** on the `Customer` table has many more options, such as:

- **Data** → **Edit** allows you to directly affect the contents of the target table in a spreadsheet like interface.
- **Data** → **Extract** allows you to extract data into a file using a delimiter (comma, semicolon, space, tab, vertical bar):

```
"111-11-1111","MR","Henry","Cui"
"222-22-2222","MS","Pinar","Ugurlu"
......
```

- **Data** → **Load** allows you to load data from a file, such as produced by extract.
- **Data** → **Extract as XML** allows you to generate an XML file from the database table:

```
<?xml version="1.0" encoding="UTF-8"?>
<SQLResult>
    <CUSTOMER>
        <SSN>111-11-1111</SSN>
        <TITLE>MR</TITLE>
        <FIRSTNAME>Henry</FIRSTNAME>
        <LASTNAME>Cui</LASTNAME>
    </CUSTOMER>
   ......
</SQLResult>
```

- **Data** → **Load from XML** allows you to update a database table from an XML file, such as produced by extract as XML.

Creating SQL statements

You can create an SQL statement by using the SQL builder in the Data perspective. The SQL builder supports creating SELECT, INSERT, UPDATE, DELETE, FULLSELECT, and WITH (DB2 only) statements.

In this section, we create and run an SQL query to retrieve a customer name based on the social security number, and the total amount of money involved in each transaction type (credit, debit). The SQL select statement includes table aliases, table joins, a query condition, a column alias, a sort type, a database function expression and a grouping clause.

Creating a data development project

Before you create routines or other database development objects, you must create a data development project to store your objects. A data development project is linked to one database connection in the Database Explorer.

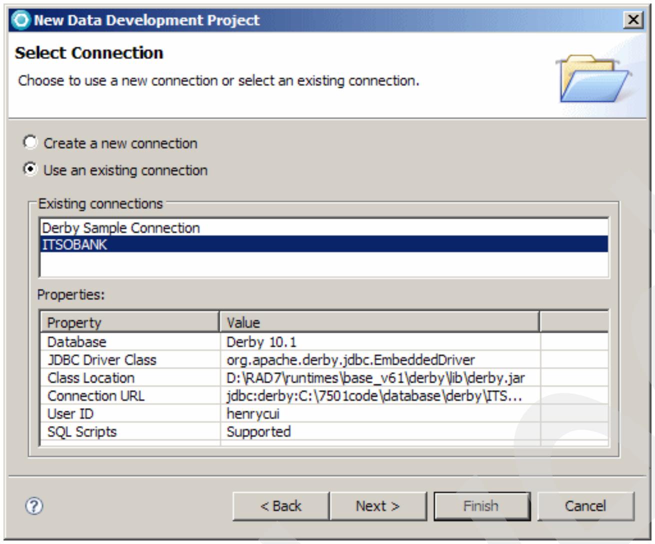
A data development project is used to store routines and queries. You can store and develop the following types of objects in a database development project:

- SQL scripts
- DB2 and Derby stored procedures
- DB2 user-defined functions

You can also test, debug, export, and deploy these objects from a data development project. The wizards that are available in a data development project use the connection information that is specified for the project to help you develop objects that are targeted for that specific database.

To create a data development project:

- In the Data perspective, select **File** → **New** → **Project**, then **Data** → **Data Development Project (**alternatively right-click in the Data Project Explorer and select **New** → **Data Development Project**). Click **Next**.
- In the Data Development Project page of the wizard, type **RAD7DataDevelopment** for the project name. Select **Omit current schema in generated SQL Statements**. Click **Next**.
- In the Select Connection page, because we already created a connection in the last section, select **Use an existing connection** and then select **ITSOBANK** from the existing connections list (Figure 9-6).



Select Connection

Choose to use a new connection or select an existing connection.

○ Create a new connection

● Use an existing connection

Existing connections

- Derby Sample Connection
- ITSOBANK

Properties:

Property	Value
Database	Derby 10.1
JDBC Driver Class	org.apache.derby.jdbc.EmbeddedDriver
Class Location	D:\RAD7\runtimes\base_v61\derby\lib\derby.jar
Connection URL	jdbc:derby:C:\7501code\database\derby\ITS...
User ID	henrycui
SQL Scripts	Supported

Figure 9-6 Data Development Project: Select Connection

- Click **Finish**. The data development project is displayed in the Data Project Explorer view.

Populating the transactions table

Before we build the SQL query, we want to populate the `TRANSACTIONS` table with more data (only one customer has transactions already). To load the records into this table, do these steps:

- In the Data Project Explorer right-click the project **RAD7DataDevelopment** and select **Import** → **General** → **File System**, and click **Next**.
- Click **Browse** and locate the `C:\7501code\database\samples` directory. Select **LoadTransaction.sql** and click **Finish**.
- The **LoadTransaction.sql** appears in the **SQL Scripts** folder. Right-click **LoadTransaction.sql** and select **Run SQL** from the context menu.
- The results are displayed in the Data Output view. For the INSERT SQL statements, the status should be Success.

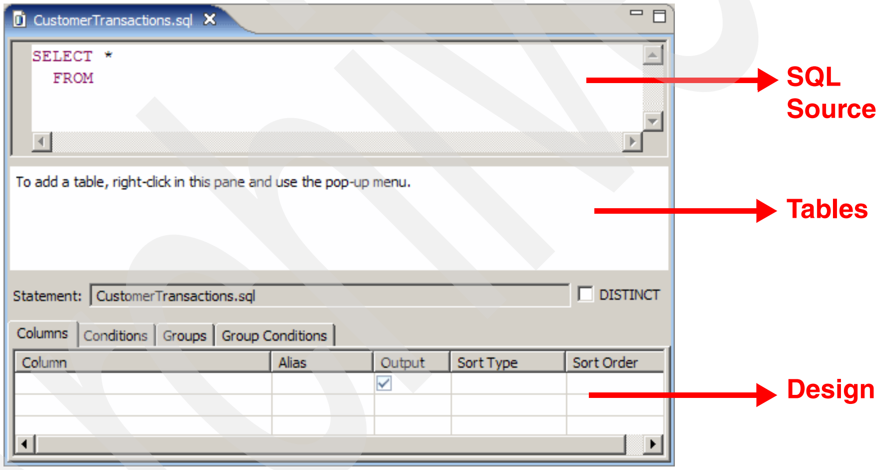
> **Note:** Stop the WebSphere Application Server v6.1 if it is running and has accessed the `ITSOBANK` database for other chapters.

Creating a select statement

We want to retrieve a customer name and the total amount of money involved in each transaction type (credit, debit).

To create the select statement:

- In the Data Project Explorer view right-click the **SQL Scripts** folder in the **RAD7DataDevelopment** project and select **New** → **SQL Statement**.
- In the Specify a Project page of the wizard, `RAD7DataDevelopment` is preselected. Click **Next**.
- In the Statement Type page of the wizard, enter **CustomerTransactions** as the Statement name. For Statement template select **SELECT** and for Edit using select **SQL Builder**.
- Click **Finish** and the `SELECT` statement template is created and opens in the SQL builder (Figure 9-7).



```
SELECT *
  FROM
```

To add a table, right-click in this pane and use the pop-up menu.

Statement: CustomerTransactions.sql ☐ DISTINCT

Column	Alias	Output	Sort Type	Sort Order
		☑		

Figure 9-7 SQL Builder

Using the SQL Builder

The SQL Builder has three main sections:

- **SQL source pane**—The SQL Source pane contains the source code of the SQL statement. You can type the SQL statement in this pane, or use the features that are provided by the SQL builder to build the statement. Content assist is available as you type and through the pop-up menu in the SQL Source pane. This pane also provides content tips through the pop-up menu. A content tip shows a simple example for the type of statement that you are creating.

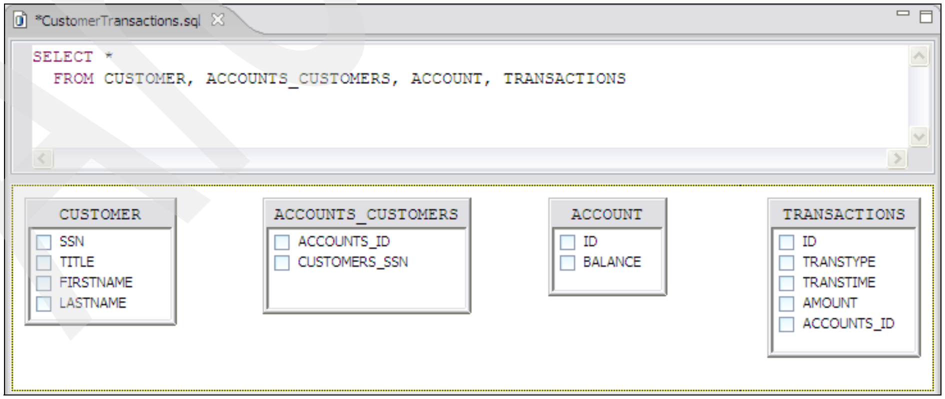
- **Tables pane**—The Tables pane provides a graphical representation of the table references that are used in the statement. In this pane, you can add or remove a table, give a table an alias, and select or exclude columns from the table. When you build a SELECT statement, you can also define joins between tables in this pane.
- **Design pane**—The options in the Design pane vary, depending on the type of statement that you are creating. When more than one set of options is available, the options appear as notebook pages. For example, for a SELECT statement, some of the options include selecting columns, creating conditions, creating groups, and creating group conditions.

Adding tables to the statement

You will have to add four tables to the SELECT statement for the `CustomerTransactions` query, because this query traverses from `CUSTOMER` through to `ACCOUNT` and `TRANSACTIONS`.

To add tables to the statement:

- In the Database Explorer expand **ITSOBANK [Derby 10.1]** → **ITSOBANK** → **Schemas** → **ITSO** → **Tables**. You can see the four tables.
- Drag the **`CUSTOMER`** table from the Database Explorer view to the Tables pane in the SQL builder. The `CUSTOMER` table is shown in the Tables pane, and the source code in the SQL Source pane shows the addition of the `CUSTOMER` table in the SELECT statement.
- Follow the same procedure to drag the `ACCOUNTS_CUSTOMERS`, `ACCOUNT`, and `TRANSACTIONS` tables from the Database Explorer view to the Tables pane in the SQL builder (Figure 9-8). Select a table and drag the sides to adjust the size of the displayed rectangle.



```
SELECT *
  FROM CUSTOMER, ACCOUNTS_CUSTOMERS, ACCOUNT, TRANSACTIONS
```

Figure 9-8 Create select statement: Tables

Adding table aliases

We will create an alias for each of the tables in the SELECT statement. An alias is an indirect method of referencing a table so that an SQL statement can be independent of the qualified name of that table. If the table name changes, only the alias definition must be changed.

The aliases for the ACCOUNT, CUSTOMER, TRANSACTIONS, and ACCOUNTS_CUSTOMERS will be A, C, T and AC, respectively.

- In the Tables pane, right-click the header in the **ACCOUNT** table and select **Update Alias**.
- In the Change Table Alias dialog, type **A** as the alias for the table and click **OK**. In the Tables pane, the alias is shown in the header for the ACCOUNT table. In the SQL Source pane, the alias is represented by ACCOUNT AS A.
- Repeat steps to add aliases for the **CUSTOMER** (**C**), **TRANSACTIONS** (**T**) and **ACCOUNTS_CUSTOMERS** (**AC**) tables.

Selecting columns for the result set

We add the following columns to the result set by selecting the columns in the tables pane:

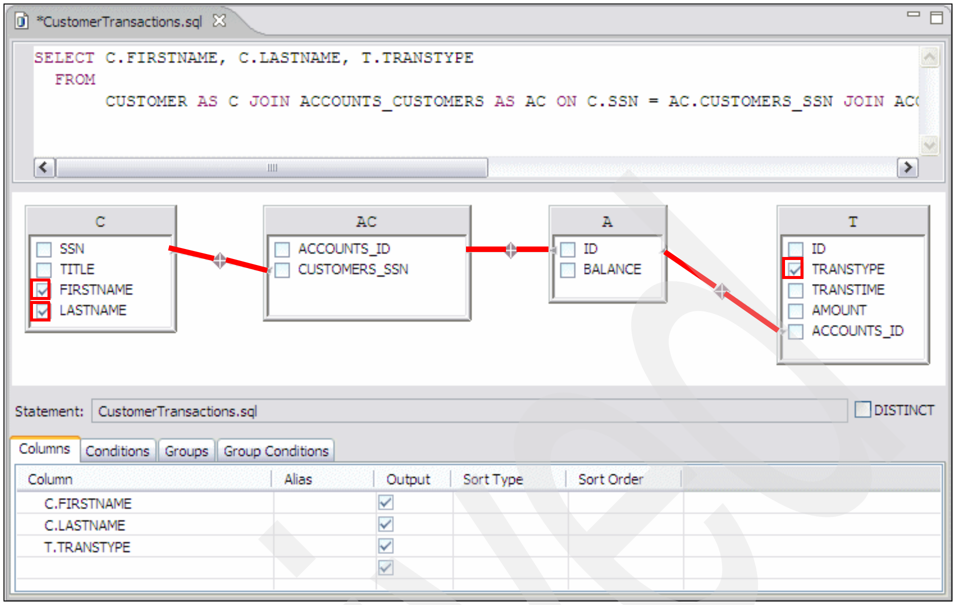
- Select FIRSTNAME and LASTNAME columns in the C (CUSTOMER) table.
- Select TRANSTYPE column in the T (TRANSACTIONS) table.

Joining tables

A join operation lets you retrieve data from two or more tables based on matching column values. Three joins are needed for this query:

- Drag the cursor from column **SSN** column in the **C** (CUSTOMER) table to column **CUSTOMERS_SSN** in **AC** (ACCOUNTS_CUSTOMERS) table.
- Drag the cursor from **ACCOUNTS_ID** in the **AC** (ACCOUNTS_CUSTOMERS) table to **ID** in the **A** (ACCOUNT) table.
- Drag the cursor from **ID** in the **A** (ACCOUNT) table to **ACCOUNTS_ID** in the **T** (TRANSACTIONS) table.

Relationship lines are drawn between the selected columns (Figure 9-9).



*CustomerTransactions.sql

```
SELECT C.FIRSTNAME, C.LASTNAME, T.TRANSTYPE
  FROM
       CUSTOMER AS C JOIN ACCOUNTS_CUSTOMERS AS AC ON C.SSN = AC.CUSTOMERS_SSN JOIN ACC
```

Statement: CustomerTransactions.sql ☐ DISTINCT

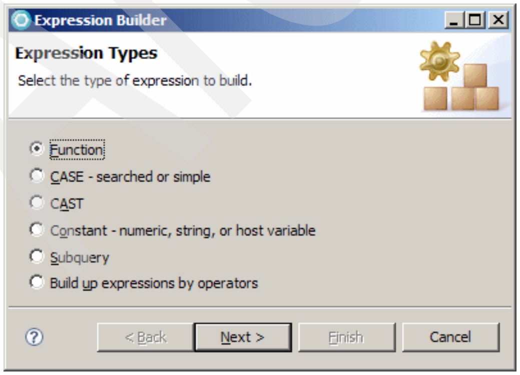
Column	Alias	Output	Sort Type	Sort Order
C.FIRSTNAME		☑		
C.LASTNAME		☑		
T.TRANSTYPE		☑		
		☑		

Figure 9-9 Create select statement: Columns and table joins

Add a function expression to the result set

The fourth column for the query result set will be the result of a column expression. We add together the total amount of each transaction type. This can be calculated using the expression builder wizard:

- In the Columns page in the Design pane double-click the fourth cell in the Column column (the first empty cell), select **Build Expression** at the end of the list, and press **Enter**.
- The Expression Builder wizard opens (Figure 9-10).



Expression Builder

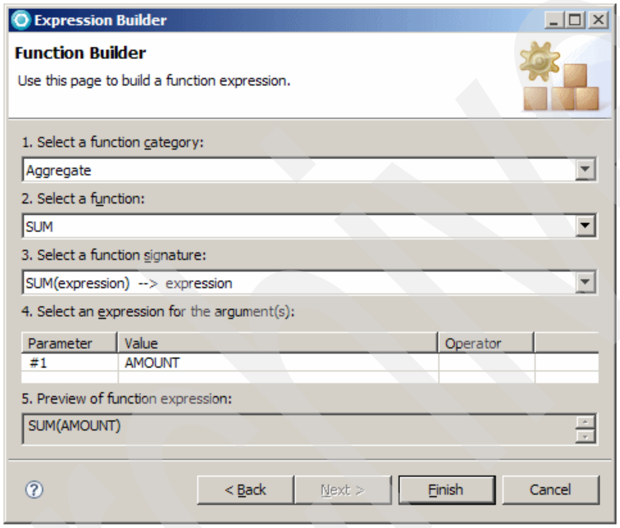
Expression Types

Select the type of expression to build.

● Function
○ CASE - searched or simple
○ CAST
○ Constant - numeric, string, or host variable
○ Subquery
○ Build up expressions by operators

Figure 9-10 Create select statement: Expression Builder (1)

- Select **Function** and click **Next**.
- In the Function Builder page, select the following options (Figure 9-11):
  - For Select a function category, select **Aggregate**.
  - For Select a function, select **SUM**.
  - For Select a function signature, select **SUM(expression) → expression**.
  - In the Value column of the argument table, double-click the cell, select **T.AMOUNT** in the list, and press **Enter**.
  - The syntax of the function expression is **SUM(AMOUNT)**.
  - Click **Finish**.



Function Builder

Use this page to build a function expression.

1. Select a function category: Aggregate
2. Select a function: SUM
3. Select a function signature: SUM(expression) --> expression
4. Select an expression for the argument(s):

Parameter	Value	Operator
#1	AMOUNT	

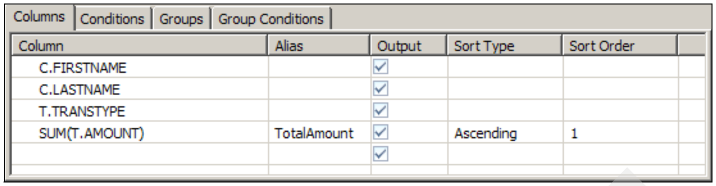
5. Preview of function expression: SUM(AMOUNT)

Figure 9-11 Create select statement: Expression Builder (2)

Adding a column alias and sort type

We add a column alias for the function column expression and sort the results:

- Click the **Columns** tab in the Design pane.
- Click the cell in the Alias column next to the `SUM(T.AMOUNT)` column expression, type **TotalAmount**, and press **Enter**.
- Click the cell in the Sort Type column next to the `TotalAmount` alias, select **Ascending**, and press **Enter**.
- The Columns page is seen in Figure 9-12.



Column	Alias	Output	Sort Type	Sort Order
C.FIRSTNAME		☑		
C.LASTNAME		☑		
T.TRANSTYPE		☑		
SUM(T.AMOUNT)	TotalAmount	☑	Ascending	1
		☑		

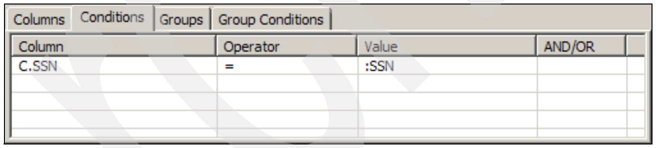
Figure 9-12 Create select statement: Columns

Creating a query condition

The query needs a query condition so that the query extracts only result rows with a given customer social security number. We add conditions to the query by using the Conditions page in the Design pane.

To create a query condition:

- In the design pane, select the **Conditions** tab.
- In the first row, double-click the cell in the **Column** column and then select **C.SSN** in the list.
- In the same row, double-click the cell in the **Operator** column and select the **=** operator.
- In that row, double-click the cell in the **Value** column and then enter **:SSN**. A colon followed by a variable name is the SQL syntax for a host variable that will be substituted with a value when you run the query.
- The Conditions page is shown in Figure 9-13.



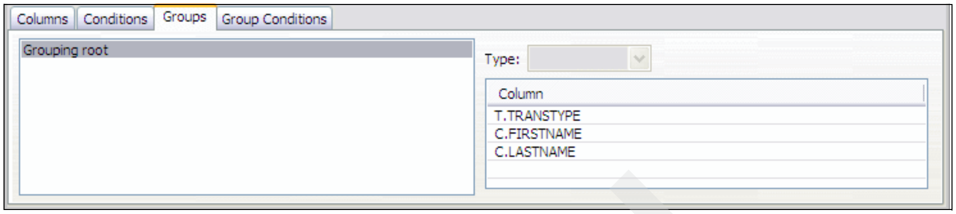
Column	Operator	Value	AND/OR
C.SSN	=	:SSN	

Figure 9-13 Create select statement: Conditions

Adding a GROUP BY clause

We group the query by the transaction type so that we have one sum of the amount for each type of transaction (credit, debit):

- In the Design pane, select the **Groups** tab.
- In the Column table, double-click the first row, select **T.TRANSTYPE** in the list, and then press **Enter**.
- Repeat this for `C.FIRSTNAME` and `C.LASTNAME`.
- The Groups page is shown in Figure 9-14.



Column
T.TRANSTYPE
C.FIRSTNAME
C.LASTNAME

Figure 9-14 Create select statement: Groups

The query is now complete. **Save** the select statement. The SQL statement is listed in Example 9-1.

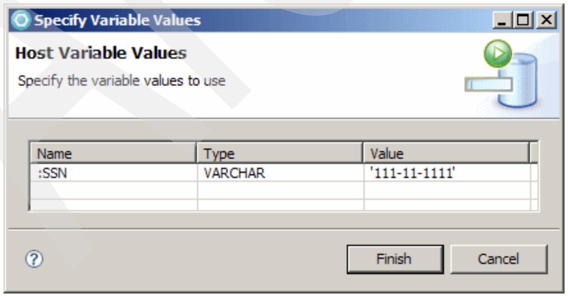
Example 9-1 CustomerTransactions.sql

```
SELECT C.FIRSTNAME, C.LASTNAME, T.TRANSTYPE, SUM(T.AMOUNT) AS "TotalAmount"
  FROM
       CUSTOMER AS C JOIN ACCOUNTS_CUSTOMERS AS AC ON C.SSN = AC.CUSTOMERS_SSN
       JOIN ACCOUNT AS A ON AC.ACCOUNTS_ID = A.ID
       JOIN TRANSACTIONS AS T ON A.ID = T.ACCOUNTS_ID
  WHERE C.SSN = :SSN
  GROUP BY T.TRANSTYPE, C.FIRSTNAME, C.LASTNAME
  ORDER BY "TotalAmount" ASC
```

Running the SQL query

To run the select statement:

- In the Data Project Explorer, right-click **CustomerTransactions.sql** and select **Run SQL**. The Specify Variable Values window opens.
- Double-click the first **Value** cell, enter the string **111-11-1111** in the cell and press **Enter** (Figure 9-15).



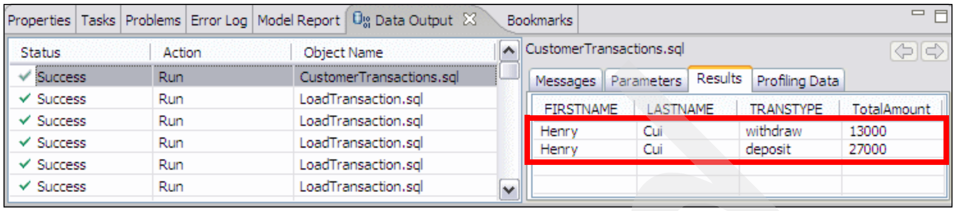
Host Variable Values

Specify the variable values to use

Name	Type	Value
:SSN	VARCHAR	'111-11-1111'

Figure 9-15 Host Variable Values prompt

- Click **Finish**. The result is seen in Figure 9-16. You can see the total amount of money for each transaction type is calculated and the results are ordered by `TotalAmount` in ascending order.



Status	Action	Object Name
✓ Success	Run	CustomerTransactions.sql
✓ Success	Run	LoadTransaction.sql
✓ Success	Run	LoadTransaction.sql
✓ Success	Run	LoadTransaction.sql
✓ Success	Run	LoadTransaction.sql
✓ Success	Run	LoadTransaction.sql

FIRSTNAME	LASTNAME	TRANSTYPE	TotalAmount
Henry	Cui	withdraw	13000
Henry	Cui	deposit	27000

Figure 9-16 Query results

Developing a Java stored procedure

A stored procedure is a block of procedural constructs and embedded SQL statements that are stored in a database and can be called by name. Stored procedures can help improve application performance and reduce database access traffic. All database access must go across the network, which, in some cases, can result in poor performance. For each SQL statement, a database manager application must initiate a separate communication with database.

To improve application performance, you can create stored procedures that run on a database server. A client application can then simply call the stored procedures to obtain results of the SQL statements that are contained in the procedure. Because the stored procedure runs the SQL statements on the server for you, database performance is improved.

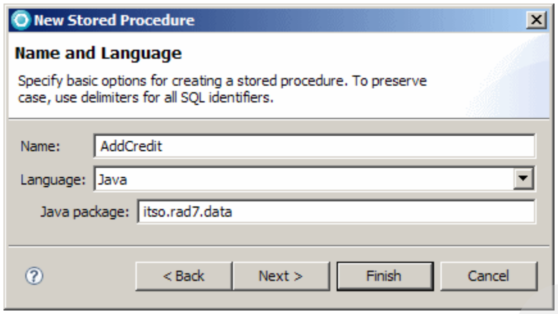
Stored procedures can be written as SQL procedures, or as C, COBOL, PL/I, or Java programs. In this section, we develop a Java stored procedure against the `ITSOBANK` Derby database to obtain the account information based on a partial customer last name. While doing so, we will give also give $100 credit to every account retrieved (this would be nice!).

Creating a Java stored procedure

To create a stored procedure using the Stored Procedure wizard, do these steps:

- In the Data Project Explorer view, expand the **RAD7DataDevelopment** project, right-click the **Stored Procedures** folder, and select **New** → **Stored Procedure**. The New Stored Procedure wizard opens.
- In the Specify a Project page, `RAD7DataDevelopment` is preselected. Click **Next**.

- In the Name and Language page, type **AddCredit** for the Name, select **Java** as the language, and type **itso.rad7.data** as the package name (Figure 9-17).



Name and Language

Specify basic options for creating a stored procedure. To preserve case, use delimiters for all SQL identifiers.

Name: AddCredit

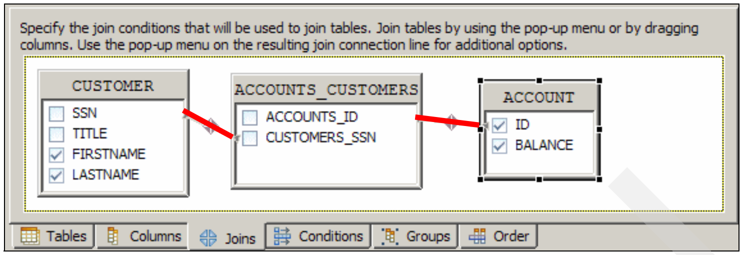
Language: Java

Java package: itso.rad7.data

Figure 9-17 Create stored procedure: Name and Language

Because we are developing a stored procedure against Derby database, **Java** is the only option for the **Language**. If you develop stored procedures against a DB2 database, you will see two options: **Java** and **SQL**. Click **Next**.

- In the SQL Statements page, click **Create SQL**. This action launches the New SQL Statement wizard that guides you through the creation of an SQL statement.
- In the first page of the New SQL Statement wizard, keep the defaults to create a **SELECT** statement using the wizard, and click **Next**.
- We go through several tabs to create the SQL statement:
  - In the **Tables** tab, in the Available Tables list, expand the `ITSO` schema, select **ITSO.CUSTOMER**, **ITSO.ACCOUNTS_CUSTOMERS**, and **ITSO.ACCOUNT**, and click > to move the three tables to the Selected Tables list.
  - Select the **Columns** tab, expand the `CUSTOMER` table and select **FIRSTNAME** and **LASTNAME**. Expand the `ACCOUNT` table and select **ID** and **BALANCE**. Click **>** to move the columns to the Selected Columns list.
  - Select the **Joins** tab, drag the cursor from **`SSN`** (`CUSTOMER`) to **`CUSTOMERS_SSN`** (`ACCOUNTS_CUSTOMERS`), and from **`ID`** (`ACCOUNT`) to **`ACCOUNTS_ID`** (`ACCOUNTS_CUSTOMERS`) to create two joins (Figure 9-18).



Specify the join conditions that will be used to join tables. Join tables by using the pop-up menu or by dragging columns. Use the pop-up menu on the resulting join connection line for additional options.

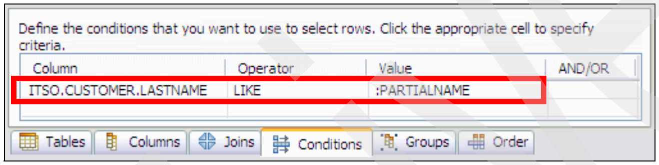
CUSTOMER
☐ SSN
☐ TITLE
☑ FIRSTNAME
☑ LASTNAME

ACCOUNTS_CUSTOMERS
☐ ACCOUNTS_ID
☐ CUSTOMERS_SSN

ACCOUNT
☑ ID
☑ BALANCE

Figure 9-18 Create stored procedure: Joins

– Select the **Conditions** tab. In the first row double-click the cell under Column and select **CUSTOMER.LASTNAME**. In the same row select **LIKE** as the Operator and type **:PARTIALNAME** as the Value (Figure 9-19).



Define the conditions that you want to use to select rows. Click the appropriate cell to specify criteria.

Column	Operator	Value	AND/OR
ITSO.CUSTOMER.LASTNAME	LIKE	:PARTIALNAME	

Figure 9-19 Create stored procedure: Conditions

– Click **Next**.

- In the Change the SQL Statement page, review the generated SQL statement and click **Finish** to close the New SQL Statement wizard.
- Back in the New Stored Procedure wizard, select **One** for the **Result set** and click **Next**.
- In the Parameters page, leave the settings as default and click **Next**.
- In the Deploy Options page of the wizard, clear **Deploy**. We will deploy the stored procedure in later steps. Click **Next**.
- In the Code Fragments page of the wizard, click **Next**.
- Review your selections on the Summary page of the wizard and click **Finish**. The generated file is opened and is shown in Example 9-2.

Example 9-2 AddCredit.java

```
package itso.rad7.data;

import java.sql.*;                   // JDBC classes

public class AddCredit
{
   public static void addCredit ( String PARTIALNAME,
                                  ResultSet[] rs1 ) throws SQLException,
Exception
   {
      // Get connection to the database
      Connection con = DriverManager.getConnection("jdbc:default:connection");
      PreparedStatement stmt = null;
      boolean bFlag;
      String sql;

      sql = "SELECT CUSTOMER.FIRSTNAME, CUSTOMER.LASTNAME, ACCOUNT.ID,
                      ACCOUNT.BALANCE"
          + "   FROM"
          + "        CUSTOMER JOIN ACCOUNTS_CUSTOMERS ON CUSTOMER.SSN =
                      ACCOUNTS_CUSTOMERS.CUSTOMERS_SSN JOIN ACCOUNT ON
                      ACCOUNTS_CUSTOMERS.ACCOUNTS_ID = ACCOUNT.ID"
          + "   WHERE CUSTOMER.LASTNAME LIKE  ?";
      stmt = con.prepareStatement( sql );
      stmt.setString( 1, PARTIALNAME );
      bFlag = stmt.execute();
      rs1[0] = stmt.getResultSet();
   }
}
```

- We will give $100 credit to the selected accounts. Add the following code **under** the **rs1[0] = stmt.getResultSet()** statement (Example 9-3):

Example 9-3 Snippet to give $100 credit to each account

```
      String sql2 = "UPDATE ITSO.ACCOUNT  SET BALANCE = (BALANCE + 100)"
         + " WHERE ID IN " +
      "(SELECT ITSO.ACCOUNT.ID FROM ITSO.ACCOUNT JOIN ITSO.ACCOUNTS_CUSTOMERS"
            + " ON ITSO.ACCOUNT.ID = ITSO.ACCOUNTS_CUSTOMERS.ACCOUNTS_ID"
            + " JOIN ITSO.CUSTOMER ON ITSO.ACCOUNTS_CUSTOMERS.CUSTOMERS_SSN ="
            + " ITSO.CUSTOMER.SSN"
         + " WHERE CUSTOMER.LASTNAME LIKE  ?)";
      stmt = con.prepareStatement(sql2);
      stmt.setString( 1, PARTIALNAME );
      stmt.executeUpdate();
```

- The modified `AddCredit.java` can also be found in:

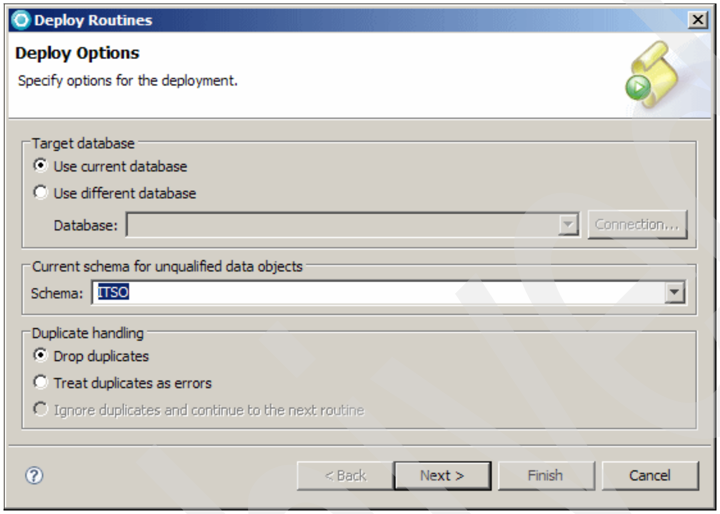
  `C:\7501code\database\samples`

Deploying a Java stored procedure

A stored procedure must be deployed to the database where it is stored in the catalog, ready for execution.

To deploy a Java stored procedure to a database:

- In the Data Project Explorer expand **RAD7DataDevelopment** → **Stored Procedures**, right-click **ADDCREDIT**, and select **Deploy**.
- The Deploy Routines wizard opens. Select **ITSO** as the Schema name and click **Finish** (Figure 9-20).


Deploy Routines

Deploy Options

Specify options for the deployment.

Target database

● Use current database

○ Use different database

Database:

Current schema for unqualified data objects

Schema: ITSO

Duplicate handling

● Drop duplicates

○ Treat duplicates as errors

○ Ignore duplicates and continue to the next routine

Figure 9-20 Deploy Routines

- You can see a success build status in the Data Output view.
- In the Data Explorer view you can see that **ADDCREDIT** has been added under ITSOBANK → Schemas → ITSO → Stored Procedures.

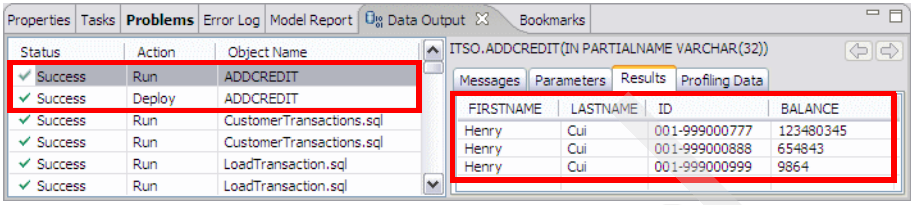
Running the stored procedure

Application Developer provides a test facility for testing the Java stored procedures.

To run the stored procedure:

- In the Data Project Explorer, right-click the stored procedure **ADDCREDIT**, and select **Run**.
- The Specify Parameter Values window opens. In the Value field type **C%** in the cell and press **Enter** (`LASTNAME LIKE 'C%'` retrieves only the customer with last name Cui).
- Click **OK**.

- The result is shown in Figure 9-21. You can see $100 has been added to the related accounts and the balances are updated.



Status	Action	Object Name
✓ Success	Run	ADDCREDIT
✓ Success	Deploy	ADDCREDIT
✓ Success	Run	CustomerTransactions.sql
✓ Success	Run	CustomerTransactions.sql
✓ Success	Run	LoadTransaction.sql
✓ Success	Run	LoadTransaction.sql

ITSO.ADDCREDIT(IN PARTIALNAME VARCHAR(32))

FIRSTNAME	LASTNAME	ID	BALANCE
Henry	Cui	001-999000777	123480345
Henry	Cui	001-999000888	654843
Henry	Cui	001-999000999	9864

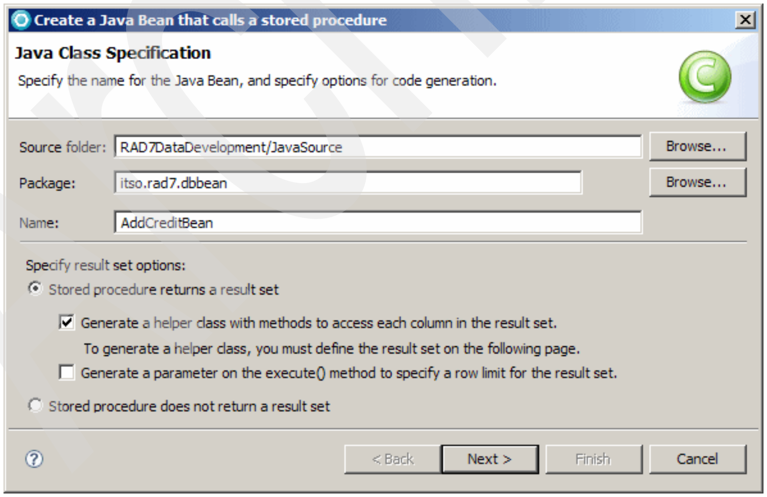
Figure 9-21 Stored procedure results

Creating a JavaBean to call a stored procedure

In this section, we describe how to use the **Create a Java Bean that calls a stored procedure** wizard to create a Java Bean that calls a specific stored procedure, and then run the Java stored procedure using the generated JavaBean.

To generate a JavaBean, do these steps:

- In the Data Project Explorer, right-click the **ADDCREDIT** stored procedure and select **Generate Java Bean**.
- In the Java Class Specification page of the wizard, enter data as shown in Figure 9-22.


Create a Java Bean that calls a stored procedure

Java Class Specification

Specify the name for the Java Bean, and specify options for code generation.

Source folder: RAD7DataDevelopment/JavaSource

Package: itso.rad7.dbbean

Name: AddCreditBean

Specify result set options:

● Stored procedure returns a result set

☑ Generate a helper class with methods to access each column in the result set.

To generate a helper class, you must define the result set on the following page.

☐ Generate a parameter on the execute() method to specify a row limit for the result set.

○ Stored procedure does not return a result set

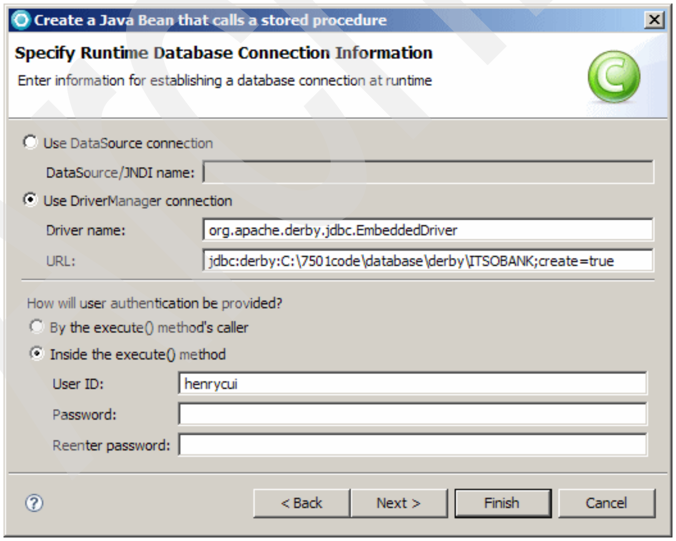
Figure 9-22 Java Class Specification

  - For the Source Folder, click **Browse** and select **RAD7DataDevelopment/JavaSource**.
  - For Package, enter **itso.rad7.dbbean**.
  - For name, enter **AddCreditBean**.
  - Select **Stored procedure returns a result set**.
  - Select **Generate a helper class with methods to access each column in the result set**. A helper class provides better access to each column of the result set.

- In the Define the result set page, select the **ITSO.CUSTOMER** table and **ITSO.ACCOUNT** table, because the stored procedure returns the customer first name and last name, and the account ID and balance.

  Click the Columns tab, expand the two tables and select the four columns (`FIRSTNAME`, `LASTNAME`, `ID`, `BALANCE`) and click **>**. Then click **Next**.

- In the Specify Runtime Database Connection Information page (Figure 9-23), do these steps:
  - Select **Use DriverManager connection**.
  - Select **Inside the execute () method**.
  - The Derby database does not require authentication. You can enter any user ID and an empty password.
  - Click **Next**.


Create a Java Bean that calls a stored procedure

Specify Runtime Database Connection Information

Enter information for establishing a database connection at runtime

○ Use DataSource connection

DataSource/JNDI name:

● Use DriverManager connection

Driver name: org.apache.derby.jdbc.EmbeddedDriver

URL: jdbc:derby:C:\7501code\database\derby\ITSOBANK;create=true

How will user authentication be provided?

○ By the execute() method's caller

● Inside the execute() method

User ID: henrycui

Password:

Reenter password:

Figure 9-23 Specify Runtime Database Connection Information

- In the Review the specification page, review the methods that will be generated (`execute`, `getDBProcedureCall`, `getRows`) and click **Finish**.
- Because the Data perspective only shows the database objects, switch to the Java (or Web) perspective to review the generated classes.

The following two classes are generated:

- `AddCreditBean`—Executes the stored procedure and provides a method to retrieve an array of `AddCreditBeanRow` objects.
- `AddCreditBeanRow`—Provides access to one row of the result set.

The **dbbeans.jar** is added to the project build path. DB beans provide applications with easy access to database information without directly using the JDBC interface. The DB beans classes are provided in the package `com.ibm.db.beans`. These beans use JDBC, and therefore they are able to access any database that has a JDBC driver that uses standard JDBC interfaces. The beans have capabilities that you would otherwise have to implement separately in each JDBC application, such as caching the rows of a result set in memory, and updating or deleting rows of a result set.

Important: Open `AddCreditBean.java` and locate the call to the stored procedure:

```
procCall.setCommand("{ CALL ADDCREDIT (:PARTIALNAME) }");
```

The schema name is missing. Change this line to:

```
procCall.setCommand("{ CALL ITSO.ADDCREDIT (:PARTIALNAME) }");
```

Save `AddCreditBean.java`.

Using the DB bean

The generated DB bean can be used in a Java application to execute the stored procedure and access the result set.

To use the DB Bean, do these steps:

- In the Java perspective expand **RAD7DataDevelopment** → **JavaSource**, right-click **itso.rad7.dbbean** and select **New** → **Class**.
- In the New Java Class page, provide the following information and click **Finish**:
  - Name: **AddCreditBeanTest**
  - Select **public static void main(String[] args)**

Copy/paste the following code to the `main` method of `AddCreditBeanTest` (Example 9-4). The `AddCreditBeanTest.java` code can be found in `C:\7501code\database\samples`.

Example 9-4 AddCreditBeanTest.java

```
public static void main(String[] args) {
   try {
      AddCreditBean bean = new AddCreditBean();
      bean.execute("C%");                  <===== try other values as well
      AddCreditBeanRow[] rows = bean.getRows();
      for (int i = 0; i < rows.length; i++) {
         System.out.println(rows[i].getCUSTOMER_FIRSTNAME() + " " +
                           rows[i].getCUSTOMER_LASTNAME());
         System.out.println(rows[i].getACCOUNT_ID() + " Balance: " +
                           rows[i].getACCOUNT_BALANCE());
         System.out.println("--------------------------------");
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
}
```

- We have to add the Derby JDBC driver library to the project build path:

  Right-click project **RAD7DataDevelopment** → **Properties** → **Libraries** tab → click **Add External JARs** → select **derby.jar**, which is in **<RAD_HOME>\runtimes\base_v61\derby\lib**. Click **OK**.

Running the DB bean

Derby accepts only one database connection at a time. Switch to the Data perspective. In the Database Explorer right-click the **ITSOBANK** connection and select **Disconnect**.

Switch back to the Java perspective. Right click **ADDCreditBeantest.java** and select **Run As** → **Java Application**. The result is displayed in the console. You can see another $100 credit to the accounts of Henry Cui:

```
Henry Cui
001-999000777 Balance: 123480445
--------------------------------
Henry Cui
001-999000888 Balance: 654943
--------------------------------
Henry Cui
001-999000999 Balance: 9964
--------------------------------
```

Developing SQLJ applications

SQLJ enables you to embed SQL statements into Java programs. SQLJ is an ANSI standard developed by a consortium of leading providers of database and application server software.

The SQLJ translator translates an SQLJ source file into a standard Java source file plus an SQLJ serialized profile that encapsulates information about static SQL in the SQLJ source. The translator converts SQLJ clauses to standard Java statements by replacing the embedded SQL statements with calls to the SQLJ runtime library. An SQLJ customization script binds the SQLJ profile to the database, producing one or more database packages. The Java file is compiled and run (with the packages) on the database. The SQLJ runtime environment consists of an SQLJ runtime library that is implemented in pure Java. The SQLJ runtime library calls the JDBC driver for the target database.

SQLJ provides better performance by using static SQL. SQLJ generally requires fewer lines of code than JDBC to perform the same tasks. The SQLJ translator checks the syntax of SQL statements during translation. SQLJ uses database connections to type-check static SQL code. With SQLJ, you can embed Java variables in SQL statements. SQLJ provides strong typing of query output and return parameters and allows type-checking on calls. SQLJ provides static package-level security with compile-time encapsulation of database authorization.

Using the SQLJ wizard shipped with Application Developer, you can do the following actions:

- Name an SQLJ file and specify its package and source folder.
- Specify advanced project properties, such as additional JAR files, to add to the project classpath, translation options, and whether to use long package names.
- Select an existing SQL SELECT statement, or construct and test a new one.
- Specify information for connecting to the database at run time.

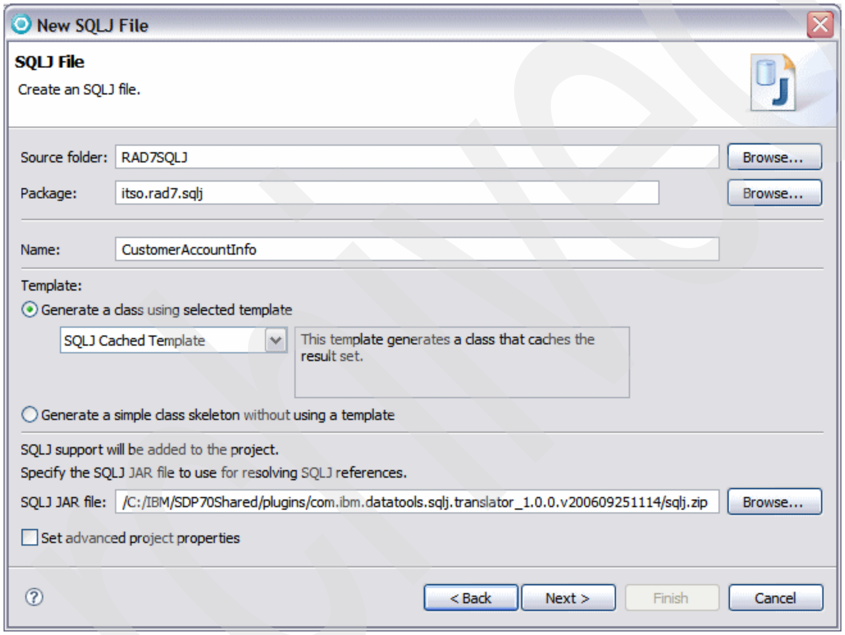
In this section, we will create an SQLJ application to retrieve the customer and the associated account information.

Creating SQLJ files

You can create SQLJ files by using the New SQLJ File wizard. The SQLJ support is automatically added to the project when you use this wizard.

We create a Java project named **RAD7SQLJ** and then create the SQLJ file in this project:

- In the Java perspective, select **File** → **New** → **Project** → **Java** → **Java Project** and click **Next**.
- Enter **RAD7SQLJ** as the Project name. Click **Finish**.
- Select **File** → **New** → **Other** → **Data** → **SQLJ Applications** → **SQLJ File** and click **Next**.
- In the SQLJ File page, enter **itso.rad7.sqlj** as the package name and **CustomerAccountInfo** as the file name and click **Next** (Figure 9-24).



SQLJ File

Create an SQLJ file.

Source folder: RAD7SQLJ

Package: itso.rad7.sqlj

Name: CustomerAccountInfo

Template:

● Generate a class using selected template

SQLJ Cached Template

This template generates a class that caches the result set.

○ Generate a simple class skeleton without using a template

SQLJ support will be added to the project.

Specify the SQLJ JAR file to use for resolving SQLJ references.

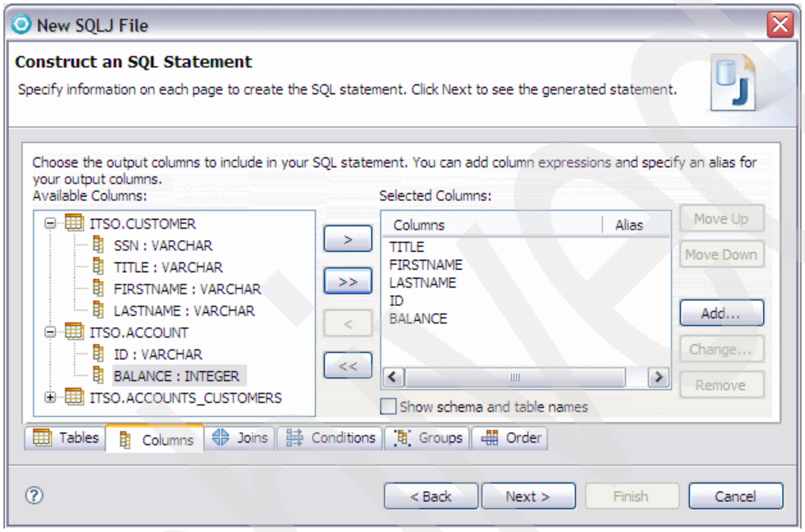
SQLJ JAR file: /C:/IBM/SDP70Shared/plugins/com.ibm.datatools.sqlj.translator_1.0.0.v200609251114/sqlj.zip

□ Set advanced project properties

Figure 9-24 New SQLJ File

- In the Select an Existing Statement Saved in Your Workspace page, click **Next**. We create a new SQL statement.
- In the Specify SQL Statement Information page, select **Be guided through creating an SQL statement**, click **Next**.
- In the Select Connection page, select the **ITSOBANK** connection that you created in the previous section. Click **Reconnect** to reconnect to the database if it is disconnected. Click **Next**.
- In the Construct an SQL Statement page, we go through several pages:

- In the Tables tab, for Available Tables, expand the **ITSO** schema, select the **CUSTOMER**, **ACCOUNT**, and **ACCOUNTS_CUSTOMERS** tables, and click **>** to move these three tables to the Selected Tables list.
- Select the **Columns** tab. In the Available columns list, select **TITLE**, **FIRSTNAME** and **LASTNAME** under the `CUSTOMER` table, and **ID** and **BALANCE** under the `ACCOUNT` table, and then click **>** to move these columns to the selected Columns list (Figure 9-25).



New SQLJ File

Construct an SQL Statement

Specify information on each page to create the SQL statement. Click Next to see the generated statement.

Choose the output columns to include in your SQL statement. You can add column expressions and specify an alias for your output columns.

Available Columns:

```
⊟ ITSO.CUSTOMER
    SSN : VARCHAR
    TITLE : VARCHAR
    FIRSTNAME : VARCHAR
    LASTNAME : VARCHAR
⊟ ITSO.ACCOUNT
    ID : VARCHAR
    BALANCE : INTEGER
⊞ ITSO.ACCOUNTS_CUSTOMERS
```

Selected Columns:

Columns	Alias
TITLE	
FIRSTNAME	
LASTNAME	
ID	
BALANCE	

□ Show schema and table names

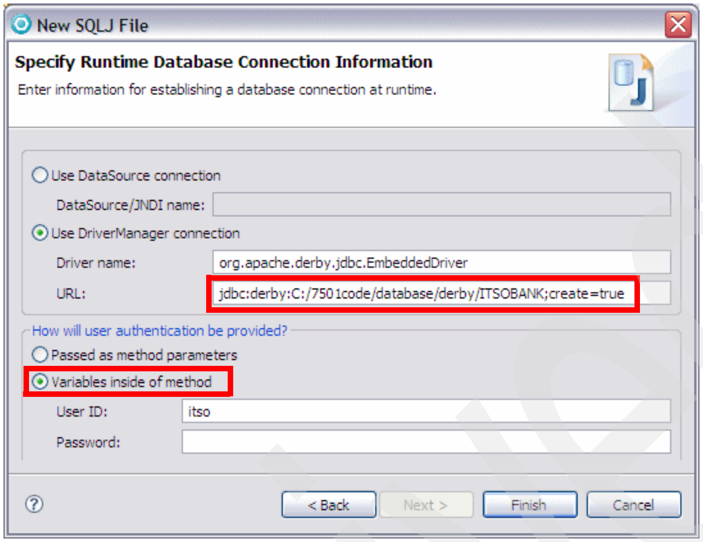
Figure 9-25 Select the output columns

- Select the **Joins** tab. Drag the cursor from `CUSTOMER.SSN` to `CUSTOMERS_SSN` and from `ACCOUNT.ID` to `ACCOUNTS_ID` (refer to Figure 9-18 on page 374).
- Select the **Conditions** tab. In the first row double-click the cell in the Column and select **ACCOUNT.BALANCE**. In the same row select **>=** as the Operator and type **:BALANCE** as the Value.
- Select the **Order** tab. Select **ACCOUNT.BALANCE** and click >. For Sort order select **DESC**. The results will be listed with the highest balance first.
- Click **Next**.

- In the Change the SQL Statement page, review the generated SQL statement and click **Next**.
- In the Specify Runtime Database Connection Information page, select **Use DriverManager Connection** (Figure 9-26). Note that the URL takes values previous dialogs. Single back slash (\) create errors when generated. Change the URL to use forward slashes:

```
jdbc:derby:C:/7501code/database/derby/ITSOBANK;create=true
```

Derby does not use authentication. Select **Variables inside of method** and enter any user ID and an empty password.



New SQLJ File

Specify Runtime Database Connection Information

Enter information for establishing a database connection at runtime.

○ Use DataSource connection

DataSource/JNDI name:

● Use DriverManager connection

Driver name: org.apache.derby.jdbc.EmbeddedDriver

URL: jdbc:derby:C:/7501code/database/derby/ITSOBANK;create=true

How will user authentication be provided?

○ Passed as method parameters

● Variables inside of method

User ID: itso

Password:

Figure 9-26 Specify Runtime Database Connection Information

- Click **Finish**. The SQLJ file is generated. If you see an error in the Problems view, then you probably did not change the URL to use forward slashes. Correct the SQLJ source file.

Examining the generated SQLJ file

Before testing the SQLJ program, let us examine the generated SQLJ file:

- The **establishConnection** method creates the database connection.
- The **execute** method executes the SQL query and stores the result set in a cache. The SQLJ statement is embedded in this method (Example 9-5):

Example 9-5 Embedded SQLJ

```
#sql [ctx] cursor1 = {SELECT ITSO.CUSTOMER.TITLE,
      ITSO.CUSTOMER.FIRSTNAME, ITSO.CUSTOMER.LASTNAME, ITSO.ACCOUNT.ID,
      ITSO.ACCOUNT.BALANCE FROM ITSO.ACCOUNT JOIN ITSO.CUSTOMER JOIN
      ITSO.ACCOUNTS_CUSTOMERS ON ITSO.CUSTOMER.SSN =
      ITSO.ACCOUNTS_CUSTOMERS.CUSTOMERS_SSN ON ITSO.ACCOUNT.ID =
      ITSO.ACCOUNTS_CUSTOMERS.ACCOUNTS_ID WHERE ITSO.ACCOUNT.BALANCE >=
      :BALANCE ORDER BY BALANCE DESC};
```

- The **next** method moves to the next row of the result set if one exists.

- The **close** method commits changes and closes the connection.
- The corresponding setter and getter methods for the table fields in the database are also generated. You can use the getter methods to retrieve the columns in a row.

Testing the SQLJ program

To create a test program to invoke the SQLJ program, do these steps:

- In the Package Explorer right-click the package **itso.rad7.sqlj** and select **New** → **Java Class**.
- Enter **TestSQLJ** as the class name. Select **public static void main(String[] args)** and click **Finish**.

  Copy/paste the following code to `TestSQLJ` (Example 9-6). The `TestSQLJ.java` code can be found in `C:\7501code\database\samples`.

Example 9-6 TestSQLJ.java

```
package itso.rad7.sqlj;

public class TestSQLJ {

   public static void main(String[] args) {
      try {
         CustomerAccountInfo info = new CustomerAccountInfo();
         info.execute(99999);         <==== minimum balance displayed
         while (info.next()) {
            System.out.println("Customer name: " + info.getCUSTOMER_TITLE()
                  + " " + info.getCUSTOMER_FIRSTNAME() + " "
                  + info.getCUSTOMER_LASTNAME());
            System.out.println("Account ID:   " + info.getACCOUNT_ID() +
                            " Balance:  " + info.getACCOUNT_BALANCE());
            System.out.println("--------------------------------------");
         }
         info.close();
      } catch (Exception e) {
         e.printStackTrace();
      }
   }
}
```

- We have to add the Derby JDBC driver library to the project build path:

  Right-click project **RAD7SQLJ** → **Properties** → **Libraries** tab → click **Add External JARs** → select **derby.jar**, which is in `<RAD_HOME>\runtimes\base_v61\derby\lib`. Click **OK**.

- Derby accepts only one database connection at a time. Switch to the Data perspective. In the Database Explorer right-click the **ITSOBANK** connection and select **Disconnect**.
- Switch back to the Java perspective. Right-click **TestSQLJ.java** → **Run As** → **Java Application**.
- The result is displayed in the console:

```
Retrieve some data from the database.
Customer name: MR Celso Gonzales
Account ID:   007-999000777 Balance:  250000000
---------------------------------------
Customer name: MR Craig Fleming
Account ID:   004-999000888 Balance:  145645646
---------------------------------------
Customer name: MR Henry Cui
Account ID:   001-999000777 Balance:  123480645
---------------------------------------
......
```

Note: If you get the following exception:

```
Failed to start database 'C:/7501code/database/derby/ITSOBANK'
```

This means that the Derby database is locked by another connection. Check the Database Explorer if the `ITSOBANK` connection is still active. Disconnect the connection, or restart the workbench.

Data modeling

Application Developer provides tools to create, modify, and generate DDL for data models. At any time when you are building a data model, you can analyze the model to verify that it is compliant with the defined constraints. If you make changes to the data model, Application Developer provides tooling to compare the changed data model with the original data model.You can also perform an impact analysis to determine how the changes might affect other objects.

A physical data model is a database-specific model that represents relational data objects (for example, tables, columns, primary keys, and foreign keys) and their relationships. A physical data model can be used to generate DDL statements which can then be deployed to a database server.

In the Workbench, you can create and modify data models by using the Data Project Explorer, the Properties view, or a diagram of the model. You can also analyze models and generate DDL.

In this section, we will create the physical model from template, create tables using the data diagram and deploy the physical model to the database. This section includes the following tasks:

- Creating a data design project
- Creating a physical data model
- Modeling with diagrams
- Generating DDL from physical data model and deploy

Creating a data design project

Before you create data models or other data design objects, you must create a data design project to store your objects.

A data design project is primarily used to store modeling objects. You can store the following types of objects in a data design project:

- Logical data models
- Physical data models
- Domain models
- Glossary models
- SQL scripts, including DDL scripts

To create a data design project:

- Open the Data perspective.
- Select **File** → **New** → **Data Design Project**. The New Data Design Project wizard opens.
- In the Project Name field type **RAD7DataDesign**, then click **Finish**.

The data design project is displayed in the Data Project Explorer view (Figure 9-27).

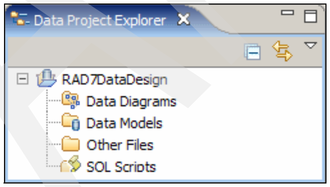


Figure 9-27 Data Project Explorer: Data Design project layout

Creating a physical data model

A physical data model is a database-specific model that represents relational data objects (for example, tables, columns, primary and foreign keys) and their relationships. A physical data model can be used to generate DDL statements which can then be deployed to a database server.

Using the data tooling, you can create a physical data model in several ways:

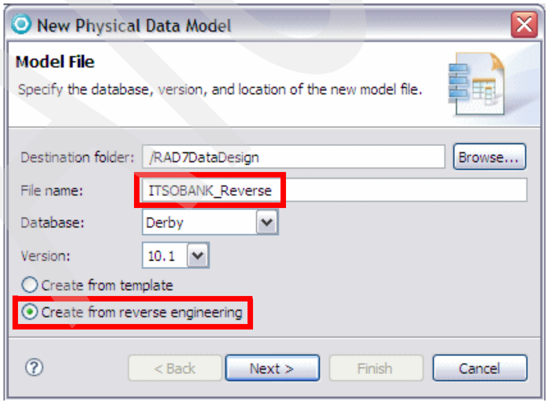
- Create a blank physical model by using a wizard.
- Create a physical model from a template by using a wizard.
- Reverse engineer a physical model from a database or a DDL file by using a wizard or by dragging data objects from the Database Explorer.
- Import a physical data model file from the file system.

In this section, we show you two ways to create the physical data model. First we create a physical data model by reversing the model from an existing database `ITSOBANK`. Then we create a new physical data model from a template and deploy this new model to the database.

Creating a physical data model using reverse engineering

To create a physical data model by reverse engineering an existing database schema, do these steps:

- Right-click **RAD7DataDesign** and select **New** → **Physical Data Model**. The New Physical Data Model wizard opens (Figure 9-28). Enter the following:
  – File name: **ITSOBANK_Reverse**
  – Database and Version: **Derby, 10.1**
  – select **Create from reverse engineering**


New Physical Data Model

Model File

Specify the database, version, and location of the new model file.

Destination folder: /RAD7DataDesign

File name: ITSOBANK_Reverse

Database: Derby

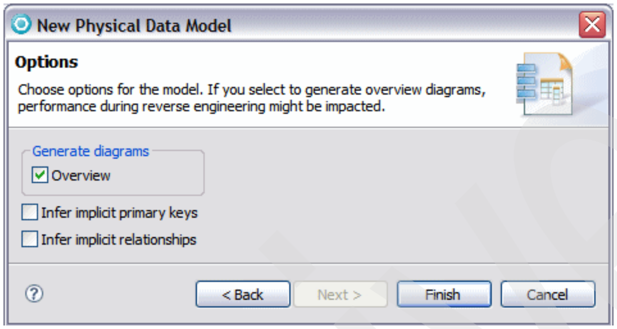
Version: 10.1

○ Create from template

● Create from reverse engineering

Figure 9-28 New Physical Data Model: Create from reverse engineering

- In the Select connection page, select **Use an existing connection** and select **ITSOBANK** from the existing connections list. Click **Next**.
- In the Schema page, select **ITSO** and click **Next**.
- In the Database Elements page, select **Tables** and click **Next**.
- In the Options page, select **Overview** in the Generate diagrams section (Figure 9-29).
- Click **Finish**.



New Physical Data Model

Options

Choose options for the model. If you select to generate overview diagrams, performance during reverse engineering might be impacted.

Generate diagrams

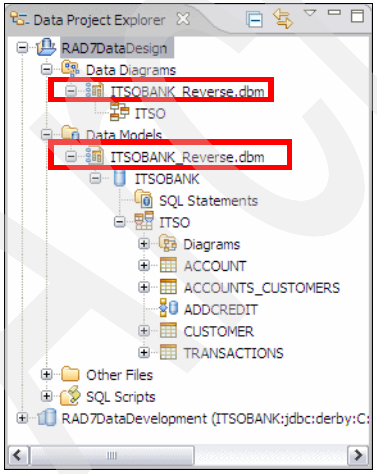
☑ Overview

□ Infer implicit primary keys

□ Infer implicit relationships

Figure 9-29 New Physical Data Model: Options

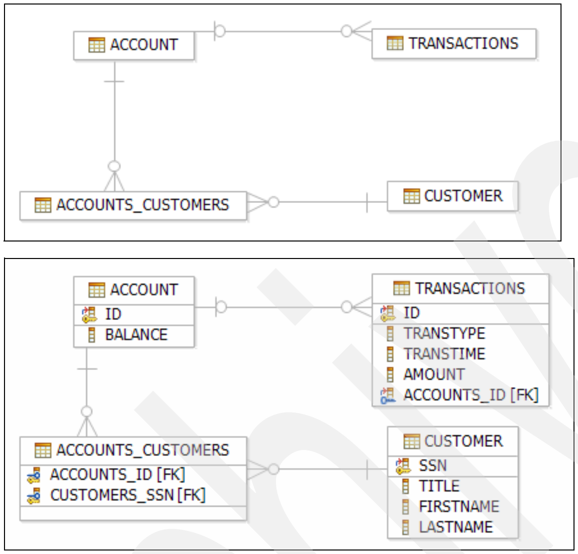
The physical model is created and added to the Data Models folder. The overview diagram is added to the Data Diagrams folder (Figure 9-30).



```
Data Project Explorer
RAD7DataDesign
    Data Diagrams
        ITSOBANK_Reverse.dbm
            ITSO
    Data Models
        ITSOBANK_Reverse.dbm
            ITSOBANK
                SQL Statements
                ITSO
                    Diagrams
                    ACCOUNT
                    ACCOUNTS_CUSTOMERS
                    ADDCREDIT
                    CUSTOMER
                    TRANSACTIONS
    Other Files
    SQL Scripts
RAD7DataDevelopment (ITSOBANK:jdbc:derby:C:
```

Figure 9-30 Data Design Project with physical model and diagram

- The Physical Data Model editor is open on the `ITSOBANK_Reverse` model. Descriptive information can be added. Do not close the physical model. It must be open to open the diagram.
- Expand **RAD7DataDesign** → **Data Diagrams** → **ITSOBANK_Reverse.dbm** and open the **ITSO** overview diagram. It contains all the tables that are in the schema. You can move the tables to get a better diagram (Figure 9-31). In the Properties view, **Filters** page, select **Show key** and **Show non-key**.



Initial diagram

Diagram with Show key and Show non-key

Figure 9-31 Overview diagram

- Select a table in the diagram, then look at the Properties view to see the columns and relationships.
- Select a relationship (line) in the diagram and look at the Properties view to see the cardinality (Details tab). The cardinality is also displayed visually in the diagram.
- Close the model.

Creating a physical data model from a template

To create a physical data model from a template, do these steps:

- Right-click **RAD7DataDesign** and select **New** → **Physical Data Model**. In the New Physical Data Model wizard (refer to Figure 9-28 on page 388), enter the following information:
- File name: **Bank_model**
- Database and Version: **Derby, 10.1**

- Select **Create from template**.
- Click **Finish**.

The physical model is created and displayed in the Data Models folder. The data diagram for the schema opens in the diagram editor.

Modeling with diagrams

You can use data diagrams to visualize and edit objects that are contained in data projects. Data diagrams are a view representation of an underlying data model. You can create diagrams that contain only a subset of model objects that are of interest.

In this section, we create a schema named `RAD7Bank` in the physical data model. Under this schema, we create two tables: `ACCOUNT` and `TRANSACTIONS`.

- In the Data Project Explorer select **RAD7DataDesign** → **Data Models** → **Bank_model.dbm** → **New Database** → **Schema**, and in the Properties view change the schema name from Schema to **RAD7Bank**.
- To add a table, do these steps:
  - Select the **Data** drawer in the palette and select **Table** in the Data drawer.
  - Click the empty area in the data diagram. A new table is added to the diagram.
  - Overtype the table name with **ACCOUNT**.
- Hover the mouse over the **ACCOUNT** table in the diagram and you see four icons appearing outside of the table (Figure 9-32). Click the **Add Key** icon.

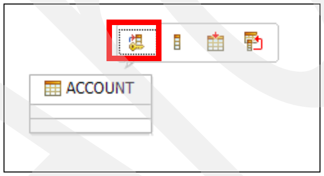
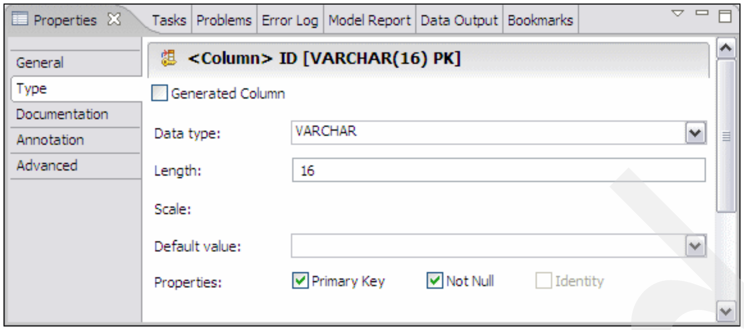


Figure 9-32 Add key, column, index, and trigger

- Overtype the name with ID (or change the name in the Properties view, General tab).
- Select the ID column, and in the Properties view, Type tab, change the Data type to **VARCHAR**. Set the Length to **16** (Figure 9-33).



<Column> ID [VARCHAR(16) PK]

□ Generated Column

Data type: VARCHAR

Length: 16

Scale:

Default value:

Properties: ☑ Primary Key ☑ Not Null □ Identity

Figure 9-33 Edit the key

> **Note:** The key **ID** must be uppercase. If you use lowercase, you might get the following error message when you run DDL on server in later section:
>
> ```
> SQL Exception: 'ID' is not a column in table or VTI 'ACCOUNT'..
> ```

- Hover the mouse over the **ACCOUNT** table and click the **Add Column** icon.
- In the Properties view change the Name to **BALANCE**, the data type to **INTEGER**, and select **Not Null**.
- Follow the same procedure to create the **TRANSACTIONS** table:
  - Key:

    ```
    ID           VARCHAR(250) NOT NULL
    ```

  - Columns:

    ```
    TRANSTYPE    VARCHAR(32) NOT NULL
    TRANSTIME    TIMESTAMP NOT NULL
    AMOUNT       INTEGER NOT NULL
    ACCOUNTS_ID  VARCHAR(16)
    ```

- The data diagram is shown in Figure 9-34.

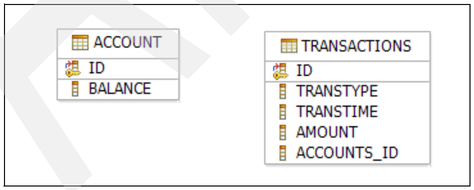
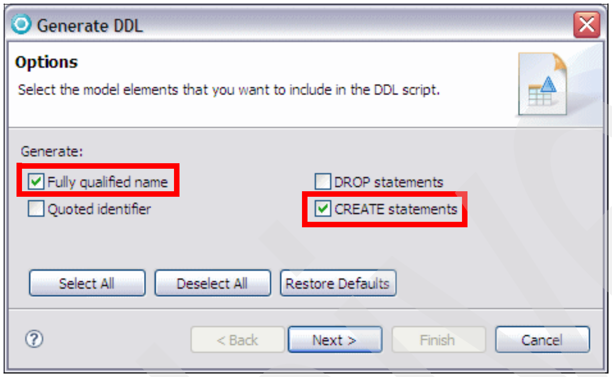


Figure 9-34 Data diagram with two tables

Generating DDL from physical data model and deploy

In this section, we generate the DDL and run the generated DDL into the Derby database:

- In the Data Project Explorer, right-click the schema **RAD7Bank** → **Generate DDL**.
- In the Generate DDL dialog, select **Fully qualified name** and **CREATE statements** (Figure 9-35), and click **Next**.



Generate DDL

Options

Select the model elements that you want to include in the DDL script.

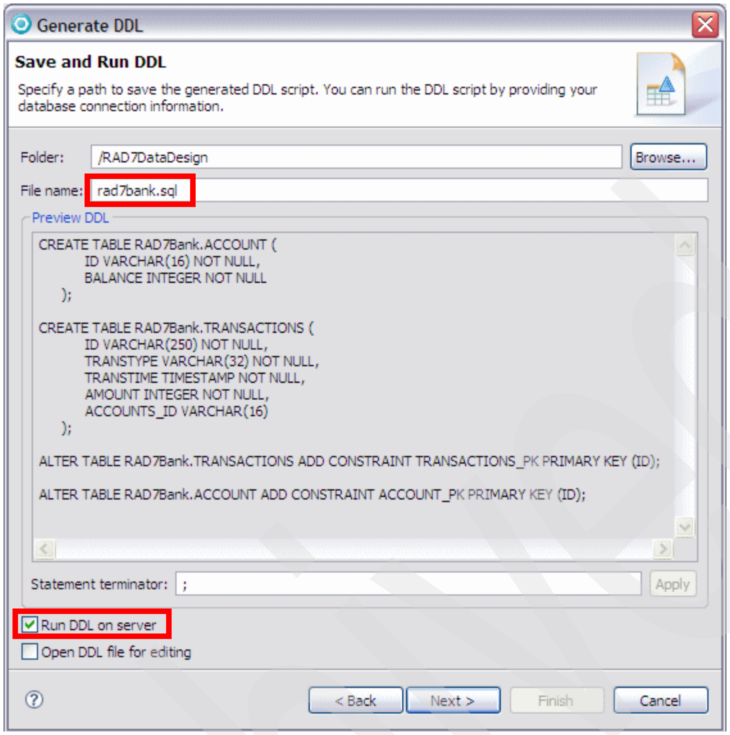
Generate:

☑ Fully qualified name
☐ Quoted identifier
☐ DROP statements
☑ CREATE statements

Quoted identifiers are required if the names contains blanks.

Figure 9-35 Generate DDL

- In the Objects page, leave everything selected and click **Next**.
- In the Save and Run DDL page, change the file name to **rad7bank.sql**, review the DDL, select **Run DDL on server**, and click **Next** (Figure 9-36).



Save and Run DDL

Specify a path to save the generated DDL script. You can run the DDL script by providing your database connection information.

Folder: /RAD7DataDesign

File name: rad7bank.sql

Preview DDL

```
CREATE TABLE RAD7Bank.ACCOUNT (
        ID VARCHAR(16) NOT NULL,
        BALANCE INTEGER NOT NULL
    );

CREATE TABLE RAD7Bank.TRANSACTIONS (
        ID VARCHAR(250) NOT NULL,
        TRANSTYPE VARCHAR(32) NOT NULL,
        TRANSTIME TIMESTAMP NOT NULL,
        AMOUNT INTEGER NOT NULL,
        ACCOUNTS_ID VARCHAR(16)
    );

ALTER TABLE RAD7Bank.TRANSACTIONS ADD CONSTRAINT TRANSACTIONS_PK PRIMARY KEY (ID);

ALTER TABLE RAD7Bank.ACCOUNT ADD CONSTRAINT ACCOUNT_PK PRIMARY KEY (ID);
```

Statement terminator: ;

☑ Run DDL on server

□ Open DDL file for editing

Figure 9-36 Save and Run DDL

- In the Select connection page, select **Use an existing connection**, select **ITSOBank**, and click **Next**.
- In the Summary page, click **Finish**.
- The SQL script is created and stored in the **SQL Scripts** folder. Open `rad7bank.sql` to review the DDL.
- In the Database Explorer right-click the **ITSOBank** connection and select **Refresh**.

Tip: You cannot see the `RAD7BANK` schema! Remember that your schemas are filtered. Right-click **Schemas** → **Filters** and select **Disable filters**, then click **Finish**. The `RAD7BANK` schema with two tables is now visible.

Modifying the data model

In this section, we add a foreign key relationship between the `ACCOUNT` and `TRANSACTIONS` tables:

- Hover the mouse over the **ACCOUNT** table object in the diagram and you should see two arrows appearing outside of the table, pointing in opposite directions. Use the arrow that points away from the **ACCOUNT** table (representing a relationship from parent to child) to create a relationship between the **ACCOUNT** table and the **TRANSACTIONS** table.
- Drag the arrow that points away from the **ACCOUNT** table, and drop it on the **TRANSACTIONS** table. In the menu that opens select **Create Non-Identifying Optional FK Relationship** (Figure 9-37)

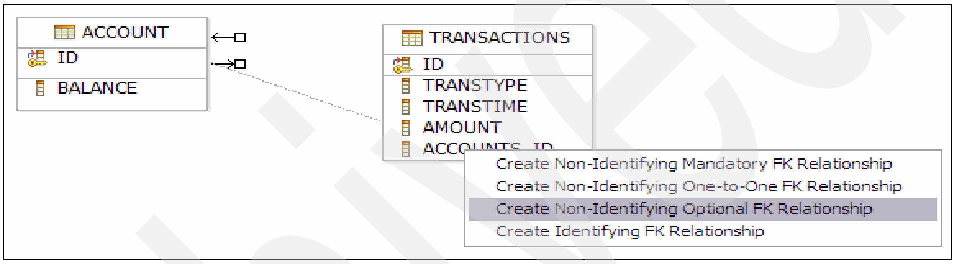
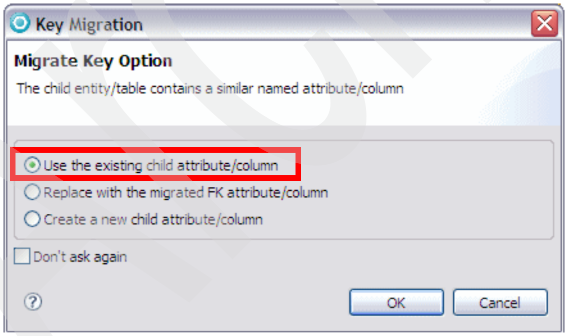

ACCOUNT
ID
BALANCE
TRANSACTIONS
ID
TRANSTYPE
TRANSTIME
AMOUNT
ACCOUNTS_ID
Create Non-Identifying Mandatory FK Relationship
Create Non-Identifying One-to-One FK Relationship
Create Non-Identifying Optional FK Relationship
Create Identifying FK Relationship

Figure 9-37 Add relationship between tables

- In the Migrate Key Option dialog, select **Use the existing child attribute/column**, and click **OK** (Figure 9-38).


Key Migration

Migrate Key Option

The child entity/table contains a similar named attribute/column

● Use the existing child attribute/column
○ Replace with the migrated FK attribute/column
○ Create a new child attribute/column

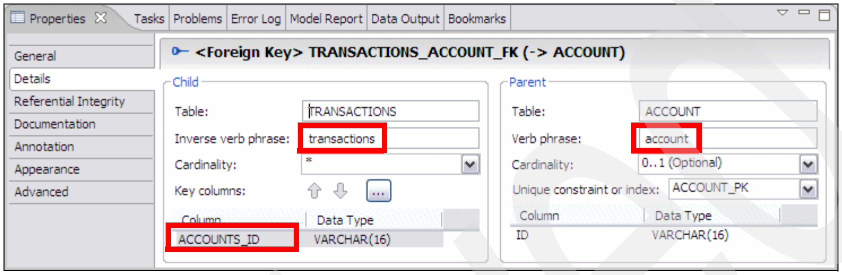
□ Don't ask again

We use an existing column (`ACCOUNTS_ID`) as foreign key

Figure 9-38 Key migration

- Select the foreign key relationship you just created. In the Properties view, select the **Details** page.

- Click the ellipsis button ... next to the Key Columns field. In the dialog, that opens, select **ACCOUNTS_ID** and clear **ID** (use the check boxes). Click **OK**.
- Add information to the relationship properties to identify the roles of each table in the relationship (Figure 9-39):
  - In the Inverse Verb Phrase field, type **transactions**.
  - In the Verb Phrase field, type **account**.
  - Leave the cardinality as * and 0..1.



<Foreign Key> TRANSACTIONS_ACCOUNT_FK (-> ACCOUNT)

Child

Table: TRANSACTIONS

Inverse verb phrase: transactions

Cardinality: *

Key columns:

Column	Data Type
ACCOUNTS_ID	VARCHAR(16)

Parent

Table: ACCOUNT

Verb phrase: account

Cardinality: 0..1 (Optional)

Unique constraint or index: ACCOUNT_PK

Column	Data Type
ID	VARCHAR(16)

Figure 9-39 Relationship Details page

- Save the diagram. Notice the relationship verbs in the diagram (Figure 9-40).

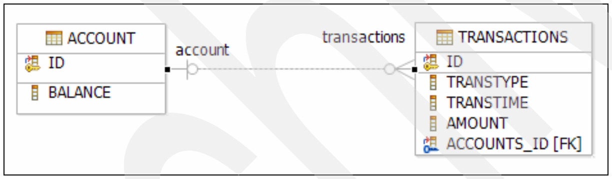


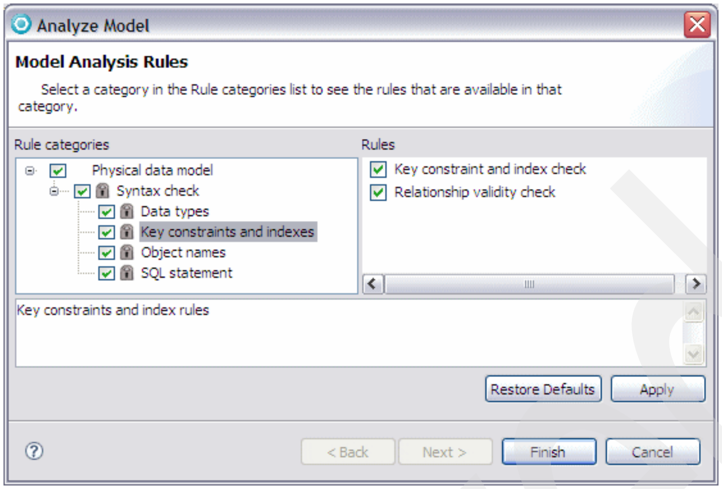
Figure 9-40 Relationship with verbs

Analyzing the data model

The Analyze Model wizard analyzes a data model to ensure that it meets certain specifications. Model analysis helps to ensure model integrity and helps to improve model quality by providing design suggestions and best practices.

To analyze the RAD7Bank schema in the physical data model:

- Right-click the schema **RAD7Bank** in the Data Project Explorer and select **Analyze Model**.
- The Analyze Model dialog opens (Figure 9-41). Select the different items in the list to see the rules that are checked.
- Click **Finish**.



Analyze Model

Model Analysis Rules

Select a category in the Rule categories list to see the rules that are available in that category.

Rule categories
- ☑ Physical data model
  - ☑ Syntax check
    - ☑ Data types
    - ☑ Key constraints and indexes
    - ☑ Object names
    - ☑ SQL statement

Rules
- ☑ Key constraint and index check
- ☑ Relationship validity check

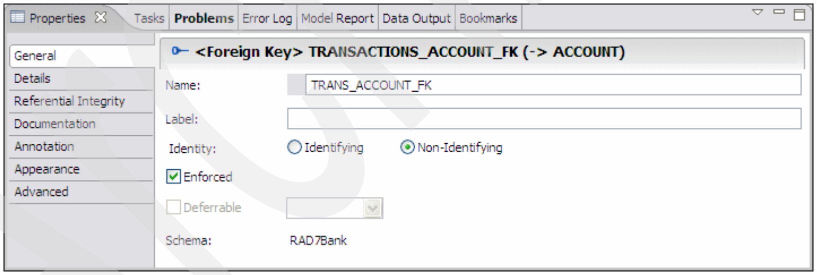
Key constraints and index rules

Figure 9-41 Analyze Model

- In the Problems view, an error message is displayed:

  ```
  The name length of TRANSACTIONS_ACCOUNT_FK exceeds identifier length limit
  which is 18
  ```

- To correct this problem, select the foreign key relationship in the data diagram. In the Properties view, General page, change the Name from `TRANSACTIONS_ACCOUNT_FK` to **TRANS_ACCOUNT_FK** (Figure 9-42).



<Foreign Key> TRANSACTIONS_ACCOUNT_FK (-> ACCOUNT)

Name: TRANS_ACCOUNT_FK

Label:

Identity: ○ Identifying ● Non-Identifying

☑ Enforced

□ Deferrable

Schema: RAD7Bank

Figure 9-42 Change the name of the foreign key

- Save the diagram and rerun **Analyze Model**. The error in the problems view is gone.

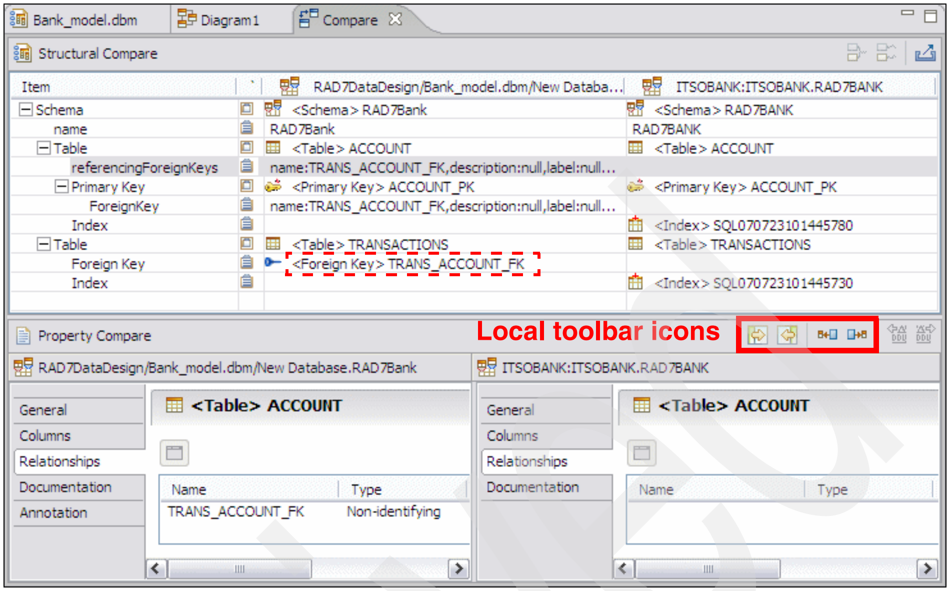
Comparing data objects and managing changes

You can use the comparison editor to compare data objects or analyze impact and dependencies. You can compare objects with other objects or with another version of the same object. The differences between data objects are highlighted in the compare editor, so that you can browse and copy changes between the compared resources. You can navigate and merge structural differences between data objects, merge properties changes between data objects, generate DDL for changes that you made, and export the structural differences to an XML file on the file system.

Comparing data objects

To compare and merge the changed physical data model (in the Workbench) with the original schema (in the database), do these steps:

- Right-click the **ITSOBANK [Derby 10.1]** connection in the Database Explorer, and select **Refresh**.
- In the Data Project Explorer, right-click the **RAD7Bank** schema and select **Compare With** → **Another Data Object**. The compare editor opens.
- In the Compare with Another Data Object window, expand **ITSOBank (ITSOBANK)** and select schema **RAD7Bank.** Click **OK**.
- The compare editor opens with a Structural Compare view and a Property Compare view. The physical data model is displayed on the left, and the original source from the database is displayed on the right. The columns at the top of the Structural Compare view display the name of each object.
- Expand the data objects in the Structural Compare view to see the changes that you made to the physical data model (Figure 9-43).
- In the Structural Compare view, navigate through each difference. You can use the **Go to Next Difference** ⇩ and **Go to Previous Difference** ⇧ toolbar buttons on the main toolbar to navigate.



Item		RAD7DataDesign/Bank_model.dbm/New Databa...	ITSOBANK:ITSOBANK.RAD7BANK
Schema		<Schema> RAD7Bank	<Schema> RAD7BANK
name		RAD7Bank	RAD7BANK
Table		<Table> ACCOUNT	<Table> ACCOUNT
referencingForeignKeys		name:TRANS_ACCOUNT_FK,description:null,label:null...	
Primary Key		<Primary Key> ACCOUNT_PK	<Primary Key> ACCOUNT_PK
ForeignKey		name:TRANS_ACCOUNT_FK,description:null,label:null...	
Index			<Index> SQL070723101445780
Table		<Table> TRANSACTIONS	<Table> TRANSACTIONS
Foreign Key		<Foreign Key> TRANS_ACCOUNT_FK	
Index			<Index> SQL070723101445730

RAD7DataDesign/Bank_model.dbm/New Database.RAD7Bank

<Table> ACCOUNT

Name	Type
TRANS_ACCOUNT_FK	Non-identifying

ITSOBANK:ITSOBANK.RAD7BANK

<Table> ACCOUNT

Name	Type

Figure 9-43   Compare data objects

- We can see the following differences between the modified data model modified and the model in the Derby database:
  - We added a foreign key in the modified physical data model.
  - When you export the database model to the Derby server, the Derby database automatically creates an index on the primary key (one in each table on the right-hand side).

Analyzing impact and dependency for data objects

You can use the Impact Analysis dialog to perform an impact analysis and to find dependencies among data objects, so that you can understand how changes you make to one data object impacts other related objects.

There are several options for impact analysis in the tooling:

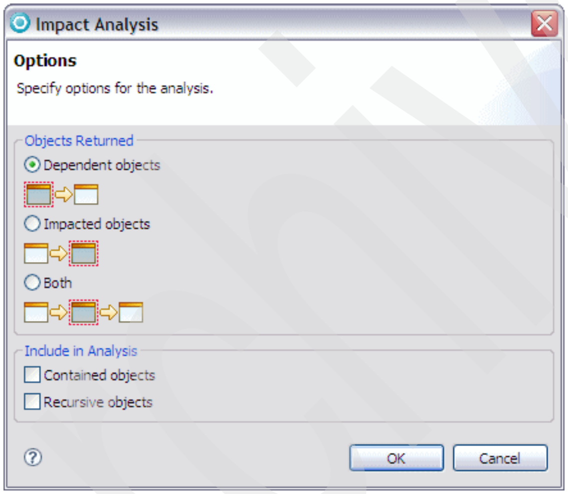
- **Dependent objects**—Find objects that the selected object depends on.
- **Impacted objects**—Find objects that the selected object impacts. For example, if you modify an object, you can view the list of objects that might be impacted.
- **Both**—Find objects that the selected object both depends on and impacts.
- **Contained objects**—Analyze objects under the containment hierarchy to find objects that the selected object depends on and impacts.

- **Recursive objects**—Analyze objects recursively to find objects that the selected object depends on and impacts.

We want to find the impact when adding the foreign key relationship to the data model in the database. First we add the relationship in the compare editor, analyze the impact, and then perform the change.

To analyze impact and dependency for data objects:

- Select **<Foreign Key> TRANS_ACCOUNT_FK**, and click the **Copy from Left to Right** local toolbar button in the compare editor. A number of entries are added to the database model.
- Select **<Foreign Key> TRANS_ACCOUNT_FK**, and click **Analyze Right Impact** local toolbar button.
- The Analyze Impact dialog opens (Figure 9-44):



Impact Analysis

Options

Specify options for the analysis.

Objects Returned

● Dependent objects

○ Impacted objects

○ Both

Include in Analysis

□ Contained objects

□ Recursive objects

Figure 9-44 Impact Analysis options

- Select **Dependent objects** and click **OK**.
- You can see the following information in the **Model Report** view:
  - Dependent Object: `ITSOBANK.RAD7Bank.TRANSACTIONS.TRANS_ACCOUNT_FK`
  - Dependent Object Type: Foreign Key
  - Impactor Object: `ITSOBANK.RAD7Bank.TRANSACTIONS.ACCOUNTS_ID`
  - Impactor Object Type: Column

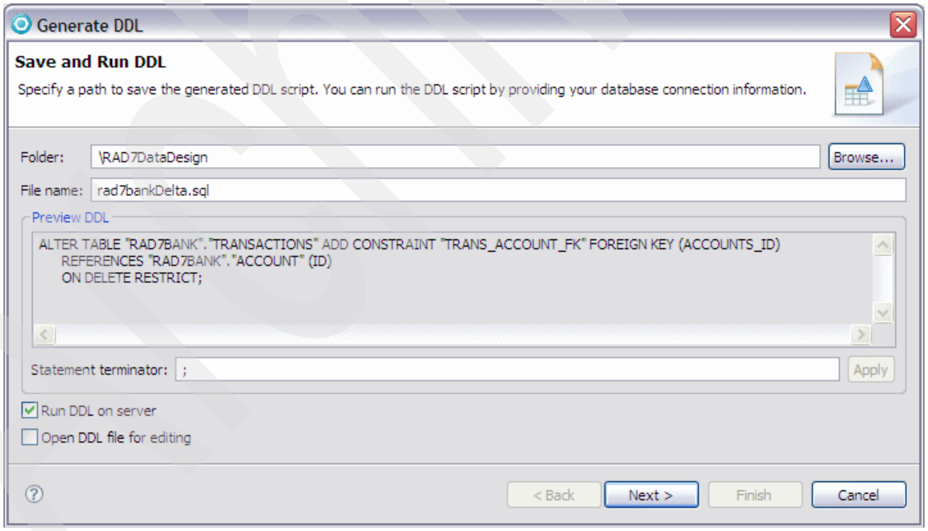
Updating the database model with the changes

Because we are making changes to the server object, we have generated a DDL script to capture the changes that we made in the compare editor:

- Click the **Generate Right DDL** local toolbar button. The Generate DDL wizard opens.

> **Tip:** If the Generate Right DDL button is not active, close the compare editor and run the comparison again (right-click the **RAD7Bank** schema and select **Compare With** → **Another Data Object**). Then copy the **<Foreign Key> TRANS_ACCOUNT_FK** from left to right, and the button is active.

- In the Save and Run DDL page, examine the DDL in the **Preview DDL** section. Specify the following settings (Figure 9-45) and click **Next**:
  - Select the design project as the Folder.
  - Type **rad7bankDelta.sql** as the File name.
  - Review the DDL, which adds the foreign key constraint to the `TRANSACTIONS` table.
  - Select **Run DDL on server**.



Generate DDL

Save and Run DDL

Specify a path to save the generated DDL script. You can run the DDL script by providing your database connection information.

Folder: \RAD7DataDesign

File name: rad7bankDelta.sql

Preview DDL

```
ALTER TABLE "RAD7BANK"."TRANSACTIONS" ADD CONSTRAINT "TRANS_ACCOUNT_FK" FOREIGN KEY (ACCOUNTS_ID)
    REFERENCES "RAD7BANK"."ACCOUNT" (ID)
    ON DELETE RESTRICT;
```

Statement terminator: ;

☑ Run DDL on server

☐ Open DDL file for editing

Figure 9-45 Save and Run DDL

- In the Select Connection page, select **Use an existing connection** and **ITSOBANK** under the Existing connections list, and click **Next**.
- In the Summary page, click **Finish**.
- In the Database Explorer, refresh the **ITSOBANK** connection. Expand the schema `RAD7Bank`. You can see that the `TRANSACTIONS` table now has a foreign key constraint **TRANS_ACCOUNT_FK** in the Constraints folder (Figure 9-46).

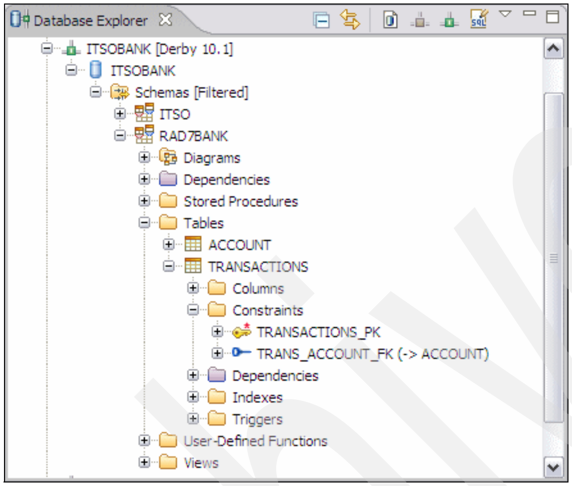


Figure 9-46 Foreign key constraint

More information

More information on JDBC can be found at:

`http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/`
`http://java.sun.com/javase/technologies/database/`
`http://developers.sun.com/product/jdbc/drivers`

More information on SQLJ can be found at:

`http://www.javaworld.com/javaworld/jw-05-1999/jw-05-sqlj.html`
`http://www.onjava.com/pub/st/27`

10

Develop GUI applications

Application Developer provides a visual editor for building Java graphical user interfaces (GUIs).

In this chapter we introduce the visual editor and develop a sample GUI, which interacts with back-end business logic. This GUI is runnable as a JavaBean and as a Java application.

The chapter is organized into the following sections:

- Introduction to the visual editor
- Prepare for the sample
- Launching the visual editor
- Visual editor overview
- Working with the visual editor

Note: The sample code described in this chapter can be completed by following along in the procedures documented. Alternatively, you can import the sample code provided in the `C:\7501code\gui\RAD7Gui.zip` project interchange file. For details, refer to Appendix B, “Additional material” on page 1307.

Introduction to the visual editor

The visual editor is used to design applications containing a graphical user interface (GUI) based on the JavaBeans component model. It supports visual construction using either the Abstract Window Toolkit (AWT), Swing, or the Standard Widget Toolkit (SWT).

More information concerning Swing and AWT can be found at:

```
http://java.sun.com/j2se/1.4.2/docs/guide/swing/
http://java.sun.com/j2se/1.4.2/docs/guide/awt/
```

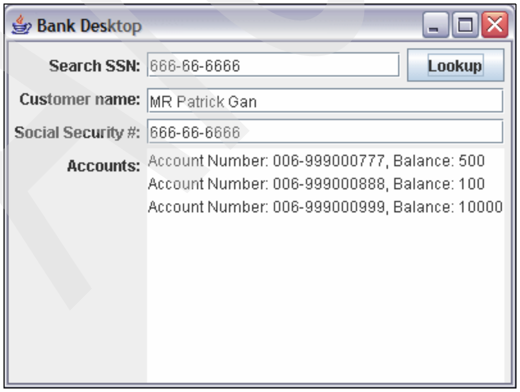
The SWT is part of Eclipse. More information can be found at:

```
http://www.eclipse.org/platform/
```

The visual editor allows you to design GUIs visually. Using the visual editor, you can drag beans from different palettes, manipulate them in the Design view, and edit their properties in the Properties view. The visual editor also includes a Source view where you can both see and modify the generated Java code. You can make changes in either the Source view or in the Design view—the visual editor uses a process known as round-tripping to synchronize the two views.

Prepare for the sample

The sample GUI we develop in this chapter is shown in Figure 10-1. It allows a user to search for a specific social security number and view full information about the customer and the accounts held by the customer. We use Swing components for our sample.



Search SSN: 666-66-6666

Customer name: MR Patrick Gan

Social Security #: 666-66-6666

Accounts:

```
Account Number: 006-999000777, Balance: 500
Account Number: 006-999000888, Balance: 100
Account Number: 006-999000999, Balance: 10000
```

Figure 10-1 Sample GUI

By creating the sample GUI, you should learn how to work with the visual editor and how to compose and add visual components, change their properties, add event handling code, and run the GUI.

To set up your development environment for the GUI sample application, you have to create the database that the GUI uses and import some database access classes.

Creating the project for the sample

To create the RAD7GUI project for our sample with Derby database support, do these steps:

- In the Workbench, select **File** → **New** → **Project**.
- In the first page of the New Project wizard, select **Java Project** and select **Next**.
- In the second page of the wizard, specify these choices and then click **Next**:
  - Project name: **RAD7GUI**
  - Select **Create new project in workspace** (default).
  - Select **Use project folder as root for sources and class files** (default).
  - Click **Finish**.
- When prompted, switch to the Java perspective. If you have turned off the prompt for switching of perspectives (using **Window** → **Preferences** → **General** → **Perspectives**), you can open the Java perspective manually.
- Optionally, close any other perspective to save memory.

Adding the JDBC driver for Derby to the project

To add the Derby JDBC driver to the project, do these steps:

- Select the project, right-click, and select **Properties**.
- Select **Java Build Path**.
- Add the JAR file that contains the JDBC drivers for the Derby database used in the sample.
  - Select the **Libraries** tab at the top of the dialog and click **Add Variable**.

    A further dialog opens, allowing you to select from a list of predefined variables. By default, there is no variable defined for the Derby JAR file, so we have to create one.
  - Select **Configure Variables**, and in the resulting dialog click **New**.
  - Enter DERBY_DRIVER_JAR in the Name field and click **File**.

– Find the appropriate JAR file:

  `<rad_home>\runtimes\base_v61\derby\lib\derby.jar`

– Click **Open**, **OK**, and **OK** and which takes you back to the New Variable Classpath Entry dialog.

– Select the **DERBY_DRIVER_JAR** variable you just created and click **OK**.

- Click **Finish** in the New Java Project wizard.

Note: Creating a variable is a good practice for this kind of situation. The alternative is to select **Add External JARs** which adds the full path to the JAR file into the project build settings. A variable can be shared among all the projects in the workspace, saving time for JAR files that are likely to be used elsewhere.

Setting up the sample database

The `ITSOBANK` database can be set up using either DB2 or Derby. In this chapter we use the Derby database.

The instructions for creating the `ITSOBANK` database are in "Setting up the ITSOBANK database" on page 1312.

Importing the model classes for the sample

To import the model classes for the GUI sample, do these steps:

- Right-click the **RAD7GUI** project and select **Import** → **General** → **File System**. Click **Next**.
- Click **Browse** and locate the folder `C:\7501code\gui`.
- Select the **gui** folder and click **Finish**.

Imported code

The imported model classes in the `itso.rad7.model.entity` package are:

- `Account`—JavaBean with two properties: id, balance.
- `Customer`—JavaBean with four properties: ssn, firstName, lastName, title.
- `AccountFactory`—Retrieve accounts from the database by id and customer.
- `CustomerFactory`—Retrieve customer from the database by ssn.
- `DatabaseManager`—Create a JDBC connection to the database.

Launching the visual editor

The visual editor allows you to create and modify GUIs by manipulating JavaBeans in a WYSIWYG (what you see is what you get) editor. There are two ways to launch the visual editor:

- Creating a visual class—Create a visual class in which case the class is automatically associated with the visual editor and opened.
- Open an existing class with the visual editor—Create a Java class and open it with the visual editor.

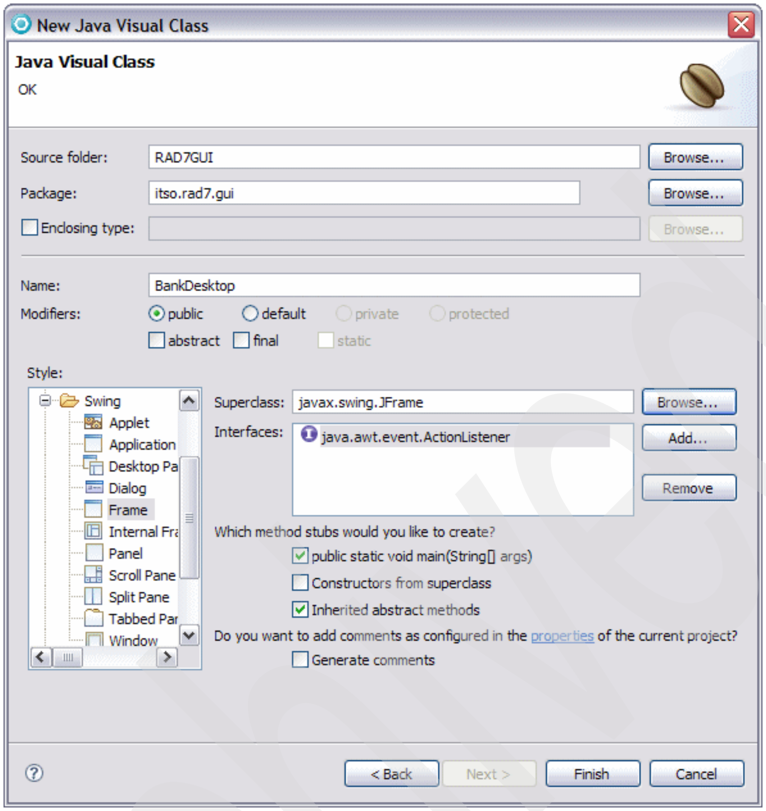
In this example we create a visual class.

Creating a visual class

To create a visual class, do these steps:

- Select **File** → **New** → **Other** → **Java** → **Visual Class**, and then click **Next**.

  Alternatively, right-click the project or package where you want to create the visual class and select **New** → **Other** → **Java** → **Visual Class**.
- In the Create a new Java class dialog, enter the following values (as seen in Figure 10-2):
  - Source Folder: `RAD7GUI`
  - Package: `itso.rad7.gui`
  - Name: `BankDesktop`
  - Style: Expand **Swing** and select **Frame**. (Note that this enters the correct superclass, `javax.swing.JFrame`, into the Superclass field.)
  - Interfaces: Click **Add** next to the Interfaces box, type `ActionListener` (to select `java.awt.event.ActionListener`), and click **OK**.
  - Which method stubs would you like to create? Select **public static void main (String[] args)** and **Inherited abstract methods**. Leave Constructors from superclass unselected.
- Click **Finish**.



Java Visual Class

OK

Source folder: RAD7GUI

Package: itso.rad7.gui

□ Enclosing type:

Name: BankDesktop

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Style:

- Swing
  - Applet
  - Application
  - Desktop Pa
  - Dialog
  - Frame
  - Internal Fra
  - Panel
  - Scroll Pane
  - Split Pane
  - Tabbed Par
  - Window

Superclass: javax.swing.JFrame

Interfaces: java.awt.event.ActionListener

Which method stubs would you like to create?

☑ public static void main(String[] args)

□ Constructors from superclass

☑ Inherited abstract methods

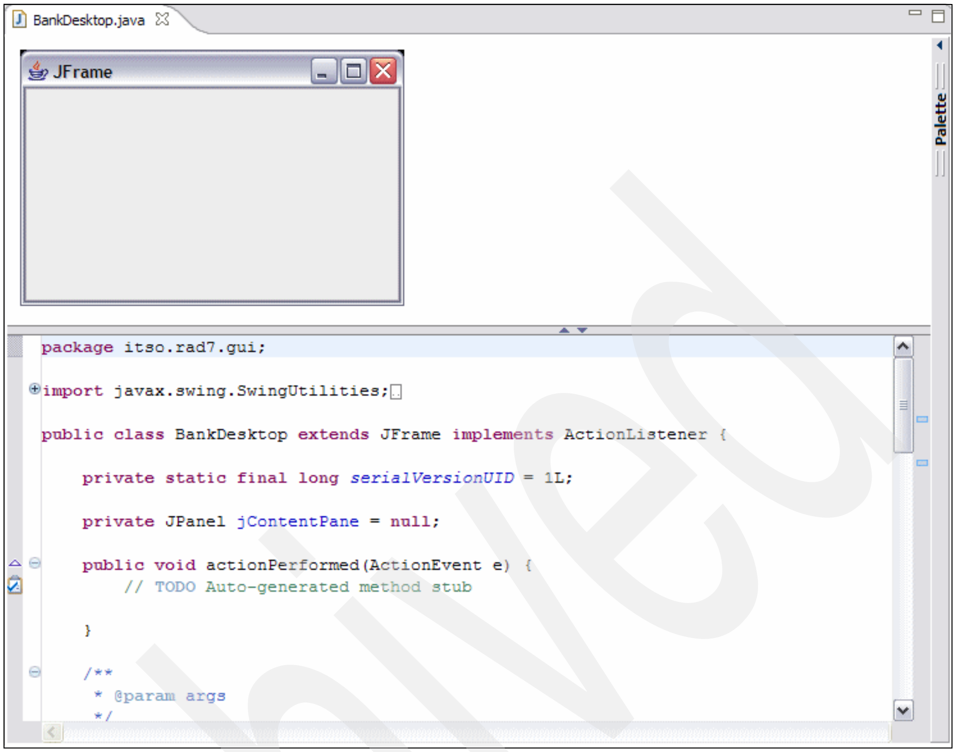
Do you want to add comments as configured in the properties of the current project?

□ Generate comments

Figure 10-2 New Java Visual Class

The `BankDesktop` GUI class opens in the visual editor. this can take a while, because Application Developer has to start a new JVM to display the GUI visually in the Display view.

Figure 10-3 shows the new class `BankDesktop` in the Visual Editor.

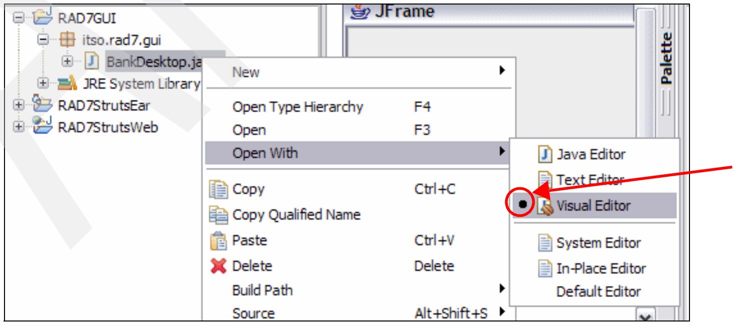


```
package itso.rad7.gui;

import javax.swing.SwingUtilities;□

public class BankDesktop extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;

    private JPanel jContentPane = null;

    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub

    }

    /**
     * @param args
     */
```

Figure 10-3 New Java Class open in the visual editor

Open an existing class with the visual editor

Alternatively, you can open any other Java class with the visual editor by using the context menu. Right-click the class and select **Open With** → **Visual Editor** (Figure 10-4).



Editor that was used the last time

Figure 10-4 Open with context menu

Visual editor overview

This section describes the overall layout of the Visual Editor and some of the options for changing the appearance and operation of the editor.

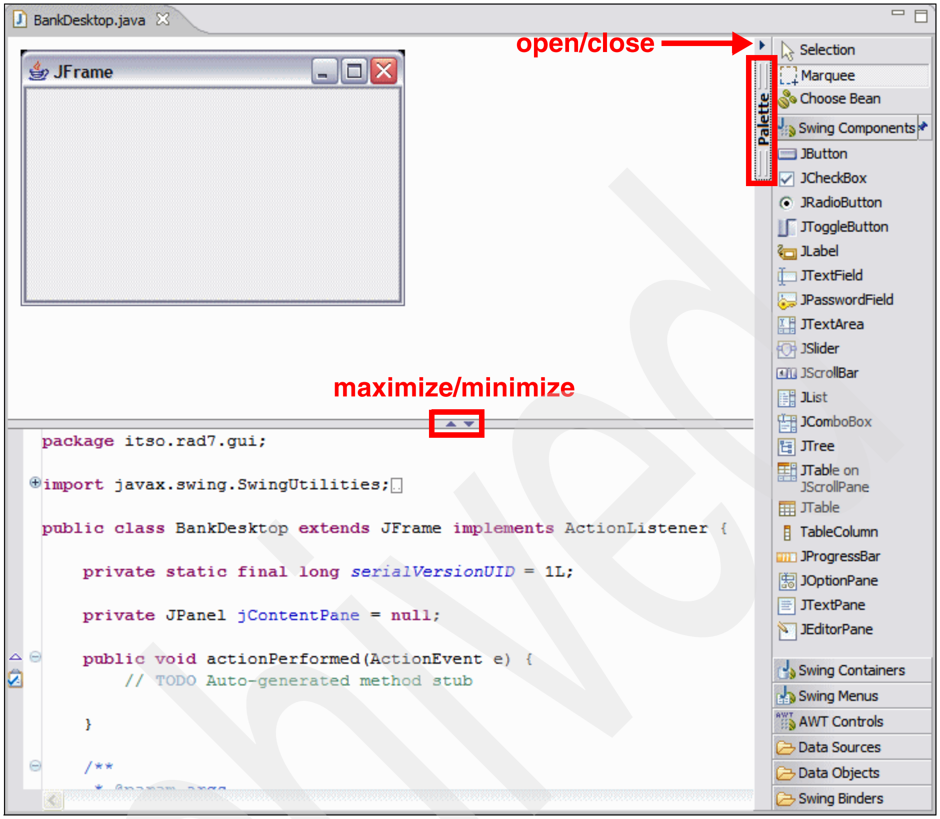
Visual editor layout

The visual editor consists of two panes and a Palette, shown in Figure 10-5:

- A graphical canvas is located in the top section of the visual editor—this is the Design view, where you compose the GUI.
- The source file is displayed beneath it in the Source view, which has all the normal functionality of a Java source editor in Application Developer.
- A Palette of common JavaBeans is available on the right:
  - The Palette allows you to select JavaBeans for building your GUI.
  - The Palette is divided into drawers, which group components by type. The Swing drawer is open because we are creating a Swing GUI.
  - The Palette is initially just a gray bar along the right-hand side of the visual editor pane (Figure 10-3 on page 409). Move the mouse pointer over this bar and the Palette opens (Figure 10-5 shows the Palette revealed).
  - The appearance of the Palette can be modified as explained in “Customizing the appearance of the visual editor” on page 411.

When the visual editor is opened, two additional views open automatically:

- The Java Beans view, in the bottom-left corner of the workspace, displays the structure of the JavaBeans used in your GUI in a tree view.
- The Properties view, at the bottom-right, lets you view and modify attribute settings.



```
package itso.rad7.gui;

import javax.swing.SwingUtilities;

public class BankDesktop extends JFrame implements ActionListener {

    private static final long serialVersionUID = 1L;

    private JPanel jContentPane = null;

    public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub

    }

    /**
     * @param args
```

Figure 10-5 Visual editor layout

Customizing the appearance of the visual editor

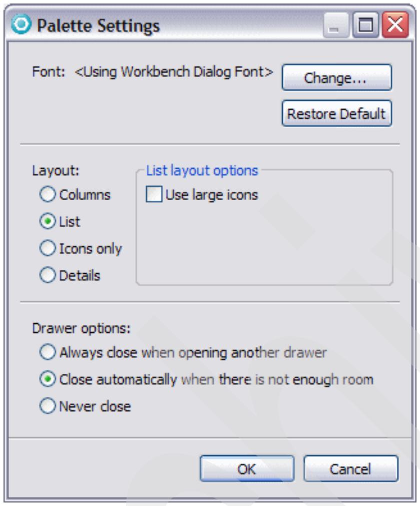
Various aspects of the visual editor can be customized:

Click the up arrow, located in the center-left of the separator between the Design and Source views (Figure 10-5), to maximize the Source view or the down arrow, located to the right of the up arrow, to maximize the Design view.

The Palette automatically hides when not in use, but the appearance of the Palette can be modified:

- Click on a drawer to open or close the drawer.
- Drawers can be fixed open within the Palette by clicking the pin icon next to the drawer name.

- The width of the Palette can also be changed by dragging the left edge (under Paltte), and it can be docked on the left-hand side of the editor if preferred (drag the Palette rectangle).
- Right-click the Palette and use the context menu to change the **Layout** (options: Columns, List, Icons Only, Details) and select whether large or small icons should be displayed.
- Select **Settings** from this menu to open a preferences dialog for the Palette. In addition to the options described above, you can also configure the behavior of the drawers and set the font for the Palette (Figure 10-6).



Font: <Using Workbench Dialog Font>

Layout:
○ Columns
● List
○ Icons only
○ Details

List layout options
☐ Use large icons

Drawer options:
○ Always close when opening another drawer
● Close automatically when there is not enough room
○ Never close

Figure 10-6 JavaBeans Palette Settings

To modify the preferences for the visual editor as a whole, select **Window** → **Preferences** and select **Java** → **Visual Editor**.

The options are as follows:

- Split or notebook view: The editor is shown as a split pane by default, but a tabbed view is also available.
- Views to open: By default, the Java Beans and Properties views opens; this can be turned off.
- Swing look and feel: The default appearance for Swing components can be set using this list. Select the desired look and feel.
- Code generation: Specify whether try/catch blocks and comments should be generated and set the interval for synchronization between the code and visual views of your GUI.

- Pattern styles: Define how the code generation uses visual beans as the GUI is displayed in the Design view.
- The Visual Editor Bindings preferences page allows you to specify the package name for generated code that binds component properties with other Java components. This is covered later in this chapter in “visual editor binding” on page 423.

Working with the visual editor

We continue to develop the `BankDesktop` GUI application to explore the main GUI design features of the visual editor.

Resizing a JavaBean component

To resize a component, select it (either by clicking the component in the Design view or in the Java Beans view). Control handles appear around the component (small black squares at the corners and in the middles of the sides). Moving these resizes the component.

Use this approach to resize the JFrame. Notice that the generated *initialize* method is modified as a result of this change, so it now looks something like Example 10-1 (the numbers in the `setSize` method call are probably different).

Example 10-1 Resize JavaBean component

```
private void initialize() {
   this.setSize(361, 265);
   this.setContentPane(getJContentPane());
   this.setTitle("JFrame");
}
```

This code generates automatically when the visual and code views of your GUI are synchronized (by default this takes place every second).

Code synchronization

We have seen that changes made in the Design view cause changes to the code. The reverse is also true. Change the `setSize` method call in the `initialize` method to `setSize(440, 300)`, as seen in Example 10-2.

Example 10-2 Resize JavaBean component

```
private void initialize() {
   this.setSize(440, 300);
   this.setContentPane(getJContentPane());
   this.setTitle("JFrame");
}
```

After you overtype the first number (the width), replacing it with `440`, wait for about a second and you see the visual representation of the GUI update even without saving the code changes. During the short interval before the views are synchronized, the status bar displays ***`Out of synch`*** instead of the usual `In synch`.

You can also disable synchronization by clicking the **Pause** icon ⏸ in the toolbar. This is a good idea when you want to make a number of large changes to the source without incurring the overhead of the round-tripping for synchronizing the code and visual representations.

Changing the properties of a component

We are going to use the `GridBagLayout` Swing layout manager to position the components in the GUI. By default, the content pane that is created within a `JFrame` uses `BorderLayout`. Do these steps:

- Select the content pane (not frame) by clicking in the grey area within the frame.

  Alternatively, select the component from the list of objects in the **J**ava Beans view (the name is `jContentPane`).

> **Tip:** Once the content pane is selected, if you accidentally press Delete, the content pane is deleted. To add it back, right-click the frame and select **Create Content Pane**.

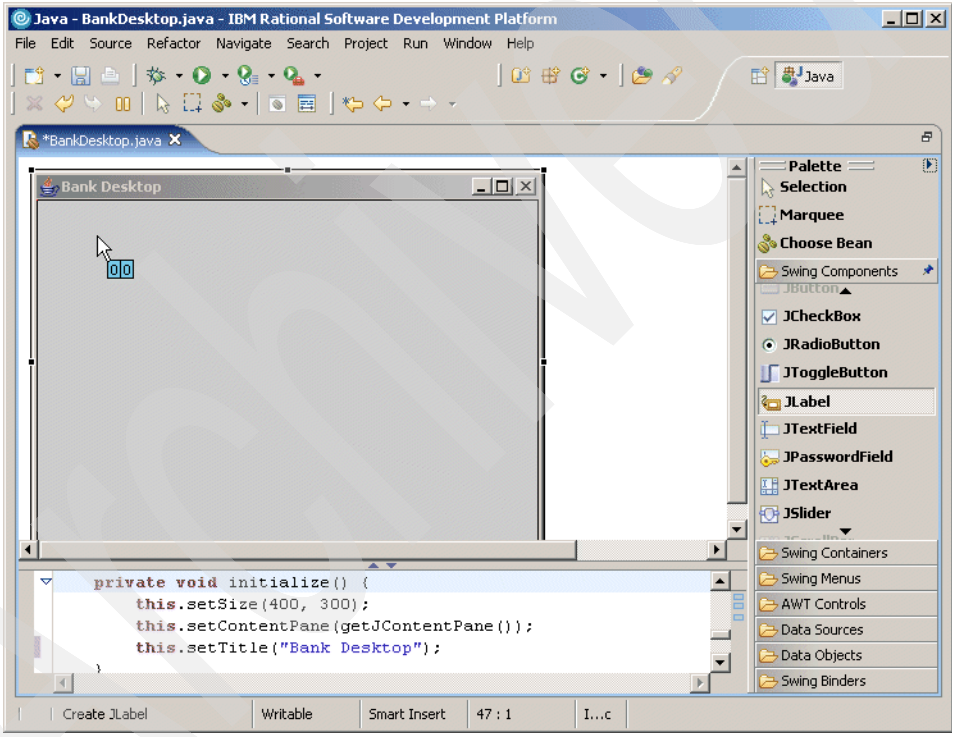
- Select the **Properties** view and scroll down until you can see the layout property with a value of `BorderLayout`.
- Click the value to select it, then click it again to display a list of the available values.
- Select **`GridBagLayout`** from the list. Press **Enter**.
- Select the **JFrame** title bar. Change the title of the JFrame by clicking the title value (**JFrame**) and change it to `Bank Desktop`.
- Save the work.

Adding JavaBeans to a visual class

Now we have to add the various GUI components (or JavaBeans) to the GUI.

- Move the cursor over the Palette to display it (or click the small left arrow **Show Palette** button at the top of the Palette if hidden).
- Open the **Swing components** drawer.
- Select the **JLabel** button and move the mouse pointer over to the content pane.

  The mouse pointer shows a set of coordinates `(0,0)` to indicate in which row and column within the `GridBagLayout` the component is placed (Figure 10-7).



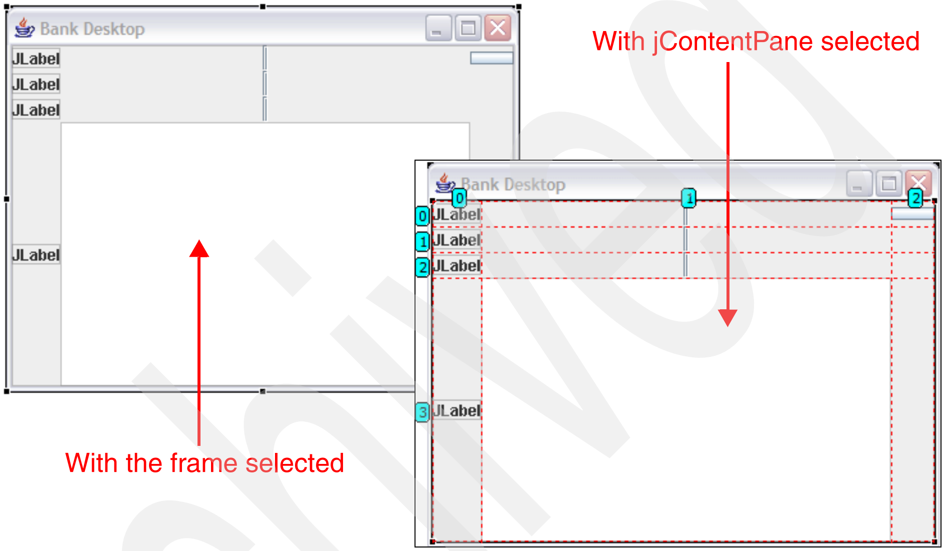
```
private void initialize() {
    this.setSize(400, 300);
    this.setContentPane(getJContentPane());
    this.setTitle("Bank Desktop");
}
```

Figure 10-7 Positioning Swing components

- Click in the content pane, click **OK** to accept the default name, and the JLabel appears in the center of the pane.
- Select **JTextField** from the Palette and drop it to the right of the JLabel.

  When the mouse pointer is in the correct position, the coordinates should be `(1,0)`—indicating column 1, row 0.

- Add a JButton to the right of the JTextField at (2,0). A yellow bar appears in the correct position when you have the mouse pointer placed correctly on the right side of the JTextField.
- Add three JLabels below the first label at (0,1), (0,2), and (0,3).
- Add `JTextFields` at (1,1) and (1,2)—to the right of the first two new JLabels, and a `JList` at (1,3).
- The interface is shown in Figure 10-8.



With jContentPane selected

With the frame selected

Figure 10-8 Basic GUI layout

Working with the Properties view

To change the GUI to our preferences and to set the label and button text, we use the Design view and the Properties view together.

Expand the text fields

The text fields are very small by default. We want to expand them:

- Select the first text field, expand **constraint** in the Properties view, and set the fill value to `HORIZONTAL`.
- Repeat this for the next two text fields. When the text field is selected, also drag the green handle on the right side over the third column to use the whole space. This changes the grid width field (under constraint) to the value 2.
- Select the jList and also drag the green handle over the third column.

- The changed layout is shown in Figure 10-9.

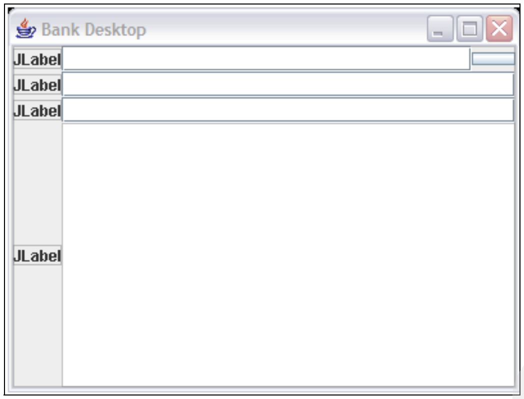


Figure 10-9 Resizing components within the layout

Change and align the label and the button texts

Next we set the texts of the labels and the button:

- Select the top **JLabel** and then click it again to make the text editable. Change the text to `Search SSN:`.
- Change the other JLabel text entries in the same way to `Customer Name:`, `Social Security #:`, and `Accounts:`. By default all the texts are centered.
- Select the button and change the text to `Lookup`.
- Select each text label, expand constraint, and set the **anchor** value to `EAST`.
- For the jList label set the anchor value to `NORTHEAST`.
- At this stage the GUI looks like Figure 10-10.

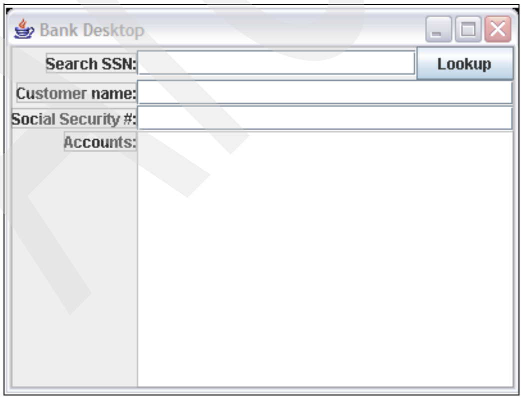


Figure 10-10 Changed and aligned text labels

Setting the fonts

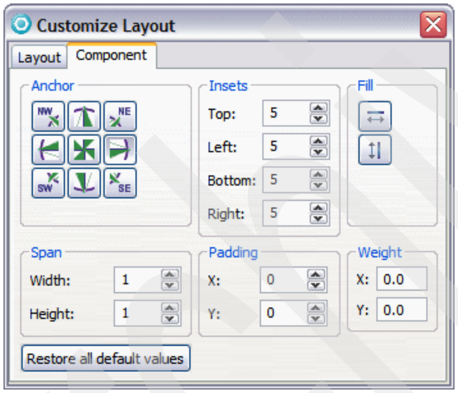
Next we set the fonts of the texts fields and the list.

- Select a text field and notice the **font** value in the Properties view: `Dialog, plain, 12`
- Select the jList and notice that the font is different: `Dialog, bold, 12`
- Click ... on the right of the font value. In the Java Property Editor change the Style from bold to plain and click **OK**.

Adding space between the components

The GUI looks compressed with labels and fields packed together.

- Right-click the first label and select **Customize Layout**.
- In the Customize Layout dialog select **5** for all the insets. This adds a little space between the components.


Customize Layout

Component

Insets
Top: 5
Left: 5
Bottom: 5
Right: 5

Span
Width: 1
Height: 1

Padding
X: 0
Y: 0

Weight
X: 0.0
Y: 0.0

Figure 10-11 Customer the layout of components

- Repeat this for all the labels and the button.
- For the text fields and the list, only set the right inset to the value 5.

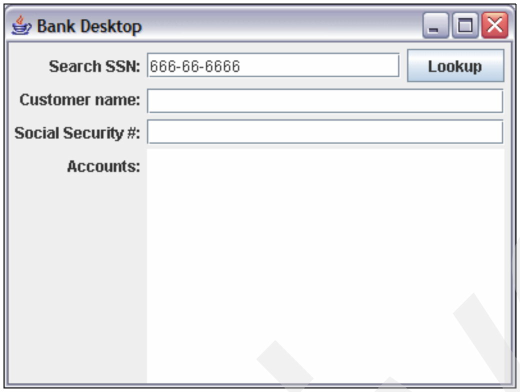
Setting user friendly field names

By default, the text fields are named `jTextField`, `jTextField1`, and so forth.

- Select the first text field and in the Properties view change the **field name** to `searchSSN`. Alternativerly, right-click the field and select **Rename Field**.
- Change the other text fields to `customerName` and `customerSSN`. Change the list to `accountList`. Change the button name to `lookupButton`.
- Notice that the get methods change to `getSearchSSN`, and so forth.
- Save the changes.

Testing the appearance of the GUI

As a quick test of the appearance of the GUI, we can run the JFrame as it is. Select **Run** → **Run As** → **Java Bean**. The GUI starts and appears as a separate window (Figure 10-12).



Search SSN: 666-66-6666

Customer name:

Social Security #:

Accounts:

Figure 10-12 First GUI test

The GUI works up to a point—you should be able to type into the text fields and click **Lookup**—but we have not yet coded any event handling behavior.

Close the GUI by clicking the close box at the top-right.

Adding event handling to GUI

For our example, we want the user to be able to enter a search term into the Search SSN field (a social security number for a customer) and click **Lookup**. When the button is clicked, we have to search the database for the customer details, find the accounts for the customer, and put the account information in the list at the bottom of the GUI.

Add event handling

To add event handling to the GUI, do these steps:

- Register the `JFrame` as an `ActionListener` for the `JButton` so that it receives messages when the button is clicked.

  The best place to do this is in the initialize method of `BankDesktop.java`, as seen in Example 10-3.

Example 10-3 Add addActionListener sample

```
private void initialize() {
   this.setSize(400, 300);
   this.setContentPane(getJContentPane());
   this.setTitle("Bank Desktop");
   this.getLookupButton().addActionListener(this);
}
```

- Add code to the `actionPerformed` method.

  Because we chose to implement the `ActionListener` interface when we created this class, the required method is already present, although at the moment it does not do anything:

  ```
  public void actionPerformed(ActionEvent e) {
        // TODO Auto-generated method stub
     }
  ```

- Change the code of the `actionPerformed` method to Example 10-4.

Example 10-4 Add actionPerformed method

```
public void actionPerformed(ActionEvent e) {
   String searchSSN = getSearchSSN().getText();
   CustomerFactory factory = new CustomerFactory();
   try {
      Customer customer = factory.getCustomer(searchSSN);
      getCustomerName().setText(customer.getFullName());
      getCustomerSSN().setText(customer.getSsn());
      getAccountList().setListData(customer.getAccounts());
   } catch (SQLException e1) {
      getCustomerName().setText("");
      getCustomerSSN().setText("<SSN not found>");
      getAccountList().setListData(new Vector());
      e1.printStackTrace();
   }
}
```

- Add the appropriate import statements. Right-click in the source and select **Source** → **Organize Imports** or press Ctrl+Shift+O.
- Save the file.

Update the imported model class database location

The event handling code described in this section uses the back-end code we imported in “Importing the model classes for the sample” on page 406. We might have to modify the `DatabaseManager` class so that it points to the correct database.

- Open the **DatabaseManager** class from the Project Explorer (in **RAD7GUI** → **itso.rad7.model.entity**).
- Change the URL string for the database in the `getConnection` method to reflect the actual location of the database, for example:

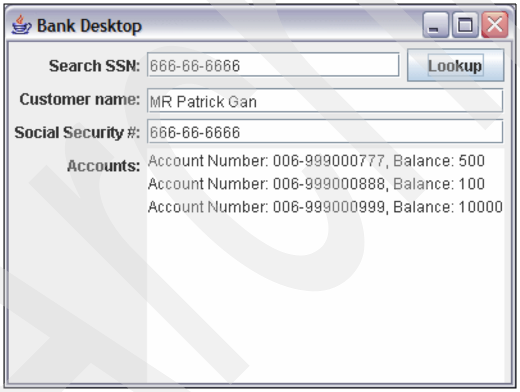
  ```
  jdbc:derby:C:\\7501code\\database\\derby\\ITSOBANK
  ```

- Save the file.

Verifying the Java GUI application

To verify that the Java GUI application is working properly, do these steps:

- Ensure that no connections to the `ITSOBANK` database are active. If they are, disconnect prior to running the Java GUI application.
- Click **Run** in the toolbar ▶. If you hover the mouse pointer over it you see that the tooltip text reads `Run BankDesktop`. Alternatively, right-click **`BankDesktop.java`** and select **Run As** → **Java Bean**.
- Enter a valid social security number into the Search SSN field. For example, enter **`666-66-6666`**.
- Select **Lookup** and the program searches the database for the requested value. The results are shown in Figure 10-13.



Search SSN: 666-66-6666

Customer name: MR Patrick Gan

Social Security #: 666-66-6666

Accounts: Account Number: 006-999000777, Balance: 500
Account Number: 006-999000888, Balance: 100
Account Number: 006-999000999, Balance: 10000

Enter **xxx-xx-xxxx**
as Search SSN where
x is any number

Figure 10-13 Testing the working GUI

- Try entering an invalid value. In the Console view, you see an error message for the invalid value entered as well as an exception stack trace:

  ```
  SQL Exception: Invalid cursor state - no current row.
  at org.apache.derby.impl.jdbc.Util.newEmbedSQLException(Unknown Source)
  ...... at
  itso.rad7.model.entity.CustomerFactory.getCustomer(CustomerFactory.java:14)
  at itso.rad7.gui.BankDesktop.actionPerformed(BankDesktop.java:52) ......
  ```

Run the sample GUI as a Java application

To run the sample GUI as a Java application, a main method is required. This method was generated when we created the class. It can also be added manually later (Example 10-5).

Example 10-5 BankDesktop generated main method

```
public static void main(String[] args) {
        // TODO Auto-generated method stub
        SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                BankDesktop thisClass = new BankDesktop();
                thisClass.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
                thisClass.setVisible(true);
            }
        });
    }
```

Run the `BankDesktop` as a Java application by selecting **Run** → **Run As** → **Java Application**. The results should be the same as before.

Automatically add event handling

The visual editor provides a mechanism for generating the code necessary for handling component events.

To demonstrate the automatic adding of event handling, we first remove the existing event handling code.

- Select all the code in the existing `actionPerformed` method and cut it (right-click and select **Cut**, or, Ctrl+X, or **Edit** → **Cut**).
- Delete the `actionPerformed` method from `BankDesktop` (this produces a syntax error, which we fix).
- Delete `implements ActionListener` from the class definition so that it reads:

  ```
  public class BankDesktop extends JFrame {
  ```

- Remove or comment the line from the `initialize` method that registers `BankDesktop` as a button listener.

  ```
  //this.getLookupButton().addActionListener(this);
  ```

- Save the class and all the errors disappear.
- In the Design view, right-click the **Lookup** JButton and select **Events** → **actionPerformed**.

This adds the `addActionListener` method call to the `getLookupButton` method with an anonymous inner class to implement the `ActionListener` interface. We must now add code to this inner class to make it work.

- Paste the code you previously cut from the `actionPerformed` method into this new `actionPerformed` method. The `getLookupButton` method is shown in Example 10-6.

Example 10-6 getLookupButton method with actionPerfomed

```
private JButton getLookupButton() {
   if (lookupButton == null) {
      lookupButton = new JButton();
      lookupButton.setText("Lookup");
      lookupButton.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent e) {
            System.out.println("actionPerformed()");
            String searchSSN = getSearchSSN().getText();
            CustomerFactory factory = new CustomerFactory();
            try {
               Customer customer = factory.getCustomer(searchSSN);
               getCustomerName().setText(customer.getFullName());
               getCustomerSSN().setText(customer.getSsn());
               getAccountList().setListData(customer.getAccounts());
            } catch (SQLException e1) {
               getCustomerName().setText("");
               getCustomerSSN().setText("<SSN not found>");
               getAccountList().setListData(new Vector());
               e1.printStackTrace();
            }
         }
      });
   }
   return lookupButton;
}
```

- Test the GUI as described in “Verifying the Java GUI application” on page 421.

visual editor binding

The visual editor can perform some automatic code generation for binding the properties of certain components to data source objects. The data source objects can be Enterprise JavaBeans (EJBs), Web services, or ordinary JavaBeans.

In the visual editor, Swing visual components (such as text fields, tables, and buttons) can be bound using the provided generic classes and interfaces.

Alternatively, you can also write your own binders for other components by implementing the provided interfaces. The visual editor provides dialog boxes that help you create data source data objects and data sources that you can use to bind a visual component. Or, you can manually add basic data objects, data source data objects, and data sources from the Palette and configure them using the Properties view.

To get a better feel, we will try this binding in the following example:

- Select the **customerName** text field.
- Click **Bind** above the field (Figure 10-14).

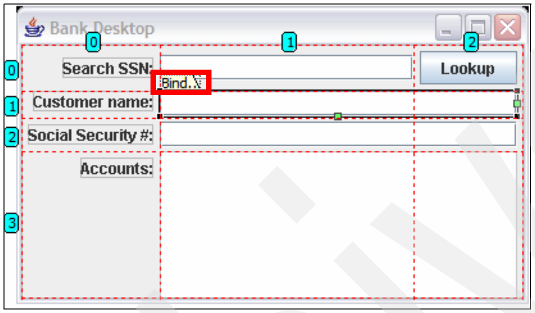
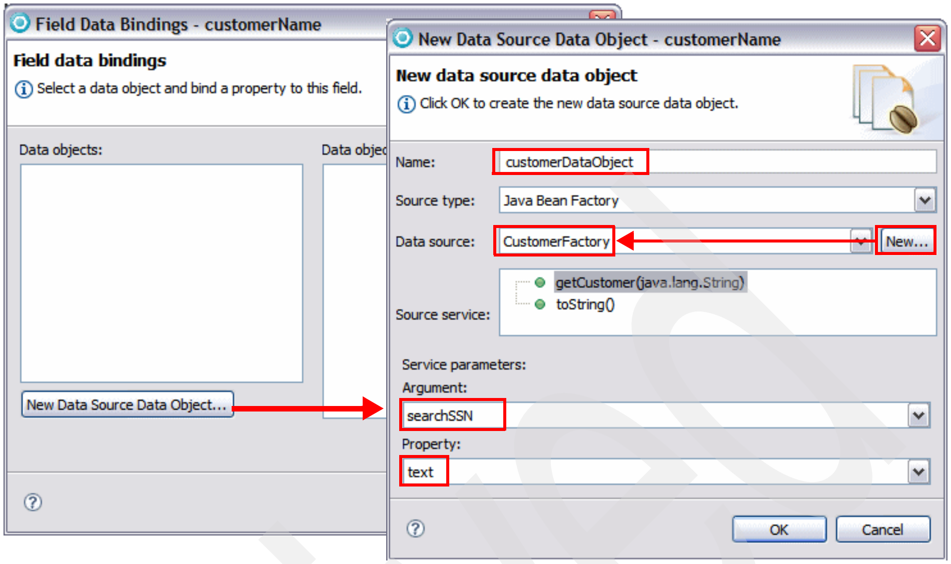


Figure 10-14 Binding a text field: Bind

- In the Find Data Bindings dialog, click **New Data Source Data Object**.
- In the New Data Source Data Object dialog (Figure 10-15), enter the following data:
  - Name: `customerDataObject`
  - Source type: Select **Java Bean Factory**.
  - Data source: Click **New** next to Data source.
  - Enter `CustomerFactory` into the search text field, and click **OK**.
  - Source service: Select **getCustomer**.
  - Argument: Select **searchSSN** (the text field).
  - Property: Select **text**.
  - Click **OK**.



Field Data Bindings - customerName

Field data bindings

Select a data object and bind a property to this field.

New Data Source Data Object - customerName

New data source data object

Click OK to create the new data source data object.

Name: customerDataObject

Source type: Java Bean Factory

Data source: CustomerFactory

Source service: getCustomer(java.lang.String), toString()

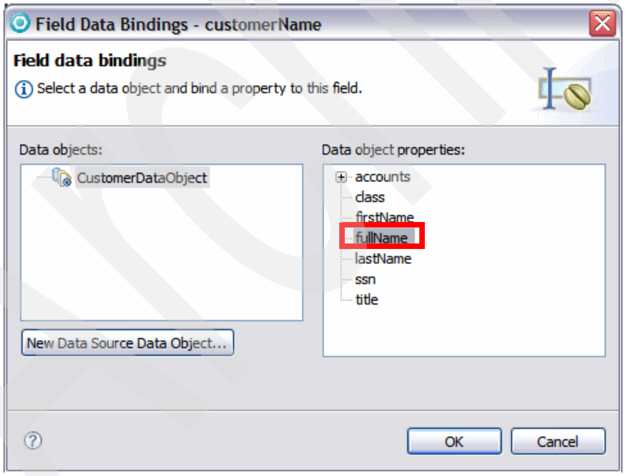
Service parameters:

Argument: searchSSN

Property: text

Figure 10-15 Binding a text field: New Data Source Object

- Select **fullName** from the Data object properties list (Figure 10-16) and click **OK**.



Field Data Bindings - customerName

Field data bindings

Select a data object and bind a property to this field.

Data objects: CustomerDataObject

Data object properties: accounts, class, firstName, fullName, lastName, ssn, title

Figure 10-16 Binding a text field: Property selection

- Repeat this for the `customerSSN` text field. This time the data object is already filled in and we select the `ssn` property.
- The binding appears in the Design view (Figure 10-17).

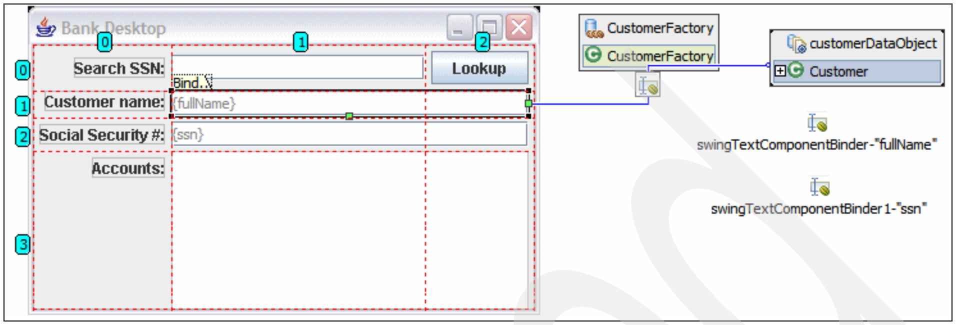


Figure 10-17 Binding a text field: Design view result

- Unfortunately, only `JTextField`, `JTable`, `JTextArea`, or `JButton` can be bound to data source objects in this way, so we cannot use this technique for the `JList`.
- A number of methods are generated in the source class:
  - `getCustomerDataObject`—This method is responsible for creating the data object.
  - `getCustomerFactory`—This method is responsible for creating the data source object.
  - `getSwingTextComponentBinderx` (one per field)—This method is responsible for creating the binding.
  - `getCustomerName`, `getCustomerSSN`—This method is responsible for adding the binding.
  - jve.generated—This package contains helper classes.
- The `getLookupButton` method can now be simplified as seen in Example 10-7.

Example 10-7 getJButton method modifications

```
private JButton getLookupButton() {
   if (lookupButton == null) {
      lookupButton = new JButton();
      lookupButton.setText("Lookup");
      lookupButton.addActionListener(new java.awt.event.ActionListener() {
         public void actionPerformed(java.awt.event.ActionEvent e) {
            System.out.println("actionPerformed()");
            String searchSSN = getSearchSSN().getText();
            CustomerFactory factory = new CustomerFactory();
            try {
               getCustomerDataObject().refresh();
               Customer customer = factory.getCustomer(searchSSN);
               //getCustomerName().setText(customer.getFullName());
               //getCustomerSSN().setText(customer.getSsn());
               getAccountList().setListData(customer.getAccounts());
            } catch (SQLException e1) {
               //getCustomerName().setText("");
               getCustomerSSN().setText("<SSN not found>");
               getAccountList().setListData(new Vector());
               e1.printStackTrace();
            }
         )
      });
   }
   return lookupButton;
}
```

The main change is highlighted in bold. The code required to set the text of the name and social security fields is no longer required—it is essentially replaced by the call to `getCustomerDataObject().refresh()`, which handles the synchronization for us.

Summary

In this chapter we demonstrated how to build a GUI application using the visual editor of Application Developer. We used a number of features, such as graphical layout, event handling generation, and data binding.

More information

For more detailed information on GUI development, refer to the following sites:

- Java Swing V1.4.2 API:

  http://java.sun.com/j2se/1.4.2/docs/guide/swing/

- Java AWT V1.4.2 API:

  http://java.sun.com/j2se/1.4.2/docs/guide/awt/

- Eclipse homepage:

  http://www.eclipse.org/platform/

11

Develop XML applications

This chapter introduces the XML capabilities provided by Rational Application Developer and describes how to use the XML tooling.

The chapter is organized into the following sections:

- XML overview and technologies
- Rational Application Developer XML tools
- Creating an XML schema
- Validating an XML schema
- Generating HTML documentation from an XML schema file
- Generating an XML file from an XML schema
- Editing an XML file
- Working with XSL files
- Transforming an XML file into an HTML file
- XML mapping
- Service Data Objects and XML

Note: The sample code described in this chapter can be completed by following along in the procedures documented. Alternatively, you can import the sample provided in the `c:\7501code\zInterchangeFiles\XML\RAD7XML.zip` project interchange file. For details refer to Appendix B, "Additional material" on page 1307.

XML overview and technologies

EXtensible Markup Language (XML) is a subset of Standard Generalized Markup Language (SGML). Both XML and SGML are meta languages, as they allow the user to define their own tags for elements and attributes.

XML is a key part of the software infrastructure. It provides a simple and flexible means of defining, creating, and storing data. XML is used for a variety of purposes ranging from configuration to messaging to data storage.

By allowing the user to define the structure of the XML document, the rules that define the structure can also be used to validate a document to ensure they conform.

XML and XML processor

XML is tag-based; however, XML tags are not predefined in XML. You have to define your own tags. XML uses a document type definition (DTD) or an XML schema definition (XSD) to describe the data.

XML documents follow strict syntax rules. To create, read, and update XML documents, you require an XML processor or parser. At the heart of an XML application is an XML processor that parses an XML document, so that the document elements can be retrieved and transformed into a format understood by the target client. The other responsibility of the parser is to check the syntax and structure of the XML document.

DTD and XML schema

Document Type Definitions (DTDs) and XML schemas are both used to describe structured information; however, in recent years the acceptance of XML schemas has gained momentum. Both DTDs and schemas are building blocks for XML documents and consist of elements, tags, attributes, and entities. Both define the rules by which an XML document must conform.

An XML schema is more powerful than DTD. Here are some advantages of XML schemas over DTDs:

- They define data types for elements and attributes, and their default and fixed values. The data types can be of string, decimal, integer, boolean, date, time, or duration.
- They apply restrictions to elements, by stating minimum and maximum values (for example, on age from 1 to 90 years), or restrictions of certain values (for

example, IBM Redbooks, residencies, redpieces with no other values accepted, such as in a drop-down list). Restrictions can also be applied to types of characters and their patterns (for example, only accepting values ‘a’ to ‘z’ and also specifying that only three letters can be accepted). The length of the data can also be specified (for example, passwords must be between 4 and 8 characters).

- They specify complex element types. Complex types can contain simple elements and other complex types. Restrictions can be applied to the sequence and the frequency of their occurrences. These complex types can then be used in other complex type elements.
- Because schemas are written in XML, they are also extensible. This also implies that the learning curve for learning another language has been eliminated. The available parsers do not have to be enhanced. Transformation can be carried out using eXtensible Style Language Transformation (XSLT), and its manipulation can be carried out using XML domain object model (DOM).

With XML schemas being extensible, they can be re-used in other schemas. Multiple schemas can be referenced from the same document. In addition, they have the ability to create new data types from standard data types.

XSL and XSLT

The eXtensible Style Language (XSL) is a language defined by the W3C for expressing style sheets.

XSL has the following three parts:

- XSL transformations (XSLT): Used for transforming XML documents
- XML path language (XPath): Language used to access or refer to parts of an XML document
- XSL-FO: Vocabulary for specifying formatting semantics

A transformation in XSLT must be a well-formed document and must conform to the namespaces in XML, which can contain elements that might or might not be defined by XSLT. XSLT-defined elements belong to a specific XML namespace. A transformation in XSLT is called a style sheet.

XSL uses an XML notation and works on two principles—pattern matching and templates. XSL operates on an XML source document and parses it into a source tree. It applies the transformation of the source tree to a result tree, and then it outputs the result tree to a specified format. In constructing the result tree, the elements can be reordered or filtered, and other structures can be added. The result tree can be completely different from the source tree.

XML namespaces

Namespaces are used when there is a requirement for elements and attributes of the same name to take on a different meaning depending on the context in which they are used. For instance, a tag called `<TITLE>` takes on a different meaning, depending on whether it is applied to a person or a book.

If both entities (a person and a book) have to be defined in the same document (for example, in a library entry that associates a book with its author), we require a mechanism to distinguish between the two and apply the correct semantic description to the `<TITLE>` tag whenever it is used in the document.

Namespaces provide the mechanism that allows us to write XML documents that contain information relevant to many software modules.

XPath

The XML path language (XPath) is used to address parts of an XML document. An XPath expression can be used to search through an XML document, and extract information from the nodes (any part of the document, such as an element or attribute) in it.

Rational Application Developer XML tools

Rational Application Developer provides a comprehensive visual XML development environment. The tool set includes components for building DTDs, XML schemas, XML, and XSL files.

Rational Application Developer includes the following XML development tools:

- DTD editor
- XML editor
- XML Schema editor
- XSL editor
- XPath Expression wizard
- XML to XML Mapping editor
- XSL debugging and transformation
- Generating Java beans from an XML schema
- Generating HTML documentation from an XML schema file
- Generating XML schema file from XML, DTD, and relational tables

Creating an XML schema

In this section, we create an XML schema for bank accounts. The schema defines a collection of accounts. Each account has an account ID, account type, account balance, interest rate, and related customer information. The account ID is 6 to 8 digits long and takes in numbers 0-9. The value of the interest rate is between 0 and 100. The phone number is in (xxx) xxx-xxxx format.

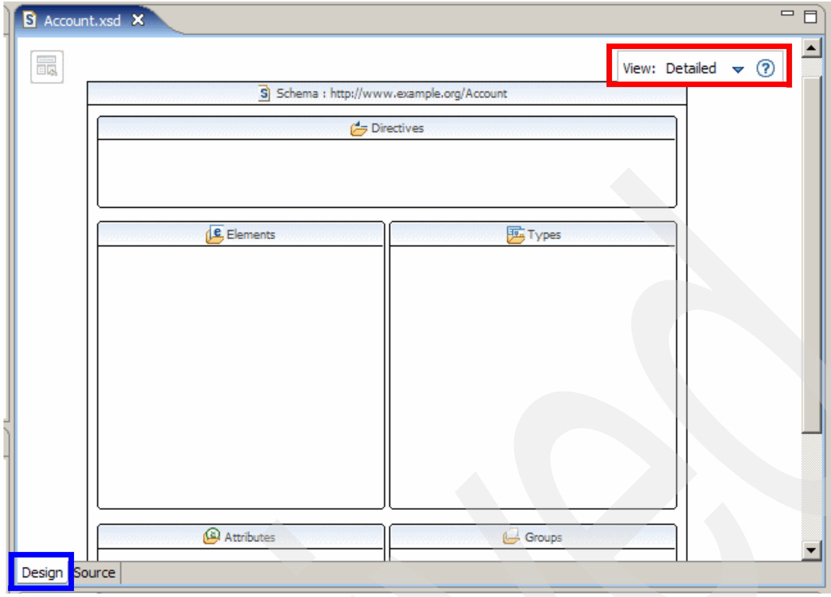
- Create a project named **RAD7XMLBank**:
  - Select **File** → **New** → **Project** → **General** → **Project**, then click **Next**.
  - In the Project name field type **RAD7XMLBank**.
  - Click **Finish** and the project is displayed in the Project Explorer view.
  - Right-click **RAD7XMLBank** → **New** → **Folder**, enter **xml** as the folder name, and click **Finish**.
- Create an XML schema:
  - Right-click the **xml** folder and select **New** → **Other** → **XML** → **XML Schema**. Click **Next**.
  - The parent folder is set to **RAD7XMLBank/xml**. In the File name field type **Accounts.xsd**.
  - Click **Finish**.
  - Make sure the Properties view is opened. If you cannot see the Properties view, select **Window** → **Show view** → **Properties**.

Working with the Design view

We create the schema in the Design view.

- Select the **Design** view. Notice the top-right corner that displays `View: Simplified`.
  - The Simplified view hides many of the complicated XML schema constructs so that you can create XML data structures that conform to best practice authoring patterns.
  - The Detailed view exposes the full set of XML schema constructs so you can create XML data structures using any authoring pattern. Complicated constructs such as choices, sequences, groups, and element references are not displayed in the Simplified view and actions in the editor that enable the creation of these constructs are not available.

  Because we have to create some complicated constructs, we use the Detailed view.
- Select **Detailed** from the View list (Figure 11-1).



View: Detailed

Figure 11-1 XSD editor

- Change the namespace prefix and target namespace:
  - In the Design view, select **Schema:http://www.example.org/Accounts**.
  - Go to the Properties view and change the Prefix to **itso**.
  - Change the Target namespace to **http://itso.rad7.xml.com**.
- The `Accounts.xsd` file has to contain a complex type for defining account information, which includes the account id, account type, balance, interest and customer information:
  - In the Design view, right-click the **Types** category and click **Add ComplexType**.
  - Overtype the name with **Account**.

> **Tip:** Alternatively, you can change the element name in the **Properties** view, General tab, Name field. The Properties view gives you lots of options to modify the properties of an XML schema.

- Right-click the **Account** complex type, and select **Add Sequence**.

Note: If you switch to the Simplified view, you would not be able to see the Add Sequence option in the context menu, as the Simplified view hides many of the complicated XML schema constructs such as sequences and actions in the editor that enable the creation of these constructs.

- Right-click the **Account** complex type again, and select **Add Element**.
- Now you are switched to the detailed view for complex type Account. Change the element name to **AccountId** (Figure 11-2).

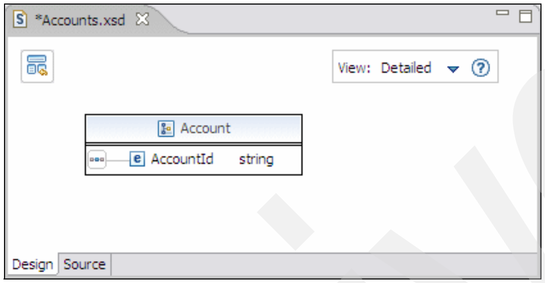
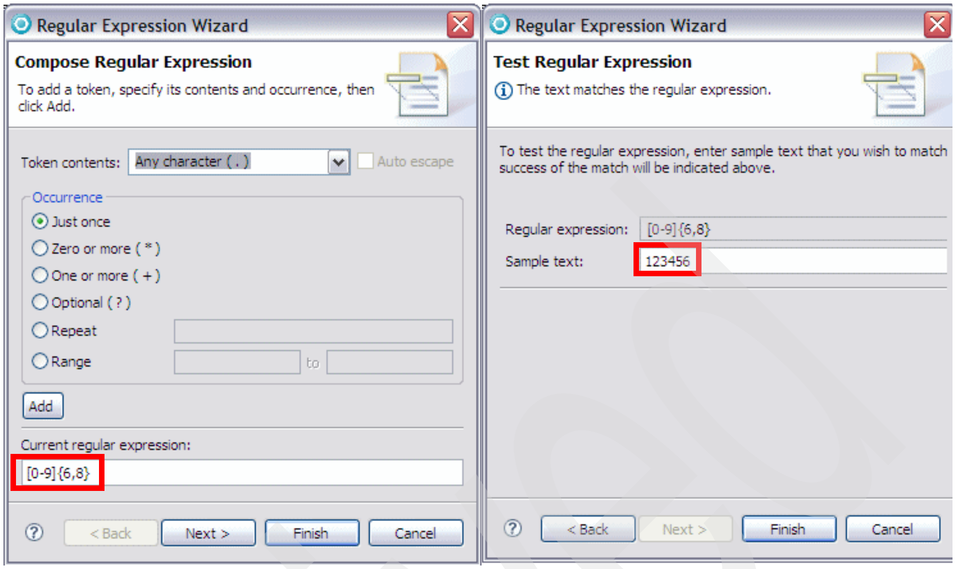


Figure 11-2 Account editor

- In our bank, the account ID is 6 to 8 digits long and takes in numbers 0-9:
  - Click the element **AccountId**. In the Properties view, select the **Constraints** tab.
  - In Specific constraint values section, select **Patterns**. Click **Add**.
  - In the Regular Expression wizard (Figure 11-3):
    - In the Current regular expression field enter **[0-9]{6,8}**.
    - Click **Next**.
    - In the Sample text area, enter **123456.**
    - Note after you enter a 6-digit number, the warning in the dialog box is gone.
    - Click **Finish**.



Compose Regular Expression

To add a token, specify its contents and occurrence, then click Add.

Token contents: Any character (.)

□ Auto escape

Occurrence

● Just once
○ Zero or more (*)
○ One or more (+)
○ Optional (?)
○ Repeat
○ Range to

Add

Current regular expression: [0-9]{6,8}

Test Regular Expression

The text matches the regular expression.

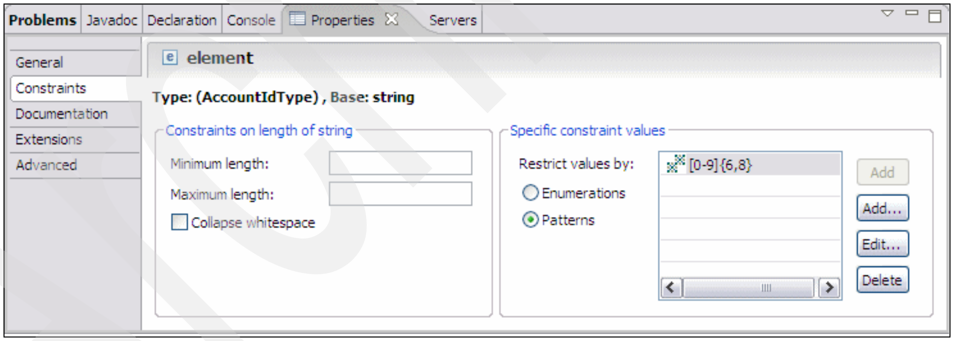
To test the regular expression, enter sample text that you wish to match success of the match will be indicated above.

Regular expression: [0-9]{6,8}

Sample text: 123456

Figure 11-3 Regular Expression wizard

- Note that the type changes from string to (AccountIdType).
- The Properties view is shown in Figure 11-4.



element

Type: (AccountIdType) , Base: string

Constraints on length of string

Minimum length:

Maximum length:

□ Collapse whitespace

Specific constraint values

Restrict values by: [0-9]{6,8}

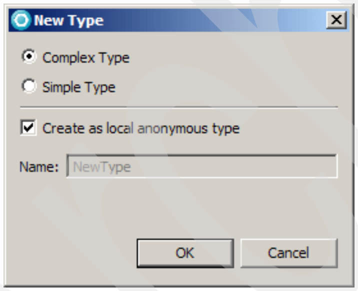
○ Enumerations
● Patterns

Figure 11-4 Specify constraint in Properties view

- In the Design view, right-click the sequence icon and select **Add Element**. Change its name to **AccountType**. The bank account has three account types: Savings, Loan, and Fixed:
  - In the Properties view, select the **Constraints** tab. In the Specific constraint values section, select **Enumerations.**
  - Click **Add** and enter **Savings**.

- Click **Add** and enter **Loan**.
- Click **Add** and enter **Fixed**.

- Add the balance element:
  - In the Design view, right-click the sequence icon and select **Add Element**. Change its name to **Balance**.
  - In the **Properties** view, **General** tab, select the drop-down menu for **Type**.
  - Select **Browse...** from the drop-down menu and set the type to **decimal**.
- Add the interest element:
  - Add another element named **Interest**, and set the type to **decimal**.
  - In the Properties view, **Constraints** tab, and set the **Minimum value** to **0** and **Maximum value** to **100**. Select **Inclusive** for the Minimum value.
- Add the CustomerInfo element:
  - Add another element named **CustomerInfo**.
  - In the Properties view, **General** tab, select the drop-down menu for the **Type**, and select **New**.
  - In the New Type dialog, select **Complex Type** and **Create as local anonymous type**, then click **OK** (Figure 11-5).



● Complex Type

○ Simple Type

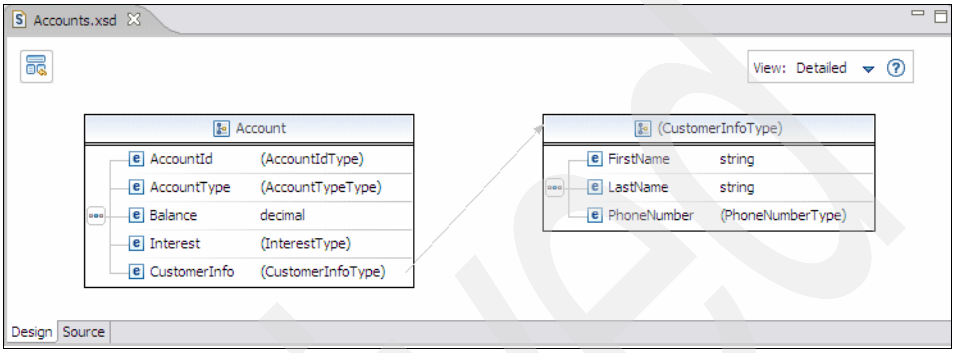
☑ Create as local anonymous type

Name: NewType

Figure 11-5 New Type dialog

- A new (CustomerInfoType) box appears on the right of the Account box.
- Right-click **(CustomerInfoType)** and select **Add Element**. Change the name to **FirstName**.
- Add another element named **LastName**.
- We create a phone number with a format such as (408) 456-7890:
  - Add another element named **PhoneNumber**.
  - Select the **PhoneNumber**.
  - In the Properties view, **Constraints** tab, select **Patterns**.

- Click **Add**. In the **Current regular expression** area, type **\([0-9]{3}\) [0-9]{3}-[0-9]{4}**, and click **Next**. Note that there is a space between area code and local phone number.
- Enter **(408) 456-7890** as the **Sample text** and click **Finish**. The Design view is shown in Figure 11-6. The Design view is printable, which is a new feature in Application Developer v7.



Account

AccountId	(AccountIdType)
AccountType	(AccountTypeType)
Balance	decimal
Interest	(InterestType)
CustomerInfo	(CustomerInfoType)

(CustomerInfoType)

FirstName	string
LastName	string
PhoneNumber	(PhoneNumberType)

Figure 11-6 Design view for Account type

- ► To create an instance document from this XML schema, the XML schema must have a global element. We add a global element named **Accounts** as follows:
  - Click the icon at the top left corner.
  - In the Design view of the schema, right-click the **Elements** category and select **Add Element**.
  - Change the name to **Accounts**.
  - In the Properties view, **General** tab, select the drop-down menu for **Type**, select **New...**, then select **complex type** and **Create as local anonymous type**, and click **OK**.
  - Double-click **Accounts** and you are switched to the detailed view for Accounts element. Right-click **(AccountsType)** and select **Add Element**.
  - Change the name to **Account**.
  - Right-click **Account** and select **Set Type** → **Browse** → **Account**.
  - In the Design view, make sure **Account** is selected. In the Properties view, General tab, set the **Minimum Occurrence** to **1** and **Maximum Occurrence** to **unbounded**.
  - Select **Source** → **Format** → **Document** to format the XSD file.
  - Save and close the file.

Source view

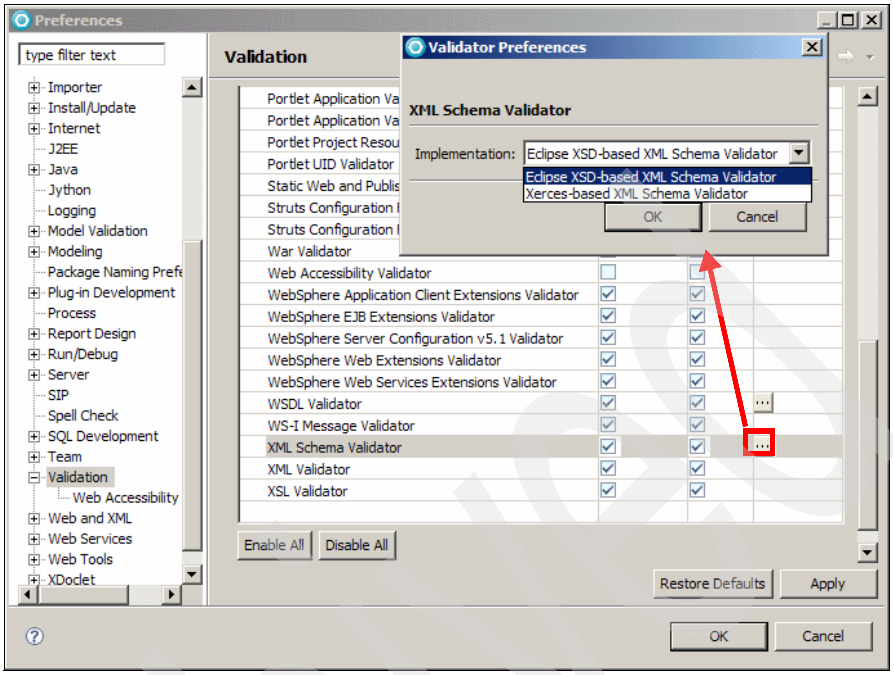
The generated `Accounts.xsd` file is listed in Example 11-1.

Example 11-1 Accounts.xsd file

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://itso.rad7.xml.com"
   elementFormDefault="qualified" xmlns="http://www.w3.org/2001/XMLSchema"
   xmlns:itso="http://itso.rad7.xml.com">

   <complexType name="Account">
      <sequence>
         <element name="AccountId">
            <simpleType>
               <restriction base="string">
                  <pattern value="[0-9]{6,8}"></pattern>
               </restriction>
            </simpleType>
         </element>
         <element name="AccountType">
            <simpleType>
               <restriction base="string">
                  <enumeration value="Savings"></enumeration>
                  <enumeration value="Loan"></enumeration>
                  <enumeration value="Fixed"></enumeration>
               </restriction>
            </simpleType>
         </element>
         <element name="Balance" type="decimal"></element>
         <element name="Interest">
            <simpleType>
               <restriction base="decimal">
                  <minExclusive value="0"></minExclusive>
                  <maxExclusive value="100"></maxExclusive>
               </restriction>
            </simpleType>
         </element>
         <element name="CustomerInfo">
            <complexType>
               <sequence>
                  <element name="FirstName" type="string"></element>
                  <element name="LastName" type="string"></element>
                  <element name="PhoneNumber">
                     <simpleType>
                        <restriction base="string">
                           <pattern
                              value="\([0-9]{3}\) [0-9]{3}-[0-9]{4}">
                           </pattern>
                        </restriction>
                     </simpleType>
                  </element>
               </sequence>
            </complexType>
         </element>
      </sequence>
   </complexType>

   <element name="Accounts">
      <complexType>
         <sequence>
            <element name="Account" type="itso:Account"
               minOccurs="1" maxOccurs="unbounded">
            </element>
         </sequence>
      </complexType>
   </element>
</schema>
```

Validating an XML schema

In Application Developer v7, two validators are available: Eclipse XSD-based XML schema validator and Xerces-based XML schema validator. The Eclipse XSD-based XML validator is faster when validating large schemas and is set as the default. You can check the setting in **Window** → **Preferences** → **Validation** → **XML Schema Validator** (Figure 11-7):



Implementation: Eclipse XSD-based XML Schema Validator
- Eclipse XSD-based XML Schema Validator
- Xerces-based XML Schema Validator

Portlet Application Va			
Portlet Application Va			
Portlet Project Resou			
Portlet UID Validator			
Static Web and Publis			
Struts Configuration I			
Struts Configuration I			
War Validator			
Web Accessibility Validator	□	□	
WebSphere Application Client Extensions Validator	☑	☑	
WebSphere EJB Extensions Validator	☑	☑	
WebSphere Server Configuration v5.1 Validator	☑	☑	
WebSphere Web Extensions Validator	☑	☑	
WebSphere Web Services Extensions Validator	☑	☑	
WSDL Validator	☑	☑	...
WS-I Message Validator	☑	☑	
XML Schema Validator	☑	☑	...
XML Validator	☑	☑	
XSL Validator	☑	☑	

Figure 11-7 XML Schema Validator

In some cases, when you build the large J2EE projects, the XSD validation process takes some time. You can disable the validator either at the project level, or, at the global level:

- To disable the validator at the project level, select the project and **Properties** → **Validation**, then select **Override validation preferences** and clear the XML Schema Validator.
- To disable the validator at the global level, use the **Window** → **Preferences** dialog (Figure 11-7) and clear the XML Schema Validator.

Run schema validation manually

The validation builder is not added to the simple projects, such as our project. To validate your XML schema, complete the following steps:

- Right-click **Account.xsd** in the Project Explorer and select **Validate**.
- If validation is not successful, the validation warning or error is displayed in the Problems view.
- If validation is successful, you do not receive a validation confirmation box as seen in Application Developer v6. This behavior might change in future versions of Application Developer.

You should not receive any XML schema validation errors or warnings against `Accounts.xsd`.

To see an error, for example, change a type from decimal to **deximal**, and run the validation. Note after correcting the error you have to run validation again to remove the error.

Generating HTML documentation from an XML schema file

HTML documentation generated from an XML schema contains various information about the schema such as its name, location, and namespace, as well as details about various components (such as elements or complex types) contained in the schema. It can be useful for providing a summary of the contents of your schema.

To generate HTML documentation based on an XML schema file:

- In the Project Explorer, right-click the **Accounts.xsd** and select **Generate** → **HTML Doc**. The Generate XML schema documentation dialog opens.
- Select **Multiple HTML file with frames**. If you select this option, several HTML files are generated instead of just one. They will be set up in a similar fashion to Javadoc files.
- Click **Next** to specify the target folder for your HTML files.
- Accept the default folder **/RAD7XMLBank/Accounts/docs**, then click **Finish**.

The HTML files are created in the location you specified. The generated `index.html` file is opened in a Web browser inside Application Developer. Explore the generated documentation by selecting the Account type and its elements.

Generating an XML file from an XML schema

To generate an XML file from the XML schema file, follow these steps:

- In the Project Explorer, right-click the **Accounts.xsd** and select **Generate** → **XML File**.
- Select **RAD7XMLbank/xml** as the parent folder and accept the default name **Accounts.xml**. Click **Next**.
- The XML schema you created earlier does not have optional attributes or elements. Accept the default values in the Select Root Element page. Click **Finish**.
- The XML file opens in the editor.

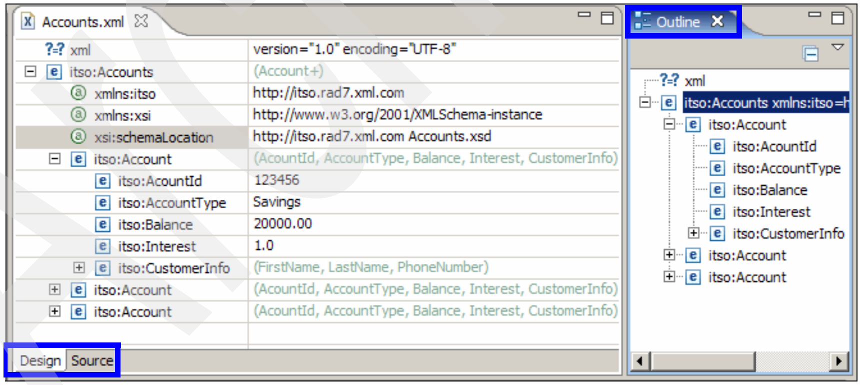
- Right-click the generated XML file **Account.xml** → **Validate**. You will notice the validation errors against AccountId.
- The XML schema defines the account id is 6 to 8 digits long and takes in numbers 0-9. Change the account id to 123456.
- The XML schema defines the phone number in (xxx) xxx-xxxx format. Change the phone number to (123) 456-7890.
- Optionally change the balance, interest, first and last name.
- Right-click **Accounts.xml** → **Validate**. The validation errors are gone.

Editing an XML file

The XML editor enables you to directly edit XML files. There are several different views you can use to edit your files:

- **Source view**—You can manually insert, edit, and delete elements and attributes in the Source view of the XML editor. To facilitate this effort, you can use content assist.
- **Design view**—You can insert, delete, and edit elements, attributes, comments, and processing instructions in this view.
- **Outline view**—You can insert and delete elements attributes, comments, and processing instructions in this view.

These three views are shown in Figure 11-8.



?=? xml	version="1.0" encoding="UTF-8"
⊟ e itso:Accounts	(Account+)
ⓐ xmlns:itso	http://itso.rad7.xml.com
ⓐ xmlns:xsi	http://www.w3.org/2001/XMLSchema-instance
ⓐ xsi:schemaLocation	http://itso.rad7.xml.com Accounts.xsd
⊟ e itso:Account	(AcountId, AccountType, Balance, Interest, CustomerInfo)
e itso:AcountId	123456
e itso:AccountType	Savings
e itso:Balance	20000.00
e itso:Interest	1.0
⊞ e itso:CustomerInfo	(FirstName, LastName, PhoneNumber)
⊞ e itso:Account	(AcountId, AccountType, Balance, Interest, CustomerInfo)
⊞ e itso:Account	(AcountId, AccountType, Balance, Interest, CustomerInfo)

```
?=? xml
⊟ e itso:Accounts xmlns:itso=h
  ⊟ e itso:Account
      e itso:AcountId
      e itso:AccountType
      e itso:Balance
      e itso:Interest
    ⊞ e itso:CustomerInfo
  ⊞ e itso:Account
  ⊞ e itso:Account
```

Figure 11-8 Design view, Source view and Outline view

Edit in the Source view

We will continue to work on the XML file we generated in the last section:

- In the Source view, place your cursor after the closing tag **</itso:Account>**.

- Press **Ctrl+Space** to activate code assist. A pop-up list of available choices, which is based on the context, is displayed.
- Double-click **<itso:Account>**.
- While the cursor is still between **<itso:Account>** tags, press **Ctrl+Space** and double-click **<itso:AccountId>**.
- Type a number for AccountId between the start and end tags.
- Repeat the same procedure to use the code assist feature to input the rest of the information, such as AccountType (`Loan`), Balance, and so forth.
- After you finish typing, you can **right-click** in the XML source area and select **Format** → **Document**.

Edit in the Design view

- In the Design view, right-click **itso:Accounts** → **Add Child** → **itso:Account**.
- Expand the Account element you just created. All the child elements are created for you. You can edit the values of the child elements.

Edit in the Outline view

- In the Outline view, right-click **itso:Accounts**. You can see a similar context menu as in the Design view.
- Save and close the file.

Working with XSL files

An XSL file is a style sheet that can be used to transform XML documents into other document types and to format the output. In this section, we create a simple XSL style sheet to format the XML file data into a table in an HTML file.

Create a new XSL file

To create a sample XSL file, do these steps:

- Select the **xml** folder and **New** → **Other** → **XML** → **XSL**. Click **Next**.
- In the File name field type **Accounts.xsl**.
- In the Select XML file page select the **Accounts.xml** file. This associates the `Accounts.xml` file with the `Accounts.xsl` file.
- Click **Finish**.

The generated XSL file is listed in Example 11-2.

Example 11-2 Generated Accounts.xsl

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0"
    xmlns:xalan="http://xml.apache.org/xslt">
</xsl:stylesheet>
```

Edit an XSL file

The XSL editor provides a number of wizards to help you create content in your style sheet. We use the two wizards in this example:

- The HTML header wizard, which creates a template that will generate an `<html>` header together with the appropriate `<xsl:output>` element.
- The XSL Table wizard, which adds HTML tables to the XSL file.

To add code snippets to the XSL file, do these steps:

- In the XSL editor position the **cursor** between the **<xsl:stylesheet>** tags, right after `xmlns:itso="http://itso.rad7.xml.com"`>.
- Select the **Snippets** view and select the **XSL drawer** as seen in Figure 11-9.

  If you cannot see the Snippets view, select **Window** → **Show view** → **Other** → **General** → **Snippets**. Move the Snippets view to any pane in the tool.

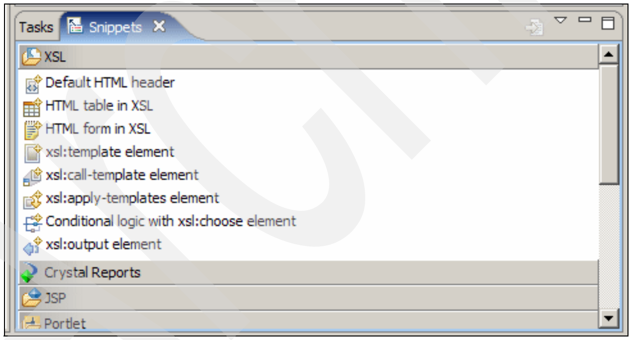
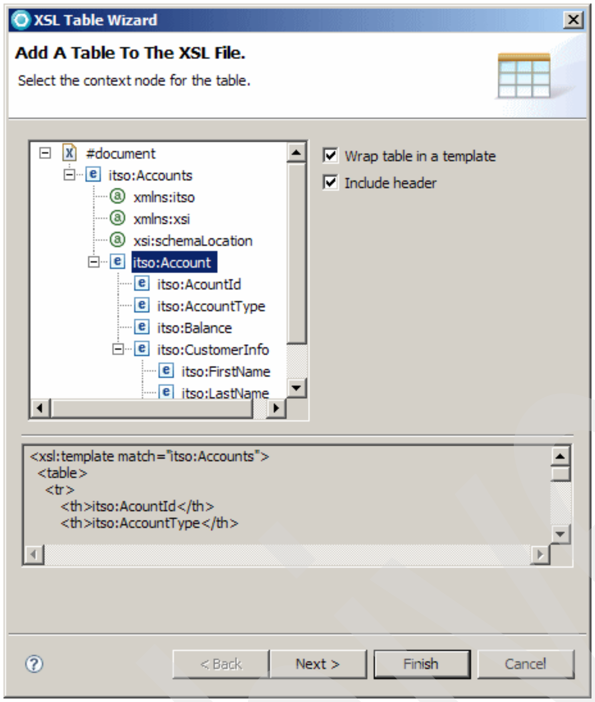


Figure 11-9 Snippets view

- Double-click **Default HTML header**. This adds the default HTML header information into the XSL file.
- Position the **cursor** after the end tag **</xsl:template>**.
- In the XSL drawer double-click **HTML table in XSL**.
- In the XSL Table wizard (Figure 11-10):



Add A Table To The XSL File.

Select the context node for the table.

☑ Wrap table in a template

☑ Include header

```
<xsl:template match="itso:Accounts">
 <table>
  <tr>
    <th>itso:AcountId</th>
    <th>itso:AccountType</th>
```

Figure 11-10 XSL Table wizard

- – Select the **itso:Account** element as the context node.
- – Select **Wrap table in a template**.
- – Select **Include header** to indicate that you want to include a header in the table.
- – Click **Next**.

- ► Type **5** for the **Border** field and 10 for the **Cell spacing** field.
- ► Click **Finish**.
- ► Right-click in the XSL source area and select **Format** → **Document**.
- ► Remove the `itso:` prefix from the values in the table header fields, for example, `<th>~~itso:~~AccountId</th>`.
- ► Save and close the file.
- ► Right-click **Accounts.xsl** → **Validate**. You should not receive any validation errors or warnings.
- ► The generated `Accounts.xsl` file is listed in Example 11-3.

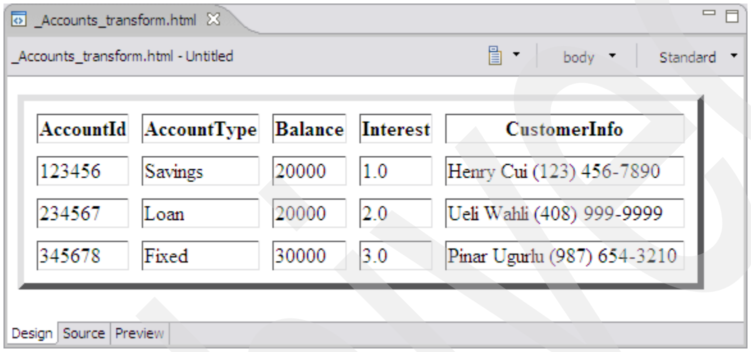
Example 11-3   Accounts.xsl file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   version="1.0" xmlns:xalan="http://xml.apache.org/xslt"
   xmlns:itso="http://itso.rad7.xml.com">
   <xsl:output method="html" encoding="UTF-8" />
   <xsl:template match="/">
      <html>
         <head>
            <title>Untitled</title>
         </head>
         <body>
            <xsl:apply-templates />
         </body>
      </html>
   </xsl:template>
   <xsl:template match="itso:Accounts">
      <table border="5" cellspacing="10">
         <tr>
            <th>AccountId</th>
            <th>AccountType</th>
            <th>Balance</th>
            <th>Interest</th>
            <th>CustomerInfo</th>
         </tr>
         <xsl:for-each select="/itso:Accounts/itso:Account">
            <tr>
               <td>
                  <xsl:value-of select="itso:AccountId" />
               </td>
               <td>
                  <xsl:value-of select="itso:AccountType" />
               </td>
               <td>
                  <xsl:value-of select="itso:Balance" />
               </td>
               <td>
                  <xsl:value-of select="itso:Interest" />
               </td>
               <td>
                  <xsl:value-of select="itso:CustomerInfo" />
               </td>
            </tr>
         </xsl:for-each>
      </table>
   </xsl:template>
</xsl:stylesheet>
```

Transforming an XML file into an HTML file

We can now use the XSL file to generate HTML from the sample XML file:

- In the Project Explorer, hold down the **Ctrl key** and select both the **Accounts.xml** and **Accounts.xsl** files.
- Right-click and select **Run As** → **XSL Transformation**.
- The resulting file name is **_Accounts_transform.html**. It automatically opens in the **Design** view of the **Page Designer** (Figure 11-11).
- Notice the transformation messages in the Console view.



AccountId	AccountType	Balance	Interest	CustomerInfo
123456	Savings	20000	1.0	Henry Cui (123) 456-7890
234567	Loan	20000	2.0	Ueli Wahli (408) 999-9999
345678	Fixed	30000	3.0	Pinar Ugurlu (987) 654-3210

Figure 11-11 XSL transformation result

XML mapping

The XML Mapping editor is a visual data mapping tool that is designed to transform any combination of XML schema, DTD, or XML documents and produce a deployable transformation document. You can map XML-based documents graphically by connecting elements from a source document to elements of a target document. You can extend built-in transformation functions using custom XPath expressions and XSLT templates. The mapping tool automates XSL code generation and produces a transformation document based on the mapping information you provide.

You can select one of the following options to transform between the nodes (Table 11-1).

Table 11-1 Options to transform between the nodes

Option	Description
Move	This type copies data from a source to a target.
Concat	This type creates a string concatenation that allows you to retrieve data from two or more entities to link them into a single result.
Inline map	This type enables the map to call out to other maps, but other maps cannot call it. It can only be used within the current map. If the inputs and outputs are arrays, the inline map is implicitly iterated over the inputs.
Submap	This type references another map. It calls or invokes a map from this or another map file. Choosing this transform type is most effective for reuse purposes.
Substring	This type extracts information as required. For example, the substring lastname, firstname with a "," delimiter and a substring index of 0 returns the value lastname. If the substring index was changed to 1 the output would now be firstname.
Group	This type takes an array or collection of data and groups them into a collection of a collection. Essentially, it is an array containing an array. Grouping is done at the field level, meaning that it is done by selecting a field of the input collection such as "department."
Normalize	This type normalizes the input string. For example, it can be used to remove multiple occurrences of white space (such as space, tab, or return).
Custom	This type allows you to enter custom code or call reference code to be used in the transform. You can extend built-in transformation functions using custom XPath expressions and XSLT templates.
Assign	Assign a constant value to an output element (right-click the output element and select **Create transform**). Set the value in the Properties view (use single quotes for strings).

In this section, we show you an example on mapping two XML schemas using move, concat, inline map, substring. and custom options.

Preparation and import

To prepare for the XML mapping we import two XSD files and one XML file:

- `Accounts.xsd`—Same file that we created previously (the source schema)
- `AccountsList.xsd`—Alternate representations of accounts (target schema)
- `Accounts.xml`—Sample data (similar to what we created earlier)

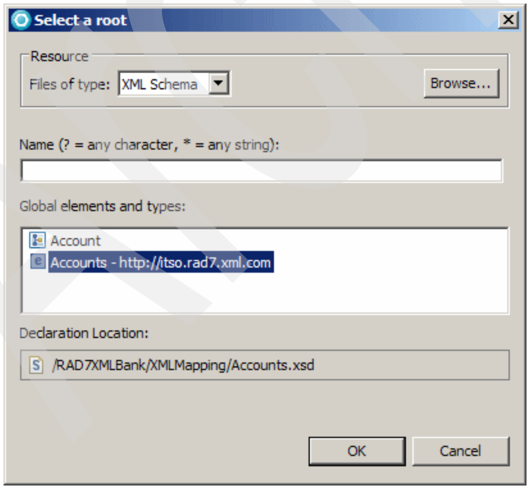
To import the files, do these steps:

- Create a folder named `mapping` in the RAD7XMLBank project by right-clicking the **RAD7XMLBank** project and **New** → **Folder**. Type **mapping** as the folder name and click **Finish**.
- Right-click the **mapping** folder and select **Import** → **General** → **File System** and click **Next**. Click **Browse** to navigate to the `c:\7501code\xml` folder. Click **OK**. Select **AccountList.xsd**, **Accounts.xml** and **Accounts.xsd**. Click **Finish**.

Launch the XML Mapping editor

The XML Mapping editor is used to create a mapping between the two XML schemas:

- To launch the XML Mapping editor, select **File** → **New** → **Other** → **XML** → **XML Mapping**. Click **Next**.
- For the parent folder use `RAD7XMLBank/mapping`.
- In the File name field type **Accounts.map**, and click **Next**.
- For Root inputs, click **Add**:
  - Select **XML Schema** from the **Files of type** drop-down list, and click **Browse**. Expand **RAD7XMLBank** → **mapping** → **Accounts.xsd** and click **OK**.
  - Select the **Accounts** element from the Global elements and types list. Click **OK** (Figure 11-12).



Files of type: XML Schema

Name (? = any character, * = any string):

Global elements and types:

Account

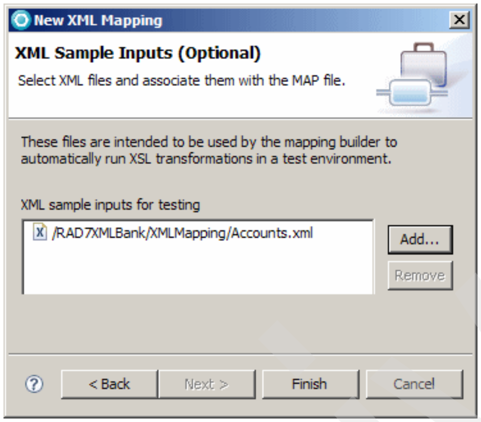
Accounts - http://itso.rad7.xml.com

Declaration Location:

/RAD7XMLBank/XMLMapping/Accounts.xsd

Figure 11-12 Select a root

- For Root outputs, click **Add** and select the AccountsList.xsd file and the Accounts element (similar to inputs).
- Click **Next**.
- To select a sample XML input file, click **Add**. Select **Accounts.xml** and click **OK** (Figure 11-13).



New XML Mapping

XML Sample Inputs (Optional)

Select XML files and associate them with the MAP file.

These files are intended to be used by the mapping builder to automatically run XSL transformations in a test environment.

XML sample inputs for testing

/RAD7XMLBank/XMLMapping/Accounts.xml

Figure 11-13 XML Sample Inputs

- Click **Finish** and the XML Mapping editor opens.

In the Project Explorer, three new files are generated:

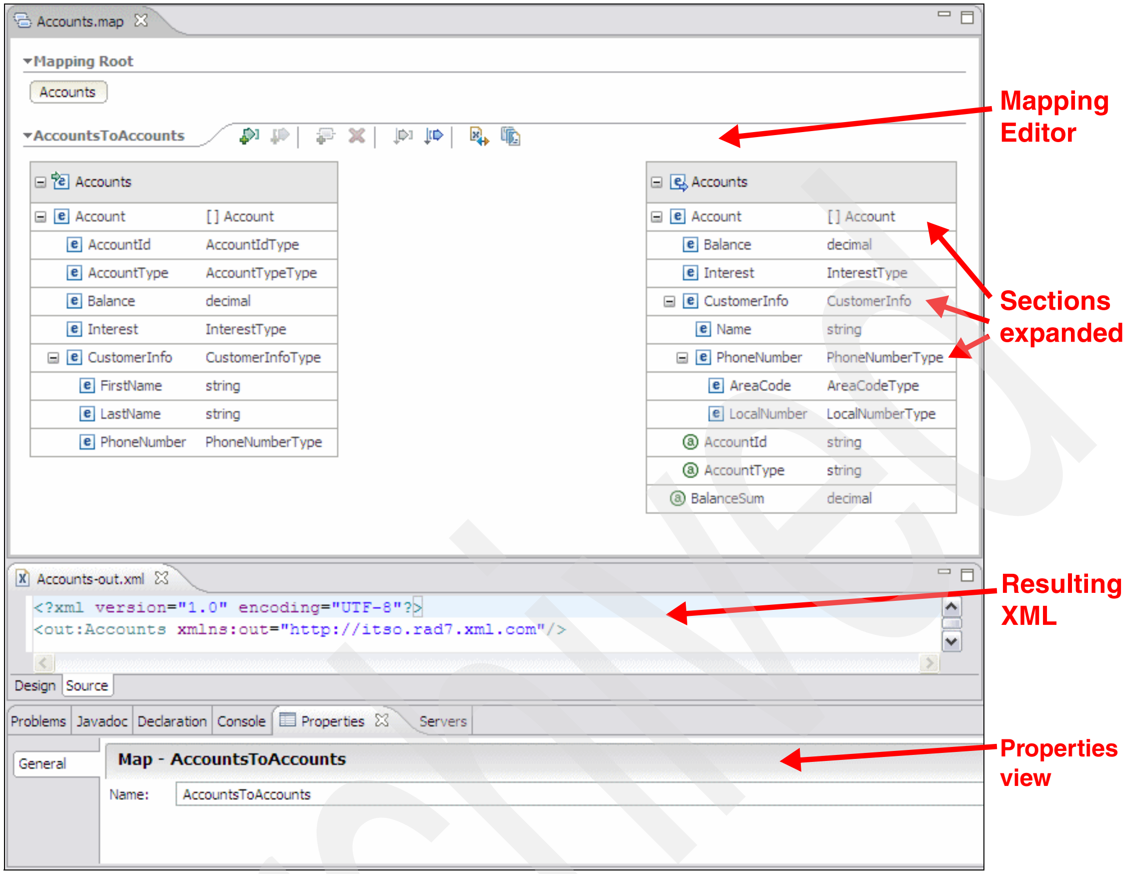
- `Accounts.xsl`: XSL transformation file
- `Accounts-out.xml`: Resulting XML file
- `Accounts.map`: Mapping file

Organizing the XML Mapping editor

Part of the beauty of this tool is that you can see what kind of changes you make in the resulting xml when you are doing the mapping.

Before we start editing the mapping, we do these steps:

- Open **Accounts-out.xml**, then drag the editor panel down to the bottom part of the mapping file so that it is under `Accounts.map`.
- Select the Properties view.
- In the Workbench layout, the mapping file is at the top. The resulting xml file is in the middle, and the Properties view is at the bottom (Figure 11-14).



```
<?xml version="1.0" encoding="UTF-8"?>
<out:Accounts xmlns:out="http://itso.rad7.xml.com"/>
```

Map - AccountsToAccounts

Name: AccountsToAccounts

Mapping Editor

Sections expanded

Resulting XML

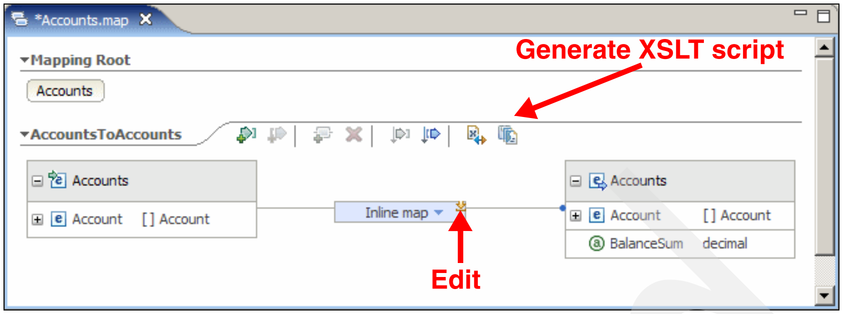
Properties view

Figure 11-14 Workbench layout for the XML Mapping editor

Editing the XML mapping

We now create the mapping between the two XML schemas:

- In the XML Mapping editor, select the **Account** element from `Accounts.xsd` (left hand side) and drag it to the **Account** element in `AccountList.xsd` (right hand side), as shown in Figure 11-15.



Generate XSLT script

Edit

Figure 11-15 Inline map

- Click **Generate XSLT script.** See Figure 11-15 to find this icon. You immediately notice that the **Accounts-out.xml** is changed.

> **Note:** Alternatively, you can simply save the mapping file and the changes are automatically reflected in the resulting xml file.

Inline mapping

- Click **Edit** at the top right corner of the **Inline map**. See Figure 11-15 to find the Edit icon.
- In the Inline map details view, perform the following mapping transforms:
  - Map the **AccountId**, **AccountType**, **Balance**, and **Interest** by dragging the elements from the left to the corresponding elements on the right. Note that we map the element AccountId to the attribute AccountId.
  - Click **Generate XSLT script** and verify how the account information is generated in the XML output.
  - Map the **CustomerInfo** element from left to right. This creates an inline map. The current mapping is shown in Figure 11-16.

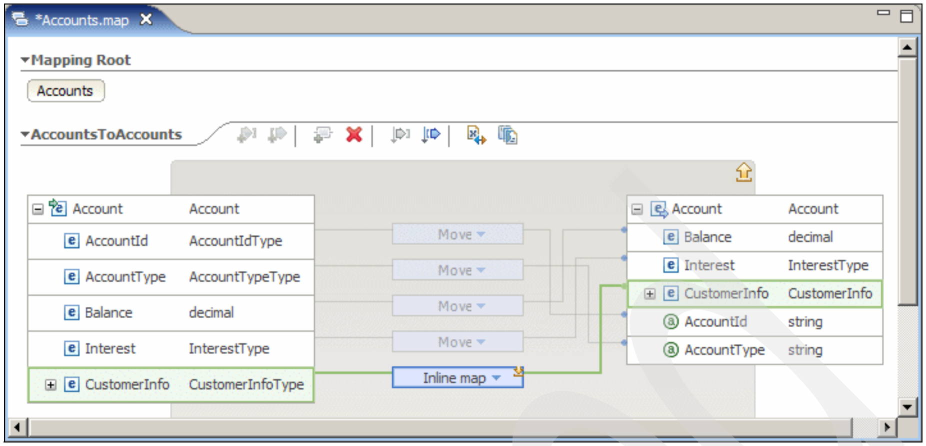


Figure 11-16 Account mapping

- ► Click **Generate XSLT script** to see the change in the output XMI file.

Concatenation mapping

We want to concatenate last name and firstname into one element:

- ► Click **Edit** at the top right corner of the CustomerInfo Inline map and add these transformations:
  - – Select the **FirstName** element and drag it to the **Name** element on the right.
  - – Select the **LastName** element and drag it to the **Move transform box** between the Firstname and Name. When you drag a second element to the transform type box, the transform type automatically changes to **Concat** (Figure 11-17).

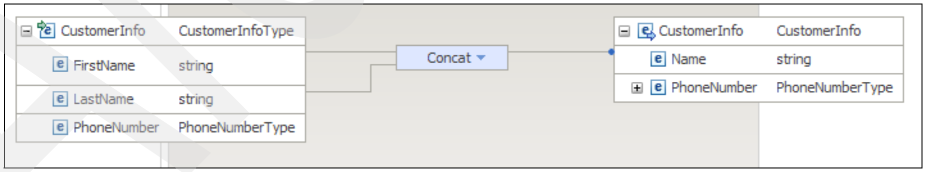

CustomerInfo CustomerInfoType
FirstName string
LastName string
PhoneNumber PhoneNumberType
Concat
CustomerInfo CustomerInfo
Name string
PhoneNumber PhoneNumberType

Figure 11-17 Concatenation

- ► We want to display the name in `LastName, FirstName` format. Do these steps:
  - – Select the **Concat transform**, and in the **Properties** view, select the **Order** tab.
  - – Select **LastName** and click the **Reorder Up** icon ⇧.

- Select General tab, select **LastName,** and put **‘, ’** in the **Delimiter** area.
- Click **Generate XSLT script icon** to see the change in **Accounts-out.xml**

Substring mapping

The phone number is stored as a single data type in the source document and we want to separate it into the sub-elements of area code and local number in the target document.

- Do the following steps to extract substrings:
  - Select the **PhoneNumber** element on the left and drag it to the **AreaCode** element on the right.
  - Click the **drop-down arrow** in the transformation and select **Substring** from the list.
  - Right-click the transformation and select **Show in Properties**.
  - In the Properties view, select the **General** tab. In the Delimiter field put a space. Because the phone number format is (xxx) xxx-xxxx, the space should be the delimiter between area code and local number.
  - In the **Substring index** field, type 0.
  - Select the **PhoneNumber** element and drag it to the **LocalNumber** element.
  - Change the transform type to **Substring**.
  - In the Properties view in the Delimiter field, type a space.
  - In the Substring index field, type 1. The current mapping is shown in Figure 11-18.

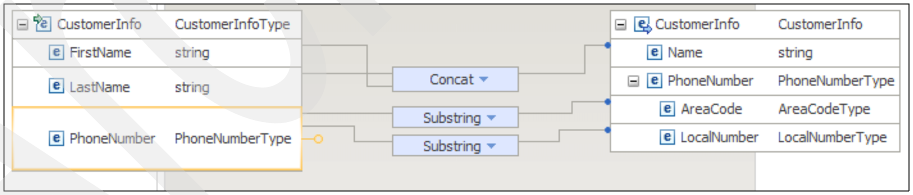


Figure 11-18 Substring

- To return to the main map, click the **Up a level** icon at the top right of the inline map details page. Do it twice to return to the main map.

Calculation

We want to calculate the sum of the balance from all accounts and put it in the `BalanceSum` attribute of the output document. We use an XPath expression to calculate the total.

- Select the **Accounts** element on the left and drag it to the **BalanceSum** attribute on the right.
- Click the **transform type box**, and select **Custom** (Figure 11-19).

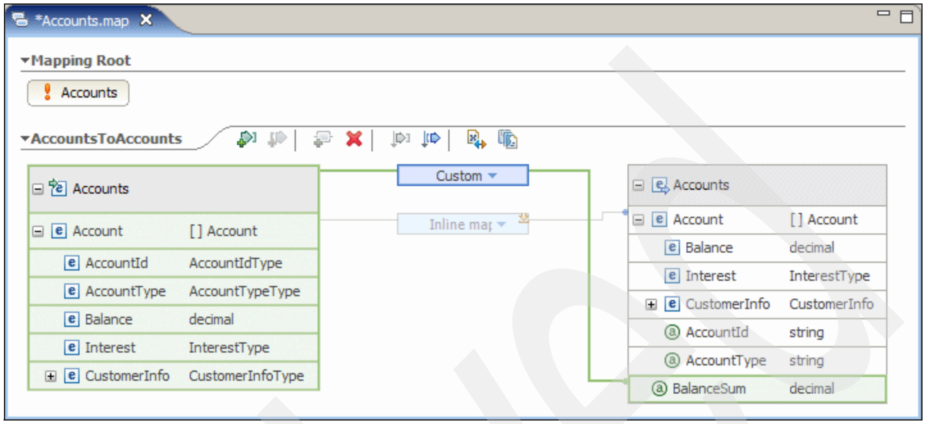
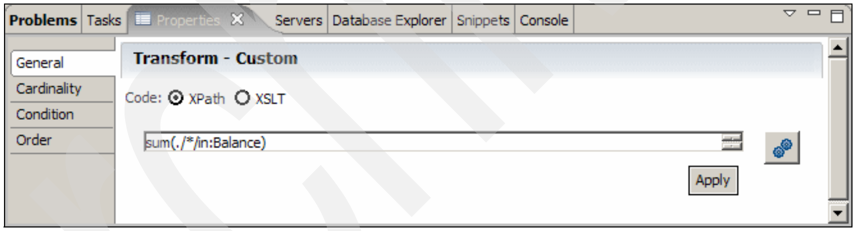


Figure 11-19 Custom mapping

- In the Properties view, General tab, select **XPath** as the **Code** and type **sum(./*/in:Balance)** as the XPath expression (Figure 11-20).



Transform - Custom

Code: ● XPath ○ XSLT

sum(./*/in:Balance)

Figure 11-20 Custom mapping using XPath

Save the mapping file and click **Generate XSLT script** to see the final output XML file. Notice the `BalanceSum` attribute in the `<Accounts>` tag and the formatting of an account:

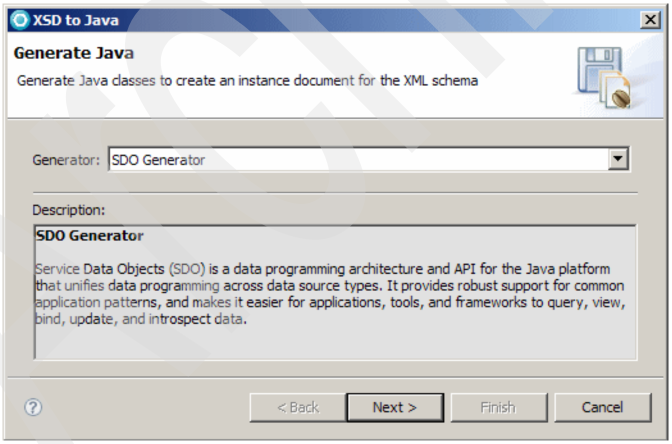
```
<out:Accounts xmlns:out="http://itso.rad7.xml.com" BalanceSum="70000">
  <out:Account AccountId="123456" AccountType="Savings">
    <out:Balance>20000</out:Balance>
    <out:Interest>1.0</out:Interest>
    <out:CustomerInfo>
      <out:Name>Cui, Henry</out:Name>
      <out:PhoneNumber>
        <out:AreaCode>(123)</out:AreaCode>
        <out:LocalNumber>456-7890</out:LocalNumber>
      </out:PhoneNumber>
    </out:CustomerInfo>
  </out:Account>
```

Generating JavaBeans from an XML schema

To allow developers to quickly build an XML application, the XML schema editor supports the generation of beans from an XML schema. Using these beans, you can quickly create an instance document or load an instance document that conforms to the XML schema.

To generate beans from an XML schema, do these steps:

- Create a Java project to contain the beans
  - Select **File** → **New** → **Other** → **Java** → **Java Project**. Click **Next**.
  - Type **RAD7XMLBankJava** in the Project name field and click **Finish**.
  - If you are prompted to switch to the Java perspective, click **Yes**.
- Generate the Java Beans:
  - In the Project Explorer expand project **RAD7XMLBank** → **xml** → **Accounts.xsd** and select **Generate** → **Java**.
  - When the Generate Java dialog is displayed (Figure 11-21), select the **SDO Generator** and click **Next**.


XSD to Java
Generate Java
Generate Java classes to create an instance document for the XML schema
< Back
Next >
Finish
Cancel

Generator: SDO Generator

Description:

SDO Generator

Service Data Objects (SDO) is a data programming architecture and API for the Java platform that unifies data programming across data source types. It provides robust support for common application patterns, and makes it easier for applications, tools, and frameworks to query, view, bind, update, and introspect data.

Figure 11-21 Java SDO classes generator

  - For the **Container** field, click **Browse** and select **/RAD7XMLBankJava**. Click **Finish**.

- Expand the RAD7XMLBankJava project and study the generated packages:
  - com.xml.rad7.itso
  - com.xml.rad7.itso.impl
  - com.xml.rad7.itso.util

Use the generated Java beans

To test the generated beans we create a test class named AccountsTest. This class creates an instance of the Accounts object and serialize the instance into XML format:

- Create a new Java class to test the generated java beans
  - In the Project Explorer, right-click **RAD7XMLBankJava** → **New** → **Class**.
  - Type **com.xml.rad7.itso.sdo** as the **package** name, enter **AccountsTest** as the **class** name, and click **Finish**.
- Complete the class with the sample code shown in Example 11-4. You can find the Accountstest.java file in the **c:\7501code\xml** folder.

  Study the main method first, then study the helper methods that are called from the main method (marked in bold). The call to save on the class ItsoResourceUtil serializes the SDO object tree and outputs the string to the supplied file name or output stream class instance.

Example 11-4 AccountTest program

```
package com.xml.rad7.itso.sdo;

import java.math.BigDecimal;
import com.xml.rad7.itso.*;
import com.xml.rad7.itso.util.ItsoResourceUtil;

public class AccountsTest {
   private DocumentRoot createDocumentRoot() {
      DocumentRoot documentRoot = ItsoFactory.eINSTANCE.createDocumentRoot();
      return documentRoot;
   }

   private AccountsType createAccountsType() {
      AccountsType accountsType = ItsoFactory.eINSTANCE.createAccountsType();
      return accountsType;
   }

   private Account createAccount(AccountTypeType accountType,
         String accountId, BigDecimal balance, BigDecimal interest,
         CustomerInfoType customerInfo) {
      Account account = ItsoFactory.eINSTANCE.createAccount();
      account.setAccountType(accountType);
      account.setAccountId(accountId);
      account.setBalance(balance);
      account.setInterest(interest);
      account.setCustomerInfo(customerInfo);
      return account;
   }

   private CustomerInfoType createAccountInfo(String firstName,
         String lastName, String phoneNumber) {
      CustomerInfoType customerInfo =
            ItsoFactory.eINSTANCE.createCustomerInfoType();
      customerInfo.setFirstName(firstName);
      customerInfo.setLastName(lastName);
      customerInfo.setPhoneNumber(phoneNumber);
      return customerInfo;
   }

   public static void main(String args[]) throws Exception {

      AccountsTest sample = new AccountsTest();
      DocumentRoot documentRoot = sample.createDocumentRoot();

      AccountsType accountsType = sample.createAccountsType();
      CustomerInfoType customerInfo = sample.createAccountInfo
            ("Henry", "Cui", "(123) 456-7890");
      Account account = sample.createAccount(AccountTypeType.SAVINGS_LITERAL,
            "123456", new BigDecimal(20000.00), new BigDecimal(3.5),
            customerInfo);
      accountsType.getAccount().add(0, account);
      customerInfo = sample.createAccountInfo
            ("Ueli", "Wahli", "(408) 345-6789");
      account = sample.createAccount(AccountTypeType.FIXED_LITERAL, "222222",
            new BigDecimal(50000.00), new BigDecimal(6.0), customerInfo);
      accountsType.getAccount().add(1, account);
      documentRoot.setAccounts(accountsType);

      ItsoResourceUtil.getInstance().save(documentRoot, System.out);
      ItsoResourceUtil.getInstance().save(documentRoot, "accounts.xml");
   }
}
```

Run the sample

To run the class as a Java application, do these steps:

- Right-click on **AccountTest.java** from the Package Explorer and select **Run As** → **Java Application**. The XML result is displayed in the Console view and generated as a serialized file `accounts.xml`.

- In the Package Explorer, right-click the project **RAD7XMLBankJava** and select **Refresh**. You can see and open the generated `accounts.xml` file:

```
<?xml version="1.0" encoding="UTF-8"?>
<itso:Accounts xmlns:itso="http://itso.rad7.xml.com">
  <itso:Account>
    <itso:AccountId>123456</itso:AccountId>
    <itso:AccountType>Savings</itso:AccountType>
    <itso:Balance>20000</itso:Balance>
    <itso:Interest>3.5</itso:Interest>
    <itso:CustomerInfo>
      <itso:FirstName>Henry</itso:FirstName>
      <itso:LastName>Cui</itso:LastName>
      <itso:PhoneNumber>(123) 456-7890</itso:PhoneNumber>
    </itso:CustomerInfo>
  </itso:Account>
```

Service Data Objects and XML

Service Data Objects is a framework for data application development, which includes an architecture and API. Service Data Objects simplify the J2EE data programming model and abstract data in a service oriented architecture (SOA). SDO unifies data application development, supports, and integrates XML. Service Data Objects incorporate J2EE patterns and best practices. SDO can be used to provide uniform access from a variety of data sources.

The core concepts in the SDO architecture are the data object and data graph:

- A data object holds a set of named properties, each of which contains either a primitive-type value or a reference to another data object. The data object API provides a dynamic data API for manipulating these properties.
- The data graph provides an envelope for data objects, and is the normal unit of transport between components. Data graphs also have the responsibility to track changes made to the graph of data objects, including inserts, deletes, and modification to data object properties.
- Data graphs are typically constructed from data sources, such as XML files, EJBs, XML databases, relational databases, or from services, such as Web services, JCA Resource Adapters, JMS messages, and so forth. Components that can populate data graphs from data sources and commit changes to data graphs back to the data source are called data mediator services (DMS). DMS architecture and APIs are outside the scope of the SDO specification.

In this section, we describe how to use SDO to access XML documents.

Load SDO from XML

In this example we load the `accounts.xml` file and print it out:

- Create a new Java class to use SDO to access XML document.
  - In the Project Explorer, right-click **RAD7XMLBankJava** → **New** → **Class**.
  - Select **com.xml.rad7.itso.sdo** as the package and type **SDOSample** as the class name.
  - Select **public static void main(String[] args]** to generate a main method.
  - Click **Finish**.
- In the Project Explorer, right-click **RAD7XMLBankJava** → **Properties** → **Java Build Path** → **Libraries**. Click **Add External JARs** and add **org.eclipse.emf.ecore.change_2.2.1.v200705141058.jar**, which is located in the installation **<SDP70Shared>/plugins** folder, where **<SDP70Shared>** is the directory where the shared components of the Rational Software Delivery Platform were placed on the initial installation. Click **OK**.

Note: `org.eclipse.emf.ecore.change_2.2.1.v200705141058.jar` is available in Application Developer 7.0.0.3. If you are on a different version of Application Developer v7, the version of this JAR might be slightly different. In theory, this JAR should be added to the classpath of the file automatically, however in Application Developer 7.0.0.3 this needs to be added manually.

This jar is used by our application code. If you do not add this JAR to the project build path, you will get `java.lang.NoClassDefFoundError: org/eclipse/emf/ecore/change/ChangeDescription`.

- Add the sample code shown in Example 11-5 to the Java main method:

Example 11-5 Load XML document

```
public static void main(String[] args) throws IOException {
   System.out.println("\n--- Printing an XML document to System.out ---");
   DocumentRoot documentRoot =
            ItsoResourceUtil.getInstance().load("accounts.xml");
   ItsoResourceUtil.getInstance().save(documentRoot, System.out);
   System.out.println("\n\n--- Done ---");
}
```

- Select **Source** → **Organize Imports** to add import statements.
- Right-click **SDOSample** → **Run As** → **Java Application**. The `accounts.xml` is shown in the java console.

Navigate the SDO

We use XPath expressions to obtain data from the SDO after loading the XML file.

Figure 11-22 shows the accounts data graph of the XML file.

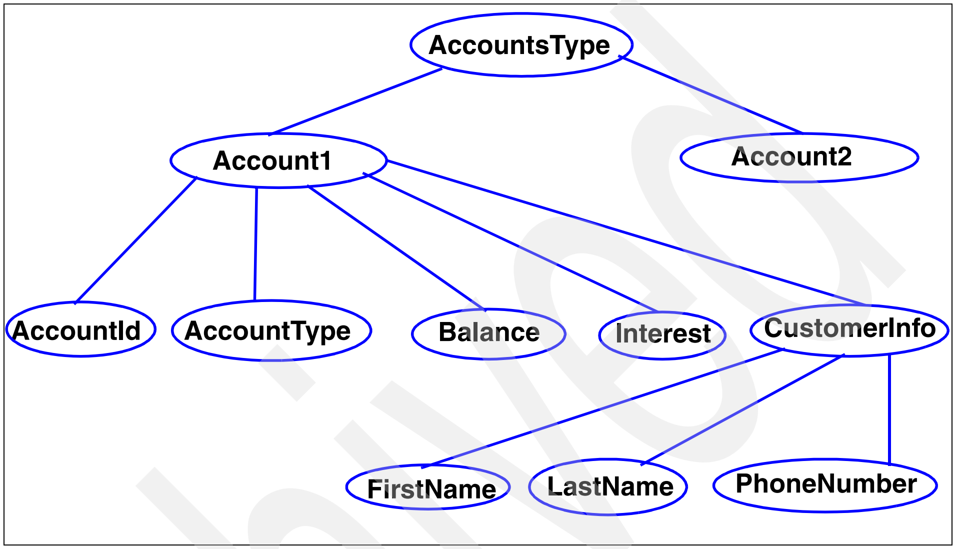


Figure 11-22 Accounts data graph

- Add the code shown in Example 11-6 after the sample code in Example 11-5:

Example 11-6 Navigate the SDO

```
AccountsType accountsType = documentRoot.getAccounts();
DataObject accountsTypeImpl = (AccountsTypeImpl) accountsType;
DataObject account1 = accountsTypeImpl.getDataObject("Account.0");
System.out.println("\n\nThe first account is: " + account1 + "\n");
DataObject account2 = accountsTypeImpl.getDataObject
                          ("Account[accountId = '222222']");
System.out.println("The second account is: " + account2 + "\n");
DataObject account1CustomerInfo = accountsTypeImpl.getDataObject
                          ("Account[accountId = '123456']/CustomerInfo");
System.out.println("The first account customer information: " +
                          account1CustomerInfo + "\n");
String account1CustomerName = (String)account1.get("CustomerInfo/FirstName");
System.out.println("The first account's customer's first name is " +
                          account1CustomerName + "\n");
```

  – The dot notation indexes data objects starting at 0, therefore, **Account.0** returns the first account data object.

  - **Account[accountId = '222222']** returns the account data object whose account id equals 222222.
  - **Account[accountId = '123456']/CustomerInfo** is an XPath expression to show how to return a data object multiple levels below the root data object. This XPath expression returns the customer info of an account whose account id is 123456.

- Right-click **SDOSample** → **Run As** → **Java Application**.

  You should see the following console output for the above code snippet:

```
The first account is: com.xml.rad7.itso.impl.AccountImpl@5ebc5ebc
(accountId: 123456, accountType: Savings, balance: 20000, interest: 3.5)

The second account is: com.xml.rad7.itso.impl.AccountImpl@36ce36ce
(accountId: 222222, accountType: Fixed, balance: 50000, interest: 6)

The first account customer information:
com.xml.rad7.itso.impl.CustomerInfoTypeImpl@34ea34ea
(firstName: Henry, lastName: Cui, phoneNumber: (123) 456-7890)

The first account's customer's first name is Henry
```

- Notice that the element names are displayed in lower case (`accountId`, `firstname`) although the XSD and the XML file define the element names in upper case (`AccountId`, `FirstName`).

Update the SDO

In this example we update the SDO after retrieval. We update the interest rate of one account, add an account, and delete an existing account:

- Add the code shown in Example 11-7 after the sample code in Example 11-6:

Example 11-7 Update a data object

```
account1.setString("interest", "10");

DataObject account3 = accountsTypeImpl.createDataObject("Account");
account3.setString("accountId", "333333");
account3.set("accountType", AccountTypeType.LOAN_LITERAL);
account3.setString("balance", "999999");
account3.setString("interest", "2.5");
DataObject newCustomerInfo = account3.createDataObject("customerInfo");
newCustomerInfo.setString("firstName", "Mike");
newCustomerInfo.setString("lastName", "Smith");
newCustomerInfo.setString("phoneNumber", "(201) 654-8754");

account2.delete();

System.out.println("\n--- Printing the updated XML document ---");
```

```
    ItsoResourceUtil.getInstance().save(documentRoot, System.out);
```

- The complete code listing of SDOSample.java can be found in the **c:\7501code\xml** folder.
- Right-click **SDOSample** → **Run As** → **Java Application**.

  You can see that the interest rate for the first account (`accountId = '123456'`) is updated. The second account (`accountId = '222222'`) is removed and a new account (`accountId = '333333'`) is added.

More information

For more information on XML schemas, refer to:

`http://www.w3.org/XML/Schema`

For more information on XML, refer to:

`http://www.w3.org/XML/`

For more information on XML parsers, refer to:

- Xerces (XML parser - Apache):

  `http://xml.apache.org/xerces2-j`
- Xalan (XSLT processor - Apache):

  `http://xml.apache.org/xalan-j`
- JAXP (XML parser - Sun):

  `http://java.sun.com/xml/jaxp`
- SAX2 (XML API):

  `http://sax.sourceforge.net`

For more information on SDO, refer to:

`http://www-128.ibm.com/developerworks/java/library/j-sdo/`
`http://www.osoa.org/display/Main/SDO+Resources`

12

Develop Web applications using JSPs and servlets

This chapter focuses on developing Web applications using JavaServer Pages (JSPs), J2EE servlet technology, and static HTML pages, which are fundamental technologies for building J2EE Web applications. Through the RedBank example, the chapter guides you through the features available in Rational Application Developer V7.0 to work with these technologies.

In the first section, we describe the main tools available in Application Developer to Web developers and introduce the new features provided with V7.0. Next we present the design of the ITSO RedBank application, then we build and test the application using the various tools available within Application Developer. In the final section, we list sources of further information on J2EE Web components and the Web tools within Application Developer.

The chapter is organized into the following sections:

- Introduction to J2EE Web applications
- Web development tooling
- Summary of new features in V7.0
- RedBank application design
- Implementing the RedBank application
- Testing the Web application
- More information

Introduction to J2EE Web applications

J2EE (Java 2 Enterprise Edition) is an application development framework that is the most popular standard for building and deploying Web applications in Java. Two of the key underlying technologies for building the Web components of J2EE applications are servets and JSPs. Servlets are Java classes that provide the entry point to logic for handling a Web request and return a Java representation of the Web response. JSPs are a mechanism to combine HTML with logic written in Java. Once compiled and deployed, JSPs run as a servlet, where they also take a Web request and return a Java object representing the response page.

Typically, in a large project, the JSPs and servlets are part of the presentation layer of the application and include logic to invoke the higher level business methods. The core business functions are usually separated out into a clearly defined set of interfaces, so that these components can be used and changed independently of the presentation layer (or layers, when there is more than one interface).

Enterprise JavaBeans (EJBs) are also a key feature included in the J2EE framework and are one popular option to implement the business logic of an application. These are described in detail in Chapter 16, “Develop Web applications using EJBs” on page 719. The separation of the presentation logic, business logic, and the logic to combine them is referred to as the model-view-controller pattern and is described later.

Technologies such as Struts, JavaServer Faces (JSF), various JSP tag libraries and numerous others have been developed to extend the JSP and servlets framework in different ways to improve aspects of J2EE Web development (for example, JSF facilitates the construction of reusable UI components which can be added to JSP pages). Some of these are described in detail in other chapters of this book, however, it is important to note that the underlying technologies of these tools are extensions to Java servlets and JSPs.

When planning a new project, the choice of technology depends on several criteria (size or project, previous implementation patterns, maturity of technology, skills of the team, and so on) and using JSPs with servlets and HTML is a comparatively simple option for building J2EE Web applications. Figure 12-1 show the relationship between J2EE, Enterprise Application, Web applications, EJBs, servlets, JSPs and additions such as Struts and JSF.

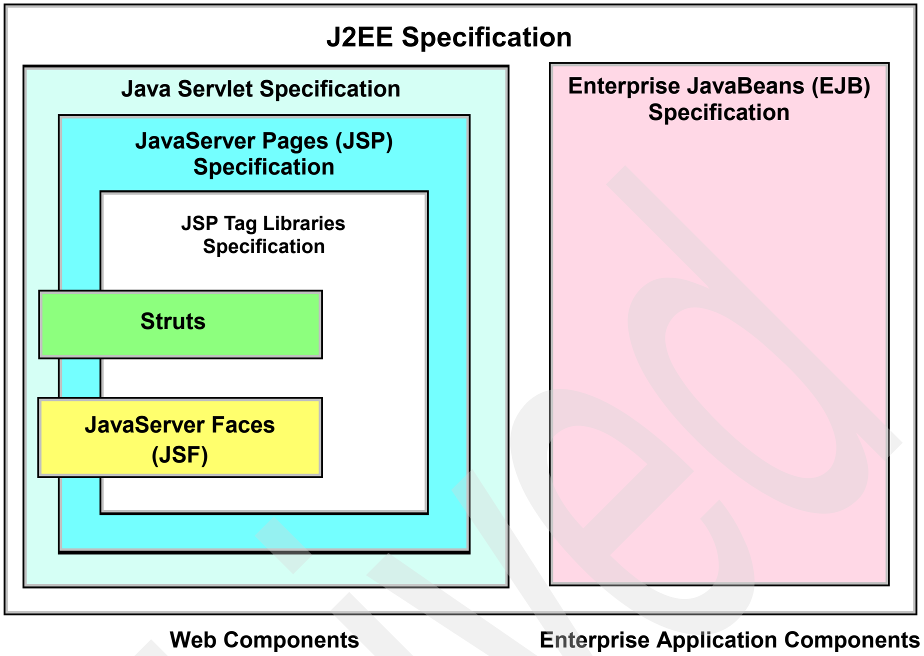

J2EE Specification
Java Servlet Specification
JavaServer Pages (JSP) Specification
JSP Tag Libraries Specification
Struts
JavaServer Faces (JSF)
Enterprise JavaBeans (EJB) Specification
Web Components
Enterprise Application Components

Figure 12-1 J2EE related technologies

Our focus in this chapter is on developing Web applications using JSPs, servlets, and static pages using HTML with the tools included with Application Developer. Once these concepts are mastered, it should be easier to understand the other technologies available.

J2EE applications

At the highest level, the J2EE specification describes the construction of two application types that can be deployed on any J2EE compliant application server. These are Web applications represented by a Web Application Archive (WAR) file or an enterprise application represented by an Enterprise Application Archive (EAR) file. Both files are constructed in zip file format, with a defined directory and file structure. Web applications generally contain the Web components required to build the information presented to the end user and some lower level logic, while the enterprise application contains an entire application including the presentation logic and logic implementing its interactions with an underlying database or other back-end system.

Also note that an EAR file can include one or more WAR files where the logic within the Web applications (WARs) usually invokes the application logic in the EAR.

Enterprise applications

An enterprise application project contains the hierarchy of resources that are required to deploy an enterprise (J2EE) application to WebSphere Application Server. It can contain a combination of Web applications (WAR files), EJB modules, Java libraries, and application client modules (all stored in JAR format). They also must include a deployment descriptor (`applicaton.xml` within the `META-INF` directory), which contains meta information to guide the installation and execution of the application.

The JAR files within an enterprise application can be used by the other contained modules. This allows sharing of code at the application level by multiple Web or EJB modules.

On deployment, the EAR file is unwrapped by the application server and the individual components (EJB modules, WAR files, and associated JAR files) are deployed individually, however, some aspects are configured across the enterprise application as a whole including, for example, shared JAR files.

The use of EJBs is not compulsory within an enterprise application. When developing an enterprise application (or even a Web application) the developer can write whatever Java is most appropriate for the situation. EJBs are the defined standard within J2EE for implementing application logic, but many factors can determine the decision for implementing this part of a solution. In the RedBank sample application presented later in this chapter, the business logic is implemented using standard Java classes that use HashMaps to store data.

Web applications

A Web application server publishes the contents of a WAR file under a defined URL root (called a context root) and then directs Web requests to the right resources and returns the appropriate Web response to the requestor. Some requests can be simply mapped to a simple static resource (such as HTML files and images) while others are mapped to a specific JSP or servlet class, which are referred to as dynamic resources. It is through these requests that the Java logic for a Web application is initiated and calls to the main business logic are processed.

When a Web request is received, the application server looks at the context root of the URL to identify which WAR the request is intended for, then the server looks at the contents after the root to identify which resource to send the request to. This might be a static resource (html file), the contents of which are simply returned, or a dynamic resource (servlet or JSP), where the processing of the request is handed over to JSP or servlet code.

In every WAR file there is descriptive meta information that describes this information and guides the application server in its deployment and execution of the servlets and JSPs within the Web application.

The structure of these elements within the WAR file is standardized and compatible between different Web application servers. The J2EE specification defines the hierarchical structure for the contents of a Web application that can be used for deployment and packaging purposes. All J2EE compliant servlet containers, including the test Web environment provided by Rational Application Developer, support this structure.

The structure of a WAR file (and an EAR file) is shown in Figure 12-2.

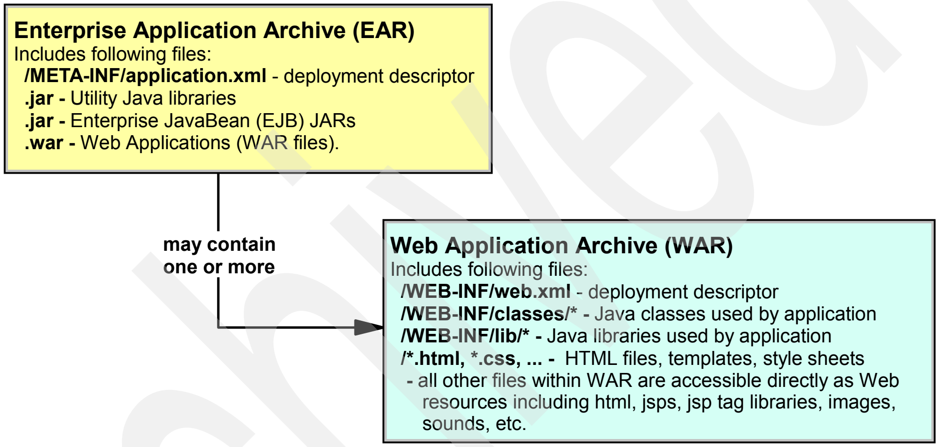

Enterprise Application Archive (EAR)
Includes following files:
/META-INF/application.xml - deployment descriptor
.jar - Utility Java libraries
.jar - Enterprise JavaBean (EJB) JARs
.war - Web Applications (WAR files).
may contain one or more
Web Application Archive (WAR)
Includes following files:
/WEB-INF/web.xml - deployment descriptor
/WEB-INF/classes/* - Java classes used by application
/WEB-INF/lib/* - Java libraries used by application
/*.html, *.css, ... - HTML files, templates, style sheets
- all other files within WAR are accessible directly as Web resources including html, jsps, jsp tag libraries, images, sounds, etc.

Figure 12-2 Structure of EAR and WAR files

The `web.xml` file (also referred to as the deployment descriptor) is read on deployment by the Web application server and guides the setting up and running of the application. In particular, it has configuration information for starting up the servlets and then ensuring that Web requests are mapped to the right servlet or JSP. It also contains information on security (which user groups can access a particular set of URLs), filters (a mechanism to call some Java code before a request is processed), listeners (a mechanism to call some Java code on certain events), and configuration parameters to be passed to servlets or the application as a whole.

There are no requirements for the directory structure of a Web application outside of the `WEB-INF` directory. All these resources are accessible to clients (generally but not always Web browsers) directly from a URL given the context root. Naturally, we recommend that you structure the Web resources in a logical way for easy management (for example, an `images` folder to store graphics).

J2EE Web APIs

The main classes used within the J2EE frame work and the interactions between then are shown in Figure 12-3. The class `ApplicationServlet` is the only class outside of the J2EE framework, and would contain the application logic.

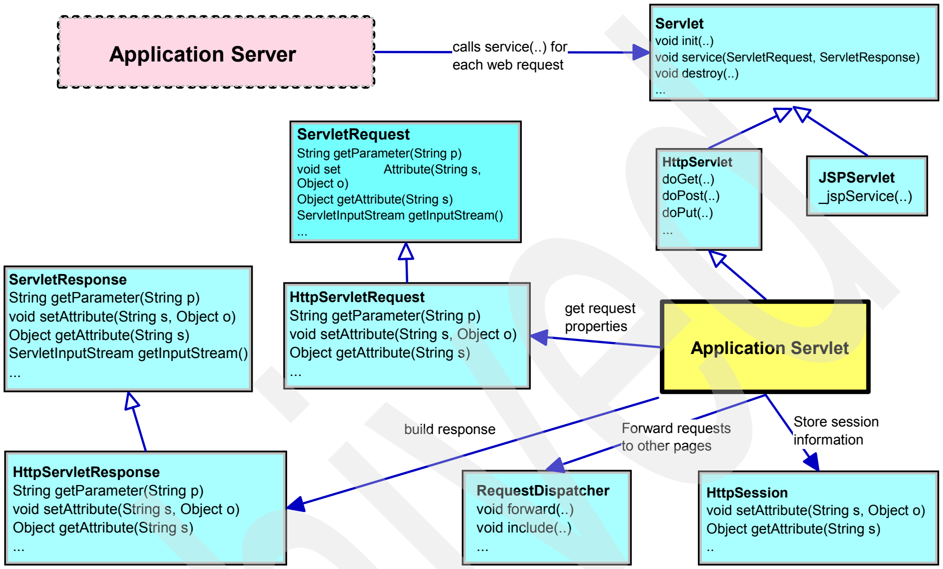

Application Server
calls service(..) for each web request
Servlet
void init(..)
void service(ServletRequest, ServletResponse)
void destroy(..)
...
ServletRequest
String getParameter(String p)
void set Attribute(String s, Object o)
Object getAttribute(String s)
ServletInputStream getInputStream()
...
HttpServlet
doGet(..)
doPost(..)
doPut(..)
...
JSPServlet
_jspService(..)
ServletResponse
String getParameter(String p)
void setAttribute(String s, Object o)
Object getAttribute(String s)
ServletInputStream getInputStream()
...
HttpServletRequest
String getParameter(String p)
void setAttribute(String s, Object o)
Object getAttribute(String s)
...
get request properties
Application Servlet
build response
Forward requests to other pages
Store session information
HttpServletResponse
String getParameter(String p)
void setAttribute(String s, Object o)
Object getAttribute(String s)
...
RequestDispatcher
void forward(..)
void include(..)
...
HttpSession
void setAttribute(String s, Object o)
Object getAttribute(String s)
..

Figure 12-3 J2EE Web component classes

The main classes are as follows:

- **HttpServlet (extends Servlet)**—The main entry point for handling a Web request. The `doGet`, `doPost` (and others) methods invoke the logic for building the response given the request and the underlying business data and logic.
- **HttpJspServlet (extends Servlet)**—The Application Server will automatically compile a JSP into a class that extends this type. It runs like a normal servlet and its only entry point is the `_jspService` method.
- **HttpRequest (extends Request)**—Provides an API for accessing all pertinent information in a request.
- **HttpResponse (extends Response)**—Provides an API for creating whatever response is required given the request and the application state.
- **HttpSession**—Stores any information required to be stored across a user session with the application (as opposed to a single request).
- **RequestDispatcher**—Within a Web application, it is often required to redirect the processing of a request to another servlet. This class provides methods to do this.

Note that there are other classes in the J2EE Web components framework, and for a full description of the classes available refer, to “More information” on page 538, which provides a link to the J2EE servlet specifications.

JSPs

It is possible to include any valid fragment of Java code in a JSP and mix it with HTML. In the JSP code, the Java tags are marked by `<%` and `%>` and on deployment (or sometimes the first page request, depending on configuration) the JSP is compiled into a servlet class. This process combines the HTML and the scriptlets in the JSP file in the `_jspService` method that populates the `HttpResponse` variable. Combining a lot of complex Java code with HTML can result in a very complicated JSP file, and except for very simple examples, this practice should be avoided.

One way around this is to use custom JSP tag libraries, which are tags defined by developers that initiate calls to a Java class. These classes implement the `Tag`, `BodyTag` or `IterationTag` interfaces from the `javax.servlet.jsp.tagext` package which is part of the J2EE framework. Each tag library is defined by a `.tld` file which includes the location and content of the taglib class file. A reference to this file has to be included in the deployment descriptor.

The most widely available tag library is the JavaServer Pages Standard Template Library (JSTL), which provides some simple tags to handle simple operations required in most JSP programming tasks, including looping, internationalization, XML manipulation, and even processing of SQL result sets.

The RedBank application presented later uses JSTL tags to display tables and add URLs to a page, and the final section of this chapter contains references on what is possible with JSPs and tag libraries.

Model-view-controller (MVC) pattern

The model-view-controller (MVC) concept is a pattern used many times when describing and building applications with a user interface component, including J2EE applications.

Following the MVC concept, a software application or module should separate its business logic (model) from its presentation logic (view). This is desirable because it is likely that the view will change over time and it should not be necessary to change the business logic each time. Also, many applications might have different views of the same business model, and if the view is not separated, then adding an additional view causes considerable disruptions and increase the component's complexity.

This separation can be achieved through the layering of the component into a *model* layer (responsible for implementing the business logic) and a *view* layer (responsible for rendering the user interface to a specific client type). In addition the *controller* layer sits between the two, intercepting the a requests from the view (or presentation) layer and mapping them to calls on the business model and then returning the response based on a response page selected by the controller layer. The key advantage provided by the controller layer is that the presentation can focus just on the presentation aspects of the application and leave the flow control and mapping to controller layer.

There are several ways to achieve this separation in J2EE applications and various technologies, such as JavaServer Faces (JSF) and Struts, focus on different ways of applying the MVC pattern. The focus of this chapter is on JSPs and servlets that fit into the view and controller layers of the MVC pattern. If only servlets and JSPs are used in an application, then the details of how to implement the controller layer is left to whatever mechanism the development team decides is appropriate, and which they can create using Java classes.

In one example presented later in this chapter, a *Command* pattern (see *Design Patterns, Elements of Reusable Object-Oriented Software* in the bibliography) is applied to encompass the request to the business logic and interactions made with the business logic through a facade class, while in the other interactions the request is sent directly to a servlet that makes method calls through the facade.

Web development tooling

Application Developer includes many Web development tools for building static and dynamic Web applications. Some of these are focused on technologies such as struts, portals and JavaServer Faces which are described in other chapters. In this section, we highlight the following tools and features, which focus on the more fundamental aspects of Web development.

The tools described in this section include:

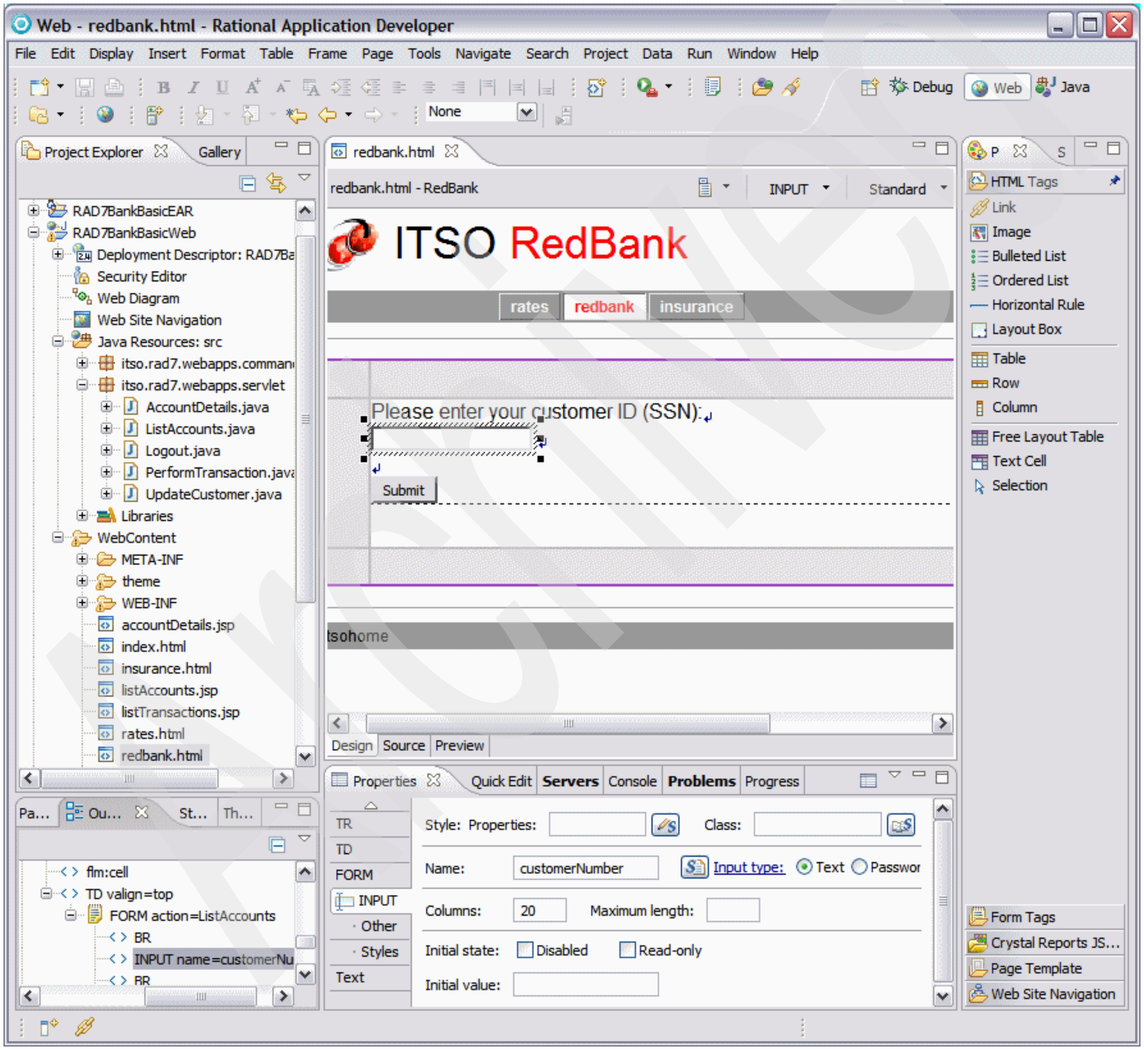
- Web perspective and views
- Web Site Navigation Designer
- Web Diagram
- Page Designer
- Page templates
- CSS Designer
- Security Editor
- File creation wizards

Some of these tools are illustrated further in the RedBank Web application example built later in this chapter.

Web perspective and views

The Web perspective and its supporting views is designed to help Web developers build and edit Web resources, such as servlets, JSPs, HTML pages, style sheets, and images, as well as the deployment descriptor files.

The Web perspective can be opened by selecting **Window** → **Open Perspective** → **Web** from the Workbench. Figure 12-4 displays the default layout of the Web perspective with a simple `index.html` open in the editor.



Style: Properties:
Class:
Name: customerNumber
Input type: ● Text ○ Passwor
Columns: 20
Maximum length:
Initial state: □ Disabled □ Read-only
Initial value:

Figure 12-4 Web perspective

In the Web perspective, there are many views accessible (by selecting **Window** → **Show View**), several of which are already open in the Web perspective default setting.

By default, the views available in the Web perspective are as follows:

- **Colors view**—When working with an HTML or JSP page in the Page Designer view, the colors view gives the user the facility to manipulate colors of cells, text, tables, and other HTML tags.
- **Console view**—Shows output to SystemOut from any running processes.
- **Gallery view**—Provides a large set of folders of predefined images, sounds, and style sheets users can apply to their Web pages.
- **Links view**—Given what is shown in the Page Designer view, the Links view shows the links out from this file and elements in the Web project that link to it.
- **Navigator view**—Provides a project and folder view of the workspace (similar to Project Explorer) but shows the files exactly as they exist in the file system.
- **Outline view**—Shows an outline of the file being viewed. For HTML and JSP this shows a hierarchy of tags around the current cursor position. Selecting a tag in this view will move the cursor in the main view to the selected element. This is particularly useful for moving quickly around a large HTML file.
- **Page Data view**—When editing JSP files, this gives a list of any page or scripting variables available.
- **Page Designer**—WYSIWYG editor for JSP and HTML. This consists of three tabs, Design (where the user can drag and drop components onto the page), Source (showing the HTML), and Preview (giving an indication of what the final page looks like).
- **Palette view**—When editing JSP or HTML files, this provides a list of HTML items (arranged in drawers) which can be dragged and dropped onto pages.
- **Problems view**—Shows any outstanding errors, warnings, or informational messages for the current workspace.
- **Project Explorer view**—Shows a hierarchy view of all projects, folders and files in the workspace. Note that in the Web perspective it structures the information within Web projects in a way that makes navigation easier.
- **Properties view**—Shows the properties for the item currently selected in the main editor.
- **Quick Edit view**—When editing HTML or JSP files, the Quick Edit view provides a mechanism to quickly add Java Script to a given screen component on certain events, for example `onClick`.
- **Servers view**—Useful if the user wants to start or stop test servers while debugging.

- **Snippets view**—Allows editing of small bits of code, including adding and editing actions assigned to tags. Items from the Snippets view can be drag and dropped into the Quick Edit view.
- **Styles view**—Allows the user to edit and apply both pre-built and user-defined styles sheets to HTML elements and files.
- **Tasks view**—Allows the user to maintain a list of things to be done within the workspace. Note for Java code comment text tags can be configured which will automatically add items to the Tasks list (see **Window** → **Preferences** then **Java** → **Compiler** → **Task Tags** to edit these tags)
- **Thumbnails view**—Given the selection of a particular folder in the Gallery view, this view shows the contents of the folder.

Note: For more information on the Web perspective and views, refer to "Web perspective" on page 146.

Web Site Navigation Designer

The Web Site Navigation Designer is provided to simplify and speed up the creation of the entire Web site navigation and can launch wizards to facilitate the creation of HTML pages and JSPs. The tool provides features to view the Web site in the Navigation tab, to add new pages, delete pages, and move pages within in the site. This tool is especially useful for building pages in a Web application that uses a page template.

The flow of an application can be visually laid out and then the elements (JSPs, HTML pages) rearranged until it fits the requirements. Once the flow is arranged, a developer can begin creating pages based on this design.

As a Web site design is built, the information is stored in the `website-config.xml` file so that navigation links can be generated automatically. This can include a page trail (showing the hierarchy of Web pages to get to the currently shown page) or a set of page tabs to quickly navigate to other aspects of the application.

When the structure of a site changes, (for example when a new page is added) the navigation links are automatically regenerated to reflect the new Web site structure.

To launch the Web Site Designer, double-click **Web Site Navigation** found in the root of the Web project folder.

Note: For a detailed example of using the Web Site Designer, refer to "Launching the Web Site Designer" on page 502.

Web Diagram

The Web Diagram (Figure 12-5) is another view of the Web application, which shows the pages, the links between pages, and the page variables available on each page. In addition, if Struts of JavaServer Faces technologies are being used in the application, this view can include extra information to link the pages and data together as appropriate for the selected technology.

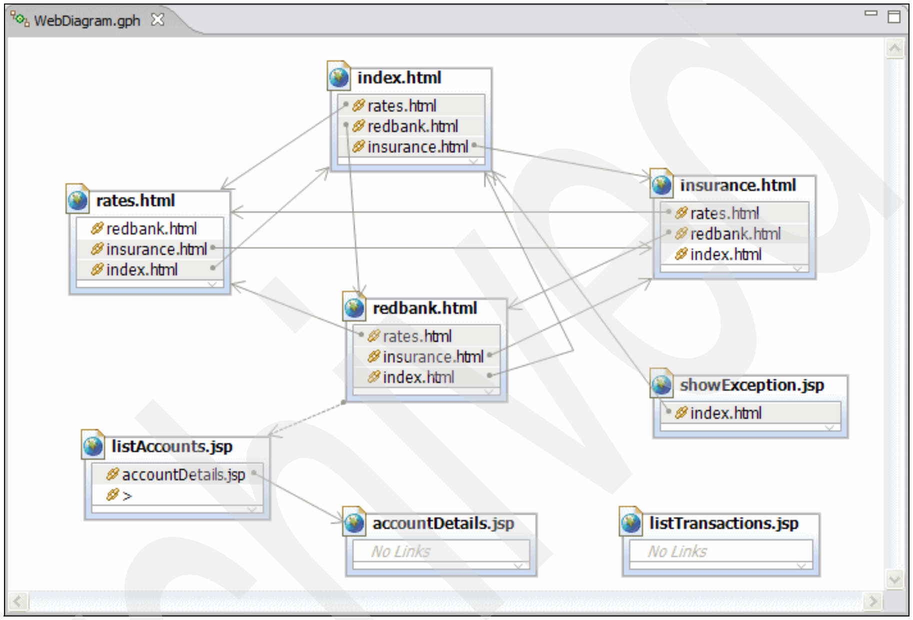


Figure 12-5 Web Diagram

In the RedBank example, the Web Diagram is not used to build the application. Instead, the Web Site Navigation Designer is the tool with which pages are built and relationships between pages are shown. Chapter 14, “Develop Web applications using JSF and SDO” on page 587 and Chapter 13, “Develop Web applications using Struts” on page 541 both provide examples of how to apply the Web Diagram.

Page Designer

Page Designer is the primary editor within Application Developer for building HTML, XHTML, JSPs, and JSF source code. It provides three representations of a page: Design, Source, and Preview:

- The Design tab provides a WYSIWYG environment to visually design the contents of the page.

- The Source tab provides access to the page source code showing the raw HTML or JSP contents.
- The Preview tab shows what the page would like if displayed in a Web browser.

A good development technique is to work within the Design tab of Page Designer and build up the HTML contents by clicking and dragging items from the Palette view onto the page and arranging them with the mouse or editing properties directly from the Properties view. The Outline view is also very helpful to navigate quickly to another tag that is related (for example, an ancestor) to the tag being edited. The Source tab can be used to change details not immediately obvious in the Design tab, and the Preview tab can be used throughout the process to verify the look of the final result.

Often it is the case that HTML content is provided to a development team and created from tools other than Application Developer. These files can be imported simply using the context menu on the target directory and selecting **File** → **Import** → **General** → **File System** and browsing to the new file and clicking **Import**. When an imported file is opened in Page Designer, all the standard editing features are available.

Note: For a detailed example of using the Page Designer, refer to “Developing the static Web resources” on page 508 and “Working with JSPs” on page 521.

Page templates

A page template contains common areas that you want to appear on all pages, and content areas which are intended to be unique on each page. They are used to provide a common look and feel for a Web project.

The Page Template File creation wizard is used to create these files. Once created the file can be modified in Page Designer. The page templates are stored as `*.htpl` files for HTML pages and `*.jtpl` files for JSP pages. Changes to the page template are reflected in pages that use that template. Templates can be applied to individual pages, groups of pages, or applied to an entire Web project. Areas can be marked as read-only meaning that Page Designer will not allow the user to modify those areas.

When creating a new page template, the user is prompted if the template is to be a **dynamic page template** or a **design time template**:

- Dynamic page templates are a new feature in Application Developer V7.0, which make it possible to apply changes dynamically on a deployed application and use Struts-Tiles technology.

- Design time templates allow changes to be made and applied to the template at design or build time, but once an application is deployed and running, the page template cannot be changed. Design time templates do not rely on any technologies other than the standard J2EE/servlet libraries.

Note: For examples of creating, customizing, and applying page templates refer to “Create new page templates” on page 494 and “Customize page templates” on page 496.

CSS Designer

Cascading style sheets (CSS) are a tool used with HTML pages to ensure that an application has consistent colors, fonts, and sizes across all pages. It is possible to create a default style sheet when creating a new project and there are several samples included with Application Developer. Usually, a good idea is to decide on the overall theme (color, fonts) for your Web application in the beginning and create the style sheet at the start of the development effort. Then, as you create the HTML and JSP files, you can select that style sheet to ensure that the look of the Web pages are consistent. Style sheets are commonly kept in the `WebContent/theme` folder.

The CSS Designer is used to modify cascading style sheet `*.css` files. It provides two panels, the right hand side showing all the text types and their respective fonts, sizes, and colors which are all editable. On the left hand side, a sample of how the various settings will look. Any changes made are immediately applied to the design in Page Designer if the HTML file is linked to the CSS file.

Note: An example of customizing style sheets used by a page template can be found in “Customize a style sheet” on page 501.

Security Editor

The Security Editor is a new enhancement with Application Developer V7.0. It provides a wizard to specify security groups within a Web application and the URLs that group has access to. The J2EE specification allows for security groups and levels of access to defined sets of URLs to be defined in the deployment descriptor and the Security Editor provides a nice interface for this information (Figure 12-6).

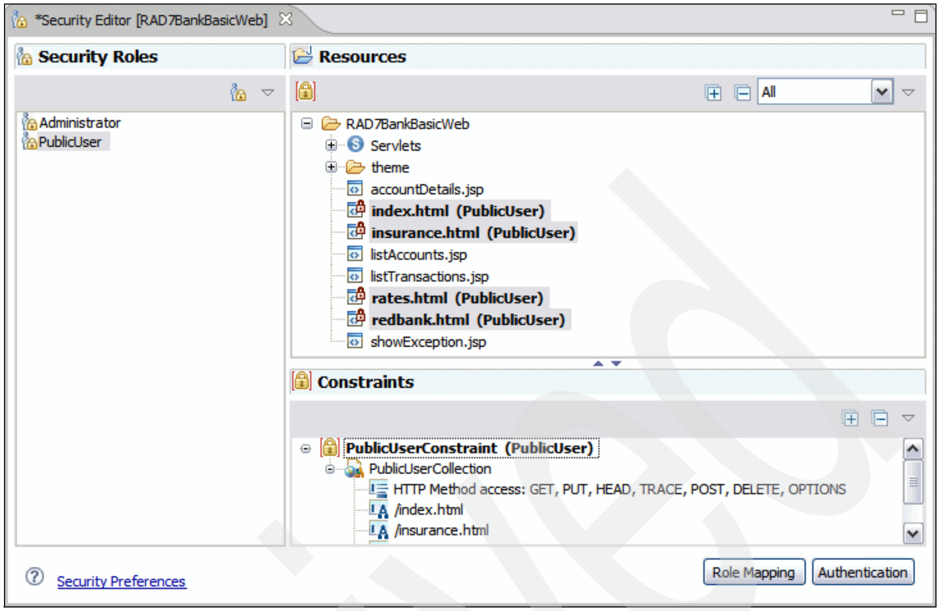


Figure 12-6 Security Editor example

Selecting an entry in the Security Roles pane shows the resources members of that role in the Resources pane, and the Constraint rules that are applicable for the role and resource (if one is selected). Each entry in the Constraints window has a list of resource collections, which specify the resources available to it and which HTTP methods can be used to access these resources. Using context menus it is possible to create new roles, security constraints, and add resource collections to these restraints.

Note that J2EE security specification defines the mechanism for declaring groups and the URL sets that each group can access, but it is up to the Web Container to map this information to an external security system. WebSphere's administrative console provides the mechanism to configure an external LDAP directory. Refer to the IBM Redbooks publication, *Experience J2EE! Using WebSphere Application Server V6.1*, SG24-7297.

File creation wizards

Application Developer provides many Web development file creation wizards by selecting **File** → **New** → **Other** and then from **Select a Wizard** expand the Web folder and select the type of file required. These wizards prompt the user for the key features of the new artifact and can help a user to quickly get a skeleton of the component they require. The artifact created by the wizard can always be manipulated directly if required.

The wizards available in the Web perspective are as follows:

- **CSS**—The CSS file wizard is used to create a new cascading style sheet (CSS) in a specified folder.
- **Dynamic Web Project**—Steps the user through the creation of a new Web project, including which features the project use and any page templates present.
- **Filter**—Constructs a skeleton Java class for a J2EE filter, which provides a mechanism for performing processing on a Web request before it reaches a servlet. The wizard also updates the *web.xml* file with the filter details.
- **Filter Mapping**—Steps the user though the creation of a set of URLs to map a J2EE filter with, The result of this wizard is stored in the deployment descriptor.
- **Javascript**—Used to create a new Javascript file in a specified folder.
- **Life-cycle Listener**—The J2EE specification allows for classes to be configured to receive pertinent events from the Web container. For example the classes which implement the `HttpSessionListener` interface and are declared in the deployment descriptor, receive notification every time a `HttpSession` is created or destroyed. This wizard guides the user through the creation of such a listener and automatically adds a reference to it to the deployment descriptor.
- **Security Constraint**—Used to populate the `<security-constraint>` in the deployment descriptor which contains a set of URLs and a set of http methods which members of particular security role might (or might not) be entitled to access.
- **Security Role**—Adds a `<security-role>` element to the deployment descriptor.
- **Servlet**—Used to create a skeleton servlet class and add the servlet to the deployment descriptor.
- **Servlet Mapping**—Steps the user through the creation of new URL to servlet mapping and adds it to the deployment descriptor.
- **Static Web Project**—Steps the user through building a new Web project containing only static pages.
- **Web Diagram**—It is possible to create a Web Diagram for a Web project which already has a large number of pages. This wizard creates an empty Web Diagram onto which the user can place existing pages and show existing page relationships.
- **Web Page**—The Web Page wizard allows you to create an HTML or JSP file in a specified folder, with the option to create from a large number of page templates.

- **Web Page Template**—The Page Template File wizard is used to create new page template files in a specified folder, with the option to create from a page template or create as a JSP Fragment, and define the markup language (HTML, HTML Frameset, Compact HTML, XHTML, XHTML Frameset, and WML 1.3). You can optionally create a new page template from an existing page template. In addition, you can select from one of the following models: Template contains Faces Components, Template containing only HTML, Template containing JSP.

Note that there are also a number of wizards specifically for Struts and JSF, which are discussed in other chapters.

Summary of new features in V7.0

The underlying Java libraries for servlets is J2EE 2.4 by default, which is the same as that supplied for Rational Application Developer V6. Therefore, the basic usage of the J2EE APIs has not changed, but there have been significant changes in the tools available for creating artifacts within a Web application.

The main changes are as follows:

- **Dynamic page templates**—Previously templates were applied at design time only, now changes to the template can be applied at runtime using struts Tile support.
- **New wizard for Web pages and Web page templates**—These replace the wizards for HTML, JSP, Faces JSP pages, and page templates. The process is now simpler and page templates are a more prominent feature.
- **Security Editor**—This is a new tool to make it easier to specify resource sets, user roles and security constraints to apply between the two.
- **Web Diagram Editor**—This view has been enhanced to show more information and to improve the support for reverse engineering. This view is especially useful for Struts and JSF development but also for standard Web pages as it provides support for showing links and page data beans as well as now reflecting changes made to the diagram to the underlying classes.
- **Web Library projects**—Previously Web Library projects were configured in the deployment descriptor editor even though the relationship to the other project was not expressed in the `web.xml` file. This relationship is now configured in the project properties, under **J2EE Module Dependencies**.
- **Cascading style sheet templates**—It is now possible to create a CSS file from a predefined set of templates supplied with Application Developer.

- **Page Designer**—The following changes have been made to the Page Designer:
  - **HTML/JSP code folding (structured source editting.pdf)**—These now have support in HTML and JSP editing for *code folding*, that is, expanding and shrinking HTML elements (also CSS, XML, HTML and JSP).
  - **Struts tiles support**—Projects can now be created using Struts tiles. JSP pages using Struts tiles now visualize these components in the Design tab.
  - **Page Data view usability**—The items and menus on the Page Data view are categorized for better usability.
  - **Preview page support for Linux**—Formerly this was only available for Windows users, but it is now also available on Linux.
  - **HTML validation**—The HTML editor now features as-you-type validation as well as validation when building.
  - **Web site navigation improvements**— New menu style navigation tabs (Tabbed menu, Vertical menu and Tabbed menu) are available. Pages which are shared in multiple locations in the Web structure can now be included in Navigation bars. Also, navigation on unmapped pages (that is pages not shown in the Web Site Navigation view) is now possible and the usual navigation elements are available for adding to the page.
  - **Visualization for third Party JSF components** —Now third party providers of JSF components can provide visualization of the components, which is displayed in the Design tab.

RedBank application design

In this section we describe the design for the ITSO RedBank Web application, also known as RedBank. The intent is to outline the design of the RedBank application and how it fits into the J2EE Web framework, particularly with regard to JSPs and servlets.

Model

The model for the RedBank project is implemented using a simple Java project, exposed to other components through a facade interface (called `ITSOBank`). The main `ITSOBank` object is a singleton object, accessible by a single static public method called `getBank`.

The `ITSOBank` object is composed of the other business objects which make up the application, including `Customer`, `Account`, and `Transaction`. The facade into the bank object includes methods such as `getCustomer`, `getAccounts`, and `withdraw`, `deport`, and `transfer`. Figure 12-7 shows a simplified UML class diagram of the model. The model is described in detail in Chapter 7, “Develop Java applications” on page 227.

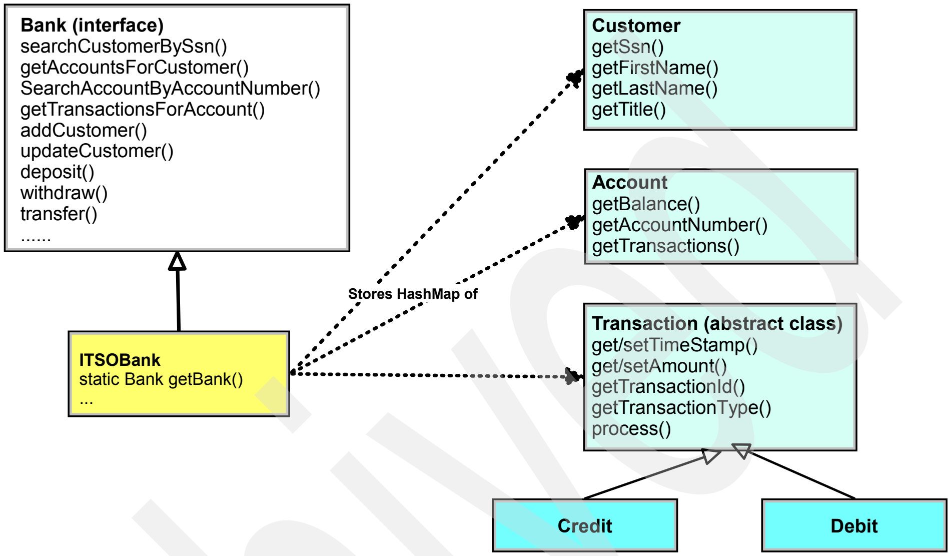


Figure 12-7 Class diagram for RedBank model

The underlying technology to store data used by the `ITSOBank` application are Java HashMaps. These are populated at startup in the constructor and obviously the data is lost every time the application is restarted. In a real world example this would be stored in a database, but for the purposes of this example HashMaps are fine. In Chapter 16, “Develop Web applications using EJBs” on page 719 the `ITSOBank` model is modified to run as EJBs and the application data is stored in a database.

View layer

The view layer of the RedBank application is composed of four HTML files and four JSP files. The application home page is the `index.html` containing a link to three HTML pages (`rates.html`, `insurance.html`, and `redbank.html`).

- `rates.html`, `index.html`, and `insurance.html` are simple static HTML pages showing information, but no forms or entry fields.
- `redbank.html` contains a single form which allows a user to type in the customer ID to access the customer services such as accessing balance, and performing transactions. Note that although the account number is verified, security issues (logon and password) are not covered in this example.

- From redbank.html the user is shown the lisAccounts.jsp page, which shows the customer's details, a list of accounts, and a button to logout.
- Selecting an account brings up the accountDetails.jsp, which also shows the balance for the selected account and a form through which a transaction can be performed. This screen also shows the current account number and balance, both dynamic values. A simple JavaScript code controls whether the amount and destination account fields are available, depending on the option selected. One of the transaction options on AccountDetails.jsp is List Transactions, which invokes the listTransactions.jsp.
- If anything goes wrong in the regular flow of events, the exception page (showException.jsp) is shown to inform the user of the error.
- These four JSP pages (listAccounts.jsp, accountDetails.jsp, listTransactions.jsp, and showException.jsp) make up the dynamic pages of the RedBank application.

Refer to Figure 12-21 on page 507 for a diagram showing the planned structure of pages within the RedBank application. The diagram was created using the Web Site Designer tool.

Controller layer

The controller layer was implemented using two different strategies: One straightforward; and the other more complex, but more applicable to a real world situation.

The application has a total of five servlets:

- **ListAccounts**—Gets the list of accounts for one customer.
- **AccountDetails**—Displays the account balance and the selection of operations: List transactions, deposit, withdraw, and transfer.
- **Logout**—Invalidates the session data.
- **PerformTransaction**—Performs the selected operation by calling the appropriate control action: ListTransactions, Deposit, Withdraw, or Transfer.
- **UpdateCustomer**— Updates the customer information.

The first three servlets use a simple function call from the servlet to the model classes to implement their controller logic and then use the RequestDispatcher to forward control onto another JSP or HTML resource. The pattern used is shown in the sequence diagram in Figure 12-8.

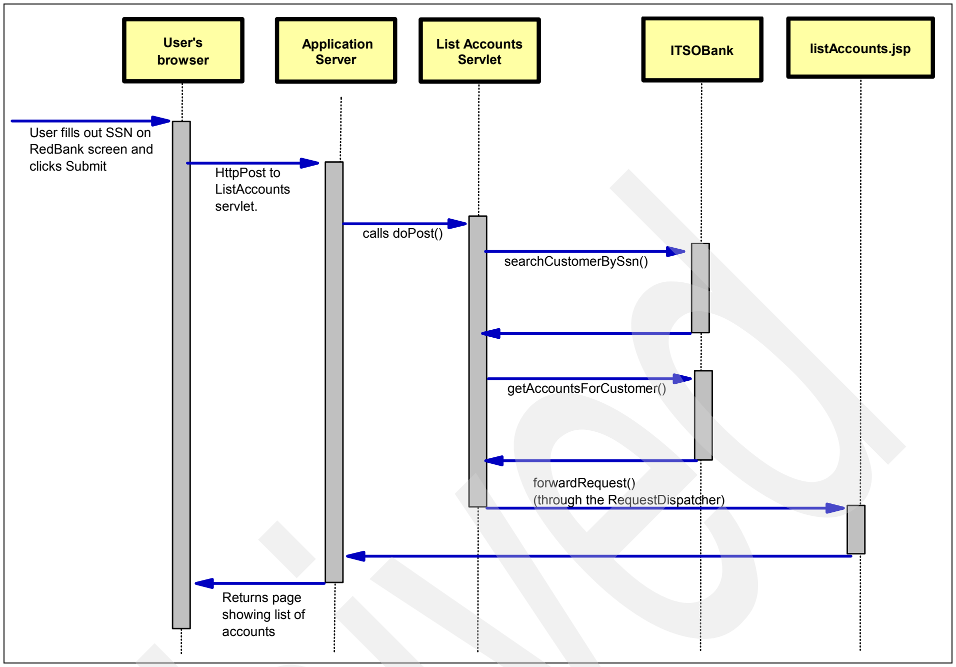

User's browser
Application Server
List Accounts Servlet
ITSOBank
listAccounts.jsp
User fills out SSN on RedBank screen and clicks Submit
HttpPost to ListAccounts servlet.
calls doPost()
searchCustomerBySsn()
getAccountsForCustomer()
forwardRequest() (through the RequestDispatcher)
Returns page showing list of accounts

Figure 12-8 ListAccounts sequence diagram

`PerformTransaction` uses a different implementation pattern. It acts as a front controller, simply receiving the HTTP request and passing it to the appropriate control action object. These objects are responsible for carrying out the control of the application. Figure 12-9 shows a sequence diagram for the list transaction operation from the account details page, including the function calls through `PerformTransaction`, the `ListTransactionsCommand` class, onto the model classes, and forwarding to the appropriate JSP.

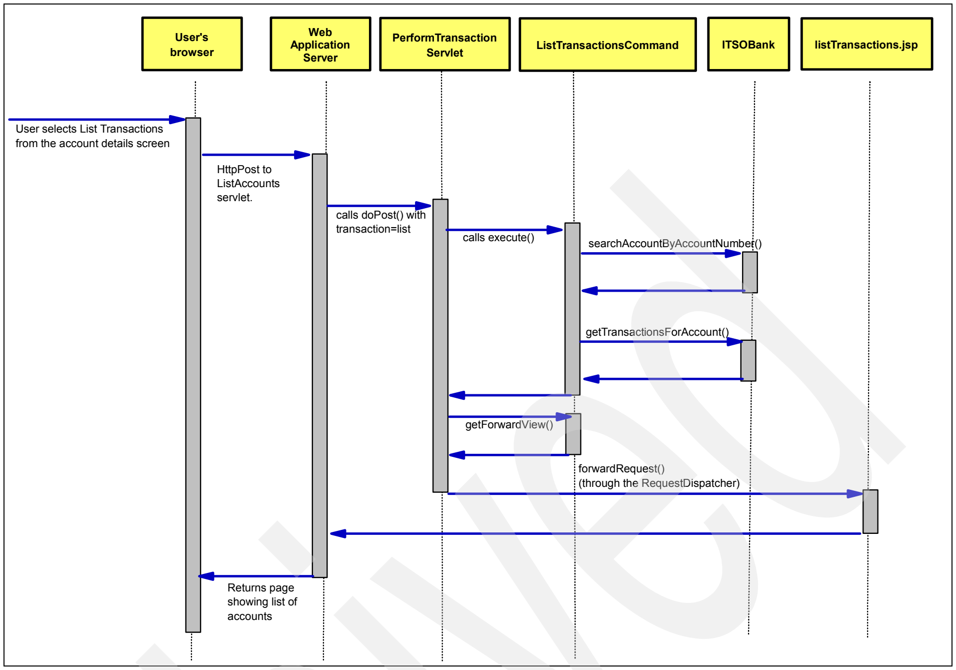

User's browser
Web Application Server
PerformTransaction Servlet
ListTransactionsCommand
ITSOBank
listTransactions.jsp
User selects List Transactions from the account details screen
HttpPost to ListAccounts servlet.
calls doPost() with transaction=list
calls execute()
searchAccountByAccountNumber()
getTransactionsForAccount()
getForwardView()
forwardRequest() (through the RequestDispatcher)
Returns page showing list of accounts

Figure 12-9   PerformTransaction sequence diagram

The Struts framework provides a much more detailed implementation of this strategy and in a standardized way. Refer to Chapter 13, “Develop Web applications using Struts” on page 541 for more details.

Note: Action objects, or commands, are part of the command design pattern. For more information, refer to *Design Patterns: Elements of Reusable Object-Oriented Software*.

Implementing the RedBank application

Using an example, this section introduces you to the tools within Application Developer’s that facilitate the development of Web applications. In the example, we step through the creation of different Web artifacts (including page templates, HTML, JSPs and servlets) and demonstrate how to use the tools available.

Note that the completed application has already been developed and is available to explore if required by importing the project interchange file:

```
c:\7501code\zInterchangeFiles\webapps\RAD7BankBasicWeb.zip
```

The section is organized as follows:

- Creating the Web project
- Web project directory structure
- Importing the RedBank model
- Defining the Web site navigation and appearance
- Developing the static Web resources
- Developing the dynamic Web resources
- Working with JSPs

At the end of this section, the RedBank application should be ready for testing.

Creating the Web project

The first step is to create a Web project in the workspace.

> **Note:** Before this, it might be a good idea to check that Web capabilities are enabled. Select **Windows** → **Preferences** and expand **General** → **Capabilities** make sure that **Web Developer** options (including basic, typical and advanced) are selected.

There are two types of Web projects available in Application Developer, namely, static and dynamic. Static Web projects contain static HTML resources and no Java code, and thus are comparatively simple. In order to demonstrate as many features of Application Developer as possible, and because the RedBank application contains both static and dynamic content, a dynamic Web project is used for this example.

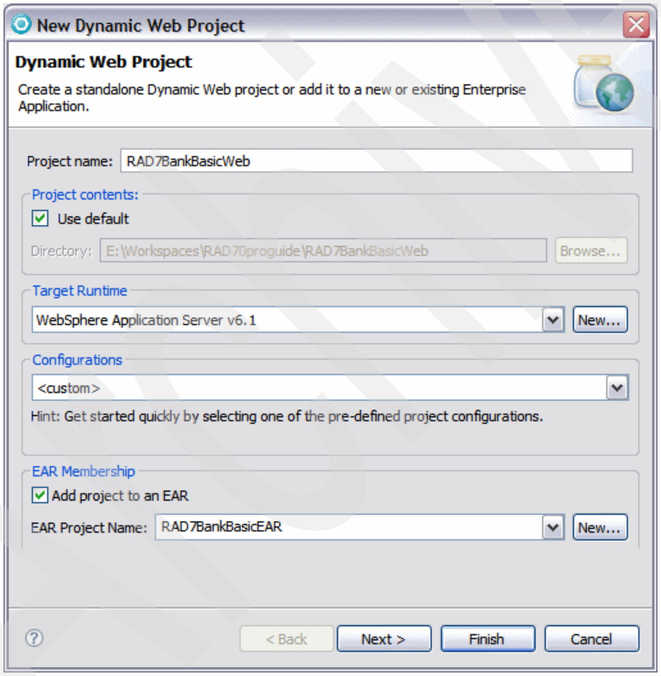
In Application Developer, perform the following steps:

- Open the Web perspective by selecting **Window** → **Open Perspective** → **Web**.
- To create a new Web Project, select **File** → **New** → **Project**.
- In the New Project dialog, select **Web** → **Dynamic Web Project**, and then click **Next**.
- In the New Dynamic Web Project dialog, enter the following items (as seen in Figure 12-10), and then click **Next**:
  - Enter `RAD7BankBasicWeb` in the name field.
  - For Project contents, select **Use Default** (default). This specifies where the project files should be placed on the file system. The default option of leaving them in the workspace is usually fine.

- Target Runtime: Select **WebSphere Application Server v6.1** (default).

  This option will display the supported test environments that have been installed. Use WebSphere Application Server V6.1 Test Environment.
- Under Configurations select **<custom>** (default). This option allows a user to make use of some pre-defined project configurations to speed up creating new projects.
- Select **Add module to an EAR project** (default).

  Dynamic Web Projects, such as the one we are creating, run exclusively within an enterprise application. For this reason, you have to either create a new EAR project or select an existing project.
- EAR Project Name: **`RAD7BankBasicEAR`** (overtype the default). Because we selected **Add module to an EAR project**, the wizard will create a new EAR project.



New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: RAD7BankBasicWeb

Project contents:

☑ Use default

Directory: E:\Workspaces\RAD70proguide\RAD7BankBasicWeb

Target Runtime: WebSphere Application Server v6.1

Configurations: <custom>

Hint: Get started quickly by selecting one of the pre-defined project configurations.

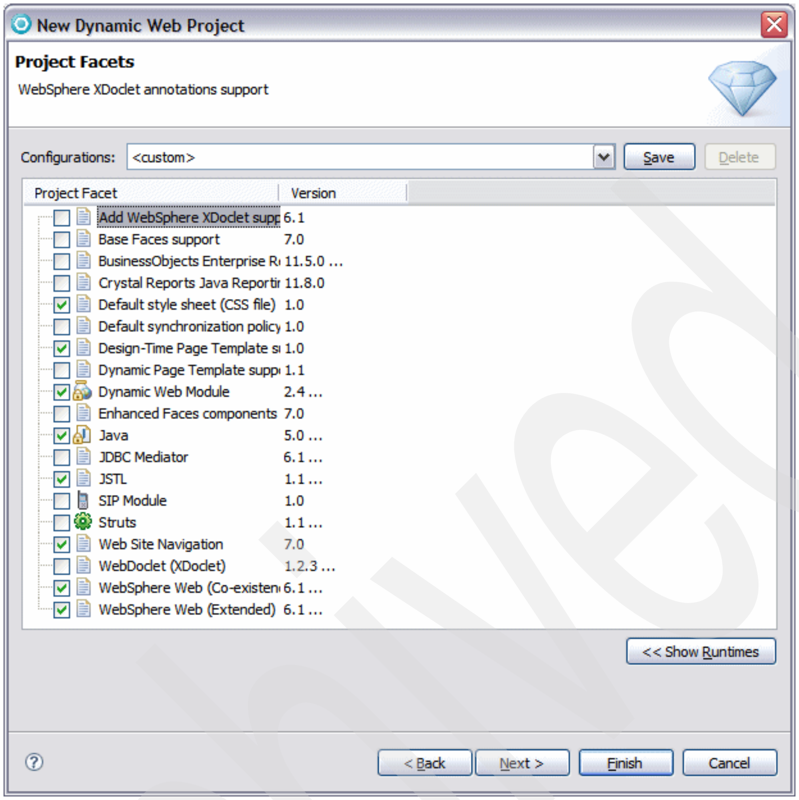
EAR Membership

☑ Add project to an EAR

EAR Project Name: RAD7BankBasicEAR

Figure 12-10 New Dynamic Web Project

- ► In the Project Facets dialog, select the features for **Default Style Sheet**, **Design-Time Page Template support, JSTL, Web Site Navigation WebSphere Web (Co-existence)** and **WebSphere Web (Extended)** as seen in Figure 12-11.



New Dynamic Web Project

Project Facets

WebSphere XDoclet annotations support

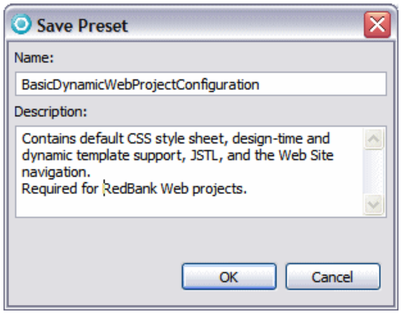
Configurations: <custom>

Project Facet	Version
□ Add WebSphere XDoclet supp	6.1
□ Base Faces support	7.0
□ BusinessObjects Enterprise R	11.5.0 ...
□ Crystal Reports Java Reporti	11.8.0
☑ Default style sheet (CSS file)	1.0
□ Default synchronization policy	1.0
☑ Design-Time Page Template s	1.0
□ Dynamic Page Template supp	1.1
☑ Dynamic Web Module	2.4 ...
□ Enhanced Faces components	7.0
☑ Java	5.0 ...
□ JDBC Mediator	6.1 ...
☑ JSTL	1.1 ...
□ SIP Module	1.0
□ Struts	1.1 ...
☑ Web Site Navigation	7.0
□ WebDoclet (XDoclet)	1.2.3 ...
☑ WebSphere Web (Co-existen	6.1 ...
☑ WebSphere Web (Extended)	6.1 ...

Figure 12-11 Dynamic Web Project Features Selection

> **Note:** The options shown with a "..” next to the Version allow you to alter the underlying version of this feature being selected. By default, the latest version available is always chosen.

From this dialog, it is also possible to save the configuration for future projects. To do this, just click **Save** and enter a configuration name and a description, as shown in Figure 12-12. The new configuration is available as a configuration option for subsequent projects and shown in the Configurations drop-down list (see Figure 12-10).


Save Preset

Name:

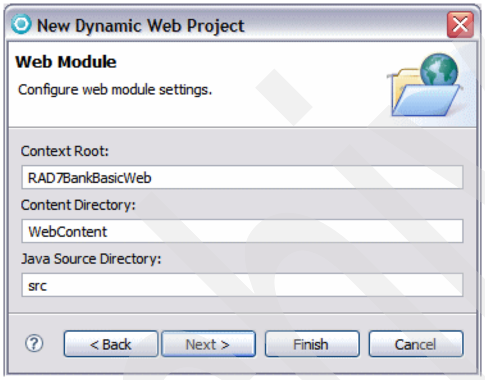
BasicDynamicWebProjectConfiguration

Description:

Contains default CSS style sheet, design-time and dynamic template support, JSTL, and the Web Site navigation.
Required for RedBank Web projects.

Figure 12-12 Save new Dynamic Web Page Facts Configuration

- In the Web Module pag, accept the default options (refer to Figure 12-13).


New Dynamic Web Project

Web Module

Configure web module settings.

Context Root: RAD7BankBasicWeb

Content Directory: WebContent

Java Source Directory: src

Figure 12-13 New Dynamic Web Project, Web Module settings

The settings here are as follows:

– Context Root: `RAD7BankBasicWeb`

The context root defines the base of the URL for the Web application. The context root is the root part of the URI under which all the application resources are going to be placed, and by which they will are referenced later. It is also the top level directory for your Web application when it is deployed to an application server.

– Content Directory: `WebContent`

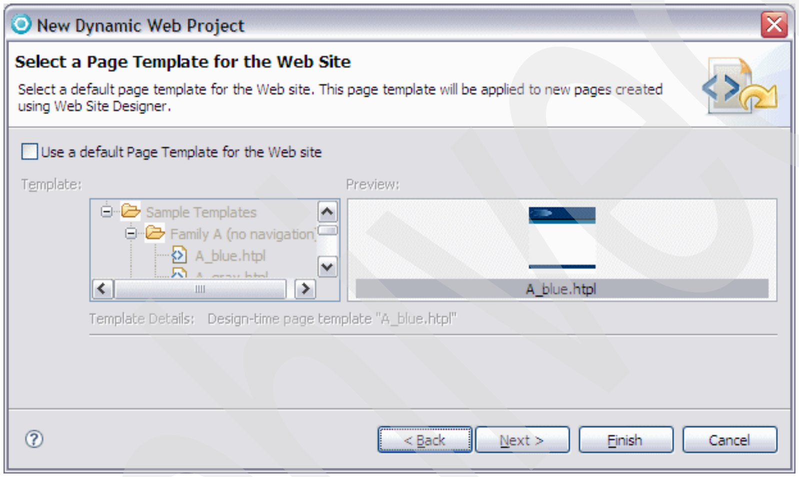
This specifies the directory where the files intended for the WAR file are created. All the contents of the WEB-INF directory, HTML, JSPs, images and any other files that are deployed with the application are contained under this directory. Usually the folder `WebContent` is sufficient.

– Java Source Directory: `src`

This specifies the directory where any Java source code used by the Web application is stored. Again, the default value of `src` should be sufficient for most cases.

Click **Next** to accept the default values.

- In the Select a Page template for the Web Site dialog (Figure 12-14), leave **Use a default Page template for the Web Site** cleared. This dialog allows you to select from page templates supplied with Application Developer. In this example, a new default template is created after the Web project is created.



New Dynamic Web Project

Select a Page Template for the Web Site

Select a default page template for the Web site. This page template will be applied to new pages created using Web Site Designer.

☐ Use a default Page Template for the Web site

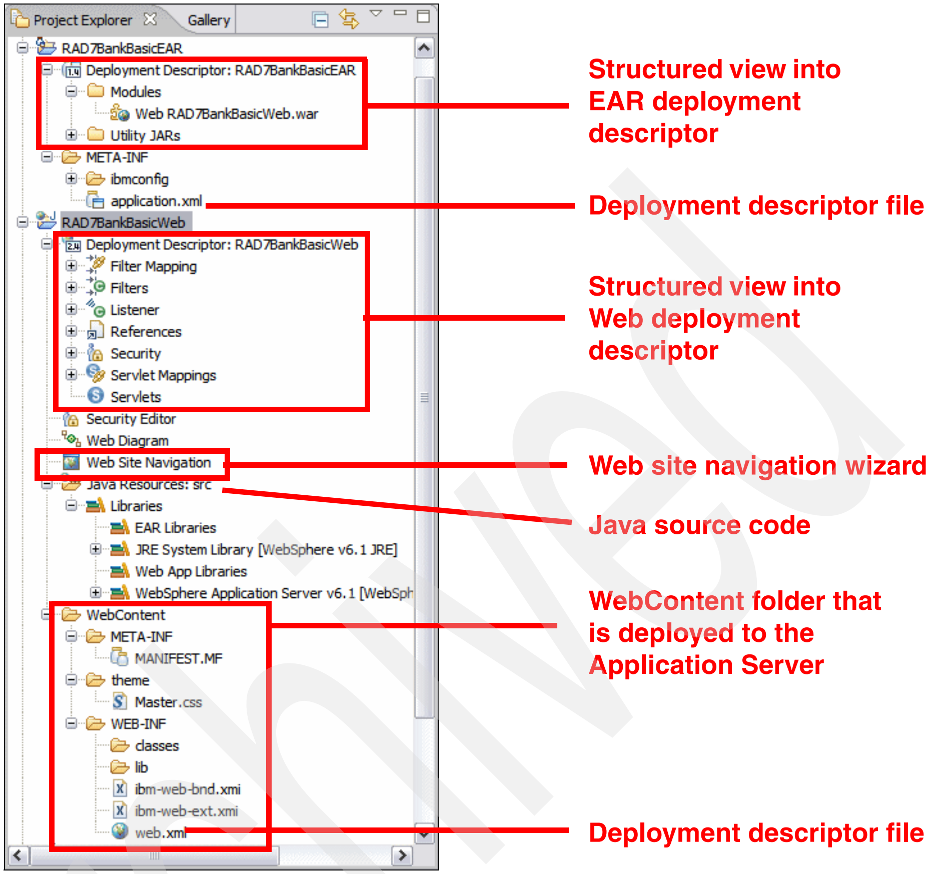
Template Details: Design-time page template "A_blue.htpl"

Figure 12-14 New Dynamic Web Project, Page Templates

- Click **Finish** and the Dynamic Web Project is created. The Web Diagram is opened automatically; this can be closed, as the work for building this application is done in the Web Site Navigation tool.

Web project directory structure

The Web project directory structure for the newly created `RAD7BankBasicWeb` project and its associated `RAD7BankBasicEAR` project (enterprise application) is displayed in Figure 12-15.



Structured view into EAR deployment descriptor

Deployment descriptor file

Structured view into Web deployment descriptor

Web site navigation wizard

Java source code

WebContent folder that is deployed to the Application Server

Deployment descriptor file

Figure 12-15 Web project directory structure

The main folders shown under the Web project are as follows:

- **Deployment Descriptor**—This shows an abstracted view of the contents of the projects `web.xml`. It includes sub folders for the main pieces which make up a Web project configuration, including Servlets and Servlet Mappings, Filters and Filter Mappings, Listeners, Security and References.
- **Web Site Navigation wizard**—Clicking on this starts up the tool for editing the page navigation structure.
- **Java Resources: src**—This folder contains the Java source code for regular classes, JavaBeans, and servlets. When resources are added to a Web project, they are automatically compiled, and the generated class files are added to the `WebContent\WEB-INF\classes` folder.
- **WebContent**—This folder holds the contents of the WAR file that is deployed to the server. It contains all the Web resources, including compiled Java

classes and servlets, HTML files, JSPs, and graphics needed for the application.

> **Important:** Files that are not under WebContent are not deployed when the Web project is published. Typically this would include Java source and SQL files. Make sure that you place everything that should be published under the WebContent folder.

Importing the RedBank model

The ITSO Bank Web application requires the classes created in Chapter 7, “Develop Java applications” on page 227. This section describes how to import the project interchange file. You can skip this import if you already have the final RAD7Java project in the workspace.

- Select **File** → **Import**.
- In the Import dialog, select **Other** → **Project Interchange** and click **Next**.
- In the **Import Projects** dialog, enter **`c:\7501code\webapps\RAD7Java.zip`** in the **From zip file** field, select the **RAD7Java** project, and then click **Finish**.

To verify that the model is running, it is possible to run the main method of the class `itso.rad7.bank.client.BankClient` class. This will invoke the bank facade classes directly, make some simple transactions directly and print out the results to `System.out`.

Defining the Web site navigation and appearance

This section demonstrates how to define the site pages and navigation using the Web Site Designer. A page template and style sheet are also created to provide a common appearance for the RedBank Web site pages. The page template is used to define a standard page layout (header, navigation menu, footer), and the style sheet is used by page templates to define standard fonts, colors, table formatting, and other style factors. Then the empty pages are created and the application is run to verify the basic navigation.

This section includes the following tasks:

- Create new page templates.
- Customize page templates.
- Customize a style sheet
- Launching the Web Site Designer
- Create the Web site navigation and pages
- Verify the site navigation and page templates

Create new page templates

Page templates provide an efficient method of creating a common layout for Web pages. They can be created from an existing sample page template or be user-defined and can be shared across many (or even all) pages in a Web application. If changes are required to one of the common sections of a template, it needs to be updated only in one place.

In this example, design-time page templates are applied for the JSPs, which means that the pages are built from the template in the design/build phase. Dynamic templates are the other option that would allow the page template data to be updated while the application is running.

The RedBank user interface (view) is made up of a combination of static HTML pages and dynamic JSPs. In this section we describe how to create an HTML page template (`itso_html_template.htpl`) and a JSP page template (`itso_jsp_template.jtpl`) from existing sample page templates, and then customize the page templates to be new user-defined templates.

We recommend that you create a page template prior to creating the pages in the Web Site Designer so that the page template can be specified at the time each page is created.

Create an HTML page template

To create a new user-defined HTML page template (`itso_html_template.htpl`) to be used by the RedBank HTML pages, perform the following steps:

- In the Project Explorer, expand **RAD7BankBasicWeb** → **WebContent** → **theme** and select **New** → **Web Page Template** from the context menu.
- In the **New Web Page Template** dialog, the following values should be entered:
  - File name: `itso_html_template`
  - Folder: `/RAD7BankBasicWeb/WebContent/theme`
  - Select **Design-time Template** for the type (dynamic templates are not applicable for HTML templates).
  - For template, expand **Sample Templates** → **Family A** and select `A-01_gray.htpl`. This template has two navigation bars on it, which is correct for the example.
  - Make sure **Link page to template** option is selected.

  Click **Finish** to create the template.
- You will see a dialog stating: *A page template must contain at least one Content Area which is later filled in by the pages that use the template.*

Click **OK**. This is telling you that all templates must have an area where the page specific content can be added. The Content Area is added later when the template is customized.

- Take note of the files that have been created in the theme directory as a result of creating the page template from an existing sample template:
  - **itso_html_template.jtpl**—This is the new template created with references to the sample template it was created from (A-01_gray.htpl).
  - **A-01_gray.htpl**—This is copy of the sample template.
  - **logo_gray.gif**—This is the logo for the A-01_gray.htpl template.
  - **gray.css**—This is a css file from the sample template.
  - **nav_head.html**—This is the navigation header file, referred to from the template.
  - **footer.html**—This is the footer file also referred to from the template.
- Create the application page template from the files generated.

> **Note:** The Page Template wizard works in such a way that if a new page template is created from a sample, then all page elements of the new template are read-only, except for the content area. This is not the desired behavior for our example because we want to be able to make changes to all aspects of the template. To get around this feature, we rename the sample template to our new template file name.

  - Close and delete the `itso_html_template.jtpl` file created by the Page Template wizard.
  - Rename the `A-01_gray.htpl` page template in the theme directory to `itso_html_template.htpl`. Select the file and from the context menu, select **Refactor** → **Rename,** and enter the new name and click **OK**.
  - When prompted with `Do you want to have the links to or from those files fixed accordingly?`, click **Yes**.

The RedBank HTML template now exists in the `itso_html_template.htpl` file under `WebContent/theme`.

We explain how to customize the `itso_html_template.htpl` in “Customize page templates” on page 496.

Create a JSP page template

To create a new user-defined page template (`itso_jsp_template.jtpl`) which will be used by the RedBank Web site JSPs, repeat the above steps, but referring to JSPs rather than HTML.

The underlying mechanism within Application Developer for applying a template to a JSP is very different from that used for an HTML template. Therefore, although the two templates have the same headers, footers, and icons two separate templates are required.

Complete the following steps to build the JSP template:

- Select **RAD7BankBasicWeb** → **WebContent** → **theme** and **New** → **Web Page Template**.
- In the New Page Template File dialog, enter these items and then click **Next**:
  - Folder: `/RAD7BankBasicWeb/WebContent/theme`
  - File name: `itso_jsp_template`
  - Select **Design-time Template** for the type.
  - Click **Options** and make sure that Markup Language is **HTML.**
  - Select **Link Page to template**.
  - Select **Sample Templates** → **Family A** → **JSP-A-01_gray.jtpl** and click **Finish**.
- Click **OK** to dismiss the warning dialog.
- Create the application page template from the files generated.

> **Note:** As with the HTML case, we want to be able to change all aspects of the original template, and so we copy the sample template over our new template.

  - Close and delete the `itso_jsp_template.jtpl`.
  - Rename the `JSP-A-01_gray.jtpl` page template to `itso_jsp_template.jtpl` using **Refactor** → **Rename**.

The RedBank JSP template now exists in the `itso_jsp_template.jtpl` file under `WebContent/theme`.

Customize page templates

This section describes how to make some common page template customizations, such as modifying the logo shown on the header, altering the navigation tabs, and adding a content area.

Customize the static page template

In this section, we describe how to customize the following elements of the HTML page template (`itso_html_template.htpl`) created in “Create an HTML page template” on page 494.

The following customizations are made:

- Customize the logo and title
- Customize the links on the navigation bar
- Insert a free table layout into the content area

Customize the logo and title

Modify the page template to include the ITSO logo image and ITSO RedBank title. In order to modify the logo displayed in `itso_static_template.htpl`, we must change the image that is referenced.

- Import the `itso_logo.gif` image:
  - Select **RAD7BankBasicWeb** → **WebContent** → **theme** and **Import**.
  - Select **General** → **File System** and click **Next**.
  - Enter `c:\7501code\webapps\images` in the From directory, select the **itso_logo.gif** file, and click **Finish**.
- Open the **itso_html_template.htpl** file and select the **Design** tab.
- Click on the oval logo image at the top of the page template in the Design view.
- Select the **Source** tab. The HTML code which references the image should be highlighted in the editor.
- Modify the source as follows:

```
From: <td width="150"><img border="0" width="150" height="55"
       alt="Company's LOGO"
       src="/RAD7BankBasicWeb/theme/logo_gray.gif"></td>
       <td></td>

To:    <td width="70"><img border="0" width="60" height="50"
       alt="Company's LOGO"
       src="/RAD7BankBasicWeb/theme/itso_logo.gif"></td>
       <td><H1>ITSO <Font color="red">RedBank</Font></H1></td>
```

  Also change the `<title>` tag to the text RedBank.

- Save the `itso_html_template.htpl` file and verify that the page template has the ITSO RedBank text and logo as desired by selecting the **Preview** tab.

Customize the links on the navigation bar

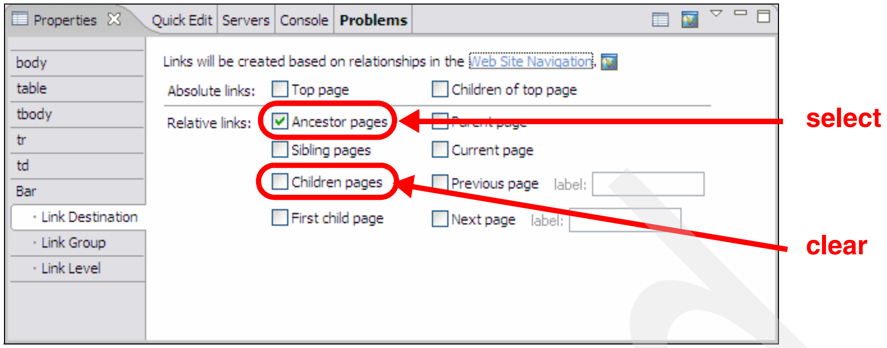
A navigation bar is a set of links to other Web pages, displayed as a horizontal or vertical bar on a Web page. When these links are applied consistently to all pages in an application, users can easily find their way around a new site. For the RedBank application, it is desirable to have a list of the children of the top page shown across the top and a list of ancestor pages, from the currently page shown up to the root, shown at the bottom.

Application Developer has a feature to make the addition of these navigation bars very easy. In the Pallet view under the Web Site Navigation drawer are a series of elements which can be dragged onto pages to provide this navigation.

In our example, the sample template already included a navigation bar to the top and bottom of the Web pages. These do however require some changes. Perform the following steps to customize the links on the navigation bar:

- Open `itso_html_template.htpl` and move to the **Design** tab.
- Click on the top grey bar, which has the text **Children of top page** written on it.
- In the Properties view (which should be showing a Bar tab selected), click on the **Link Destination** tab immediately below the Bar tab.
- Select **Children of Top Page** (should be already selected), which is what is required. This means that links to the rates, insurance, and bank pages are shown across the top bar on all pages that use this template.
- Click on the bottom grey bar which has the text **Sibling Pages** displayed on it.
- In the Properties view click on the **Link Destination** tab immediately below the Bar tab.
- Clear **Sibling pages** and select **Ancestor pages** Figure 12-16)**.** Originally this page was configured to show the siblings of the current page at the bottom, this has to be changed to show an ancestor tree back to the main index page.

> **Note:** The tools available for Web site navigation have been enhanced in V7.0. Among other features, there are now **Vertical Menu** and **Tabbed Menu** elements in the Palette view in the Web Site Navigation drawer, which allow the addition of drop-down menu style navigation bars.



Links will be created based on relationships in the Web Site Navigation.

Absolute links: □ Top page □ Children of top page

Relative links: ☑ Ancestor pages □ Parent page

□ Sibling pages □ Current page

□ Children pages □ Previous page label:

□ First child page □ Next page label:

select

clear

Figure 12-16 Link Destination tab

Insert a free table layout into the content area

Free layout tables is a feature which creates tables and cells automatically so that objects can be freely placed on the page. In the example, a free table layout is added to the page template to provide a starting point for adding unique content to the pages.

- Open `itso_html_template.htpl` and move to the **Design** tab. The current content area is the default text `Place your page content here`.
- Delete this text making sure not to delete anything else on the page.
- Click in the content area where the text used to be between the and icons.
- From the context menu select **Insert** → **Insert Free Layout Table**, and click on the cursor position.
- Drag the corner of the free layout table to increase the size. Increase the width to match the width of the navigation bar (760) and increase the height to 300 in the Properties view.
- Save and close the file.

The page template for the HTML pages is now finished (Figure 12-17) and it is possible to generate a series of HTML pages from it for the RedBank application.

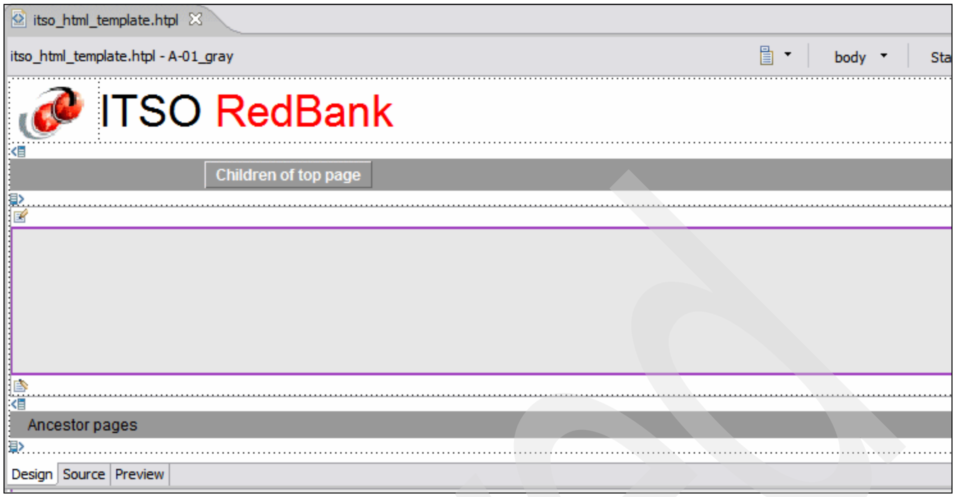


Figure 12-17 Completed HTML template (compressed)

Customize the dynamic page template

The steps to customize the dynamic page template is virtually identical to that required for the static page template.

- Open the **itso_jsp_template.jtpl** file in the Page Designer.
- Change the logo image and add the ITSO RedBank heading in the same way as for the HTML template.
- Also change the `<title>` tag to the text RedBank.
- Change the bottom grey navigation bar to show Ancestor pages (clear **Sibling pages** and select **Ancestor pages**) in the Properties view
- Add a Free Layout table into the content area in the same way as for the HTML template.
- Save the file and close it.

The page template for the JSP pages should now be created and it is possible to generate a series of JSP pages from it for the RedBank application.

> **Note:** In the files `footer.html`, `nav_head.html`, `footer.jsp` and `nav_head.jsp` there are references to an image file `c.gif`. Although this image is only used for spacing purposes, the URL to reference this image is actually wrong and will cause problems for “Web application testing” on page 1031 in Chapter 21, “Test using JUnit”.
>
> To fix this problem, copy the `c.gif` file from `WebContent\theme` to `WebContent`.

Customize a style sheet

Style sheets can be created when a Web Project is created (by selecting the **Default style sheet (CSS File)** option from the Project Facets dialog), when a page template is created from a sample, or at any time by launching CSS File creation wizard.

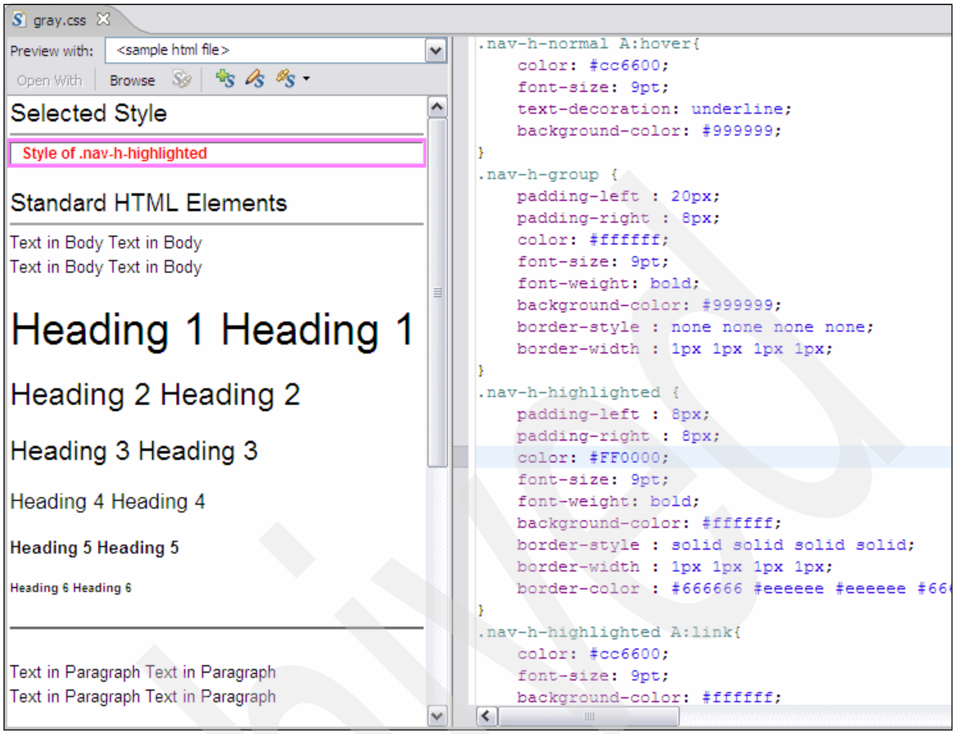
In the RedBank example a style sheet named `gray.css` was created as part of the process of creating the page templates from the samples. Both the HTML (`itso_html_template.htpl`) and the JSP (`itso_jsp_template.jhtpl`) templates reference the `gray.css` style sheet for them to have a common appearance of fonts and colors for the pages.

In the following example, we customize the colors used on the navigation bar links, and supporting variations such as hover links. By default the link text in the navigation bar is orange (`cc6600`). We will customize this to be red (FF0000). To find the hexadecimal code for a particular color a resource is supplied in the references section at the end of this chapter.

To customize the `gray.css` style sheet, do these steps:

- Open the **gray.css** file in the CSS Designer (Figure 12-18).

  By selecting the text style **`.nav-h-highlighted`** in the right-hand pane (scroll down to locate the style, or find the style in the Styles view at the bottom left) , the text in the left-hand pane is displayed and highlighted. This makes it easy to change the settings and see the change immediately.



```
.nav-h-normal A:hover{
    color: #cc6600;
    font-size: 9pt;
    text-decoration: underline;
    background-color: #999999;
}
.nav-h-group {
    padding-left : 20px;
    padding-right : 8px;
    color: #ffffff;
    font-size: 9pt;
    font-weight: bold;
    background-color: #999999;
    border-style : none none none none;
    border-width : 1px 1px 1px 1px;
}
.nav-h-highlighted {
    padding-left : 8px;
    padding-right : 8px;
    color: #FF0000;
    font-size: 9pt;
    font-weight: bold;
    background-color: #ffffff;
    border-style : solid solid solid solid;
    border-width : 1px 1px 1px 1px;
    border-color : #666666 #eeeeee #eeeeee #66
}
.nav-h-highlighted A:link{
    color: #cc6600;
    font-size: 9pt;
    background-color: #ffffff;
```

Figure 12-18 CSS Designer: gray.css

- Change the Hex HTML color code for all nav-h-highlighted entries from `color: #cc6600;` (orange) to `color: #FF0000;` (red).
- Customize the footer highlighted link text. Locate the `.nav-f-highlighted` style and change the color from `#ff6600` (orange) to `#ff0000` (red).
- Save the file.

Now when a page is navigated to, the header and footer tabs will show the current the currently viewed page in red rather than orange.

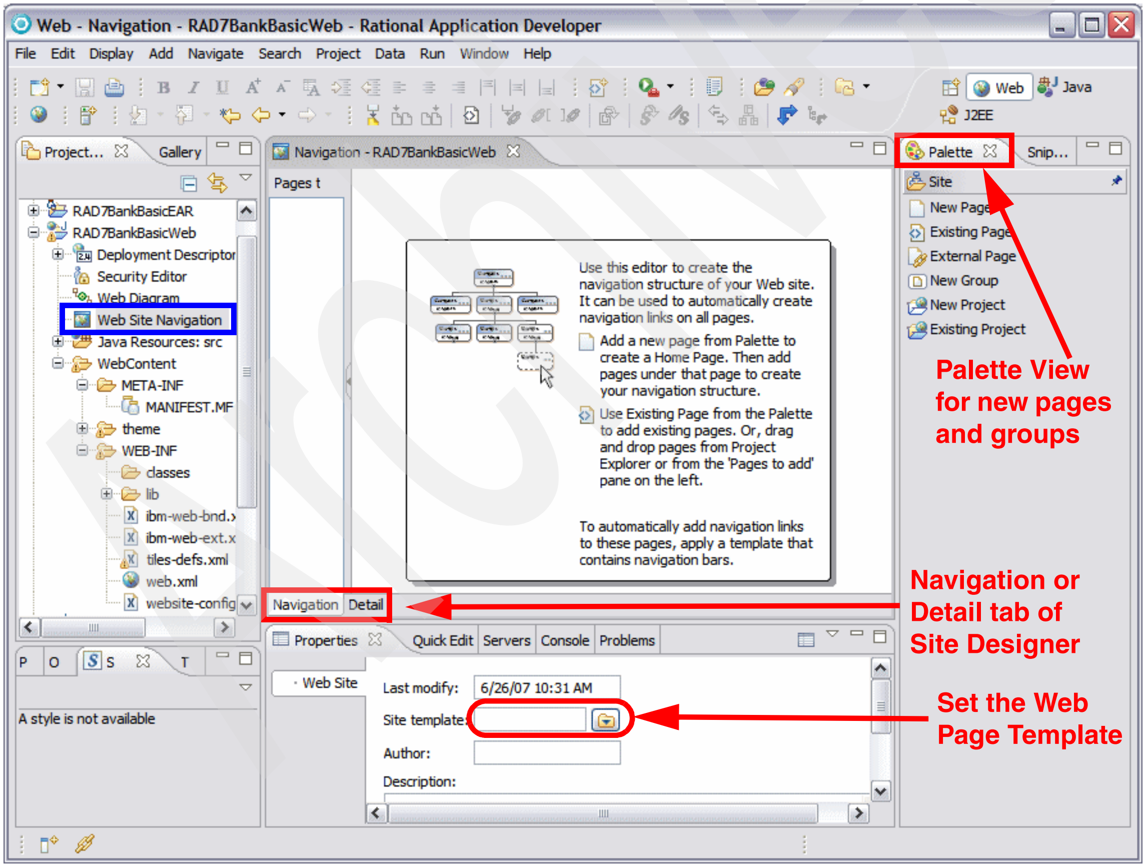
Obviously any number of changes can be applied to the style sheets to change the look, feel, and color of the application.

Launching the Web Site Designer

To launch Web Site Designer from the Project Explorer view, double-click **Web Site Navigation**.

When the Web Site Designer view opens (Figure 12-19), take note of the following features:

- **Navigation and Detail tab**—The **Navigation** tab (the default tab) allows a user to visually design the layout of the site and the **Detail** tab is used to define the fine details for each page including the ID, Navigation Label, File Path, File Name /URL, Servlet URL, and Page Title.
- **Palette view**—Selections from the Palette can be dragged to the Navigation page. For example, the **New Page** icon can be dragged from the Palette to create a new page or the **New Group** icon can be dragged onto the page, which can be used to logically organize pages in a hierarchical grouping.
- **Site template**—In the Properties view you can select a site template to define the appearance of the site. This can be a sample page template included with Application Developer, or a user-defined page template. New pages built from the Web Site Designer will use this template by default.



Use this editor to create the navigation structure of your Web site. It can be used to automatically create navigation links on all pages.

Add a new page from Palette to create a Home Page. Then add pages under that page to create your navigation structure.

Use Existing Page from the Palette to add existing pages. Or, drag and drop pages from Project Explorer or from the 'Pages to add' pane on the left.

To automatically add navigation links to these pages, apply a template that contains navigation bars.

Last modify: 6/26/07 10:31 AM

Site template:

Author:

Description:

A style is not available

Palette View for new pages and groups

Navigation or Detail tab of Site Designer

Set the Web Page Template

Figure 12-19 Web Site Designer: Navigation view

Create the Web site navigation and pages

In this section, we use the Web Site Designer to construct the page navigation and page skeletons for the RedBank Web site. At the end we will have a working skeleton of the application, where we can navigate from page to page using the tabs below the page header and the ancestor tabs. The pages will not have any actual content but this is added later.

We will create the navigation pages and corresponding HTML or JSP page for each of the following pages:

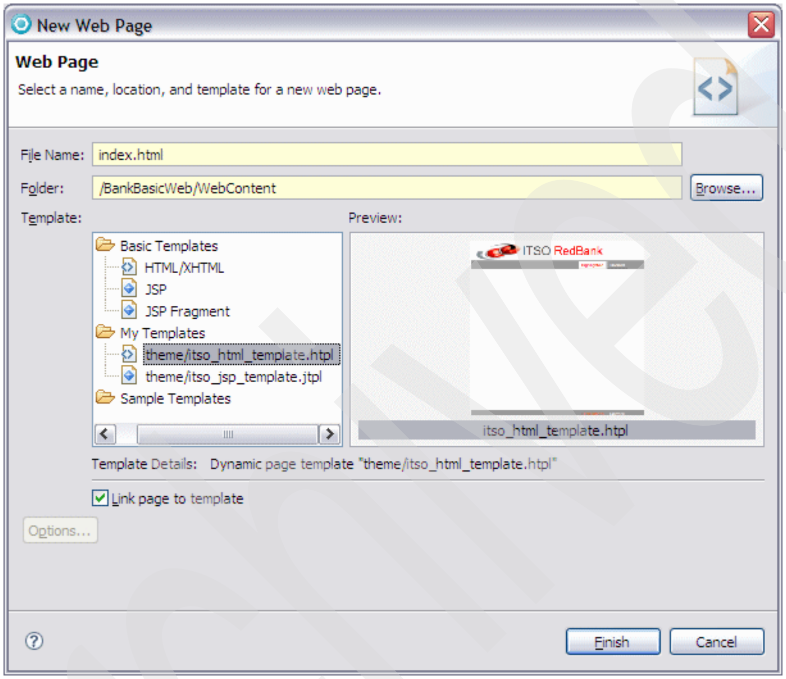
Navigation label	HTML or JSP file
itsohome	`index.html`
rates	`rates.html`
insurance	`insurance.html`
redbank	`redbank.html`
listaccounts	`listAccounts.jsp`
accountdetails	`accountDetails,jsp`
listtransactions	`listTransactions.jsp`
showexception	`showException.jsp`

To define the site navigation and create the HTML and JSP page skeletons, perform the following steps in the Web Site Designer:

- Launch the **Web Site Designer** from the Project Explorer.
- Create the root static navigation page (`index.html`):
  - Select **New Page** from the Palette and click on the Navigation page to add the first page.
  - After the new page is added, the navigation label can be entered in the Navigation page or in the Properties view under Navigation label. Enter `itsohome`. Save the Navigation page.
  - Double-click the **itsohome** navigation page to create the HTML file associated with the index page.
  - In the New Web Page dialog, enter the following items (Figure 12-20).
    - File name: `index.html`

      By default, a Web Server will look for `index.html` (or `index.htm`) when a Web project is run. Although this behavior can be changed, we recommend that you use `index.html` as the top level page name.

- Folder: `/RAD7BankBasicWeb/WebContent`
- Template: Expand **My Templates** and select `theme/itso_html_template.htpl`
- Make sure **Link page to template** is selected.
- Click **Finish**.



New Web Page

Web Page

Select a name, location, and template for a new web page.

File Name: index.html

Folder: /BankBasicWeb/WebContent

Template:

- Basic Templates
  - HTML/XHTML
  - JSP
  - JSP Fragment
- My Templates
  - theme/itso_html_template.htpl
  - theme/itso_jsp_template.jtpl
- Sample Templates

Preview:

Template Details: Dynamic page template "theme/itso_html_template.htpl"

☑ Link page to template

Figure 12-20 Create the itsohome: index.html page

- Define the navigation root.

  In our example, itsohome (`index.html`) is the *navigation root*. By default, when a page is created it is set as a *navigation candidate*, which is the desired format for all other pages.

  To make this change, select **itsohome** on the Web Navigation Diagram, and from the context menu select **Set Navigation** → **Set Navigation Root**.

- Add three static pages as children of itsohome. From the **itsohome** context menu on Page Designer, select **Add** → **New Page** → **As Child**. Perform this three times and type the navigation labels as rates, redbank, and insurance.
- Double-click each page and type the file names as `rates.html`, `redbank.html`, and `insurance.html`. Leave the template `itso_html_template.htpl` selected.
- Create a new group named RedBank.

Page groups are used to logically build or organize pages into a movable block of related pages. In the RedBank example the JSPs under `redbank.html` are grouped together.

  - Select the **redbank** page, right-click, and select **Add New Group** → **As Child**. A **Group** box appears below the redbank page.
  - Select the group and in the **Properties** view, **Group** tab, type **RedBank** as the group name.
  - Save the Navigation page.

- Create the dynamic pages (JSPs) `listaccounts`, `accountdetails`, `listtransactions`, and `showexception`.
  - From the context menu of the RedBank group, select **Add** → **New Page** → **Into Group**.
  - After the New Page is added, notice you can type the navigation label in the Navigation page or in the Properties view as Navigation label. For the first page, enter `listaccounts`.
  - Double-click the **listaccounts** box to create the JSP file associated with the navigation label.
  - In the New Web Page dialog, enter the following items and then click **Next**:
    - File name: `listAccounts.jsp`
    - Template: Select **My Templates** → `theme/itso_jsp_template.jtpl`
    - Make sure **Link Page to template** is selected.

    Note that when creating JSPs the **Options** button is active. This provides options to specify the Document Markup properties, to have the wizard automatically generate servlet stubs methods and to add extra fields to the web.xml file. For this example, the default options are fine.
  - Click **Finish** to create the page.
- Repeat the steps to add the `accountdetails` (`accountDetails.jsp`), `listtransactions` (`listTransactions.jsp`), and `showexception` (`showException.jsp`) pages. Note that the spelling and capitalization of the JSP file names must be perfect.
- The **showexception** page only appears when there is a problem and is not part of the standard navigation. Therefore, select the **showexception** page, right-click, and select **Set Navigation** → **Show in Navigation** (this clears the option).
- Change the `<title>` tag through the Navigation editor, **Detail** tab, in the Page Title column of each JSP to List Accounts, Account Detail, List Transactions, and Show Exception. This action changes the `<title>` tag in the source code of each JSP.

- Save the Navigation page.

When done adding the navigation, HTML, and JSP pages, the Navigation page should look like Figure 12-21.

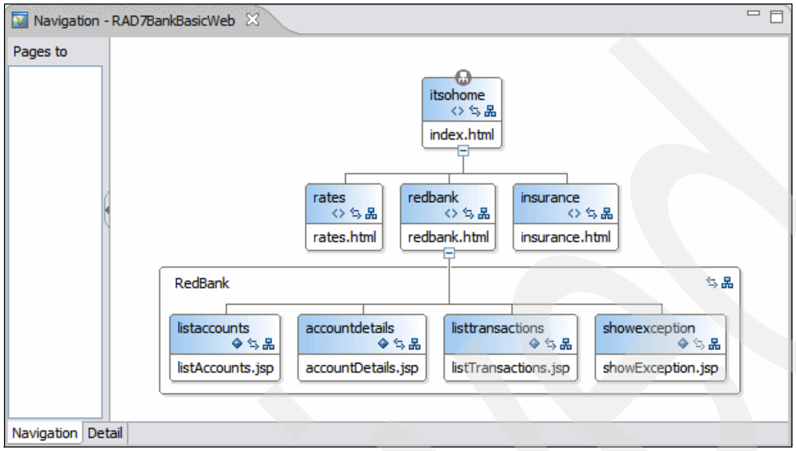


Figure 12-21 Navigation page after adding pages

Verify the site navigation and page templates

At this stage although the pages have no content, we can verify that the page templates look as expected and that the navigation links in the header and footer navigation bars work as required:

- Start the WebSphere Application Server V6.1 Test Environment in the Servers view if it is not running.
- In the Project Explorer right-click **RAD7BankBasicWeb**, and select **Run As** → **Run on Server**.
- In the Define a New Server dialog, select **Choose an existing server** and **WebSphere Application Server v6.1**. Click **Finish**.
- A browser pane starts at the index page and the user can click on the tabs for rates, redbank, and insurance to move between these pages (Figure 12-22).

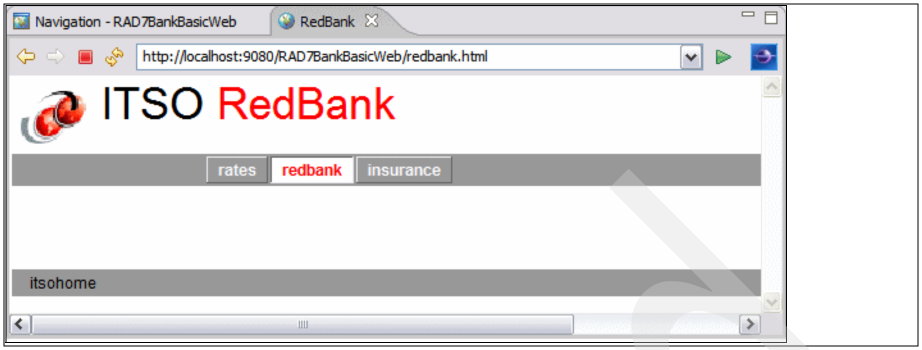

Navigation - RAD7BankBasicWeb
RedBank
http://localhost:9080/RAD7BankBasicWeb/redbank.html
ITSO RedBank
rates
redbank
insurance
itsohome

Figure 12-22 ITSO RedBank Web site

- To verify the JSPs, type this URL directly into the browser:

  ```
  http://localhost:9080/RAD7BankBasicWeb/listAccounts.jsp
  ```

  The list accounts page is shown. Note that the Ancestor trail at the bottom of the page shows **itsohome** and **redbank**, which is the correct ancestor tree for this page. The same test can be performed for the other JSPs.

- To remove the project from the test server, in the Servers view right-click **WebSphere Application Server v6.1**, select **Add Remove Projects**, and remove `RAD7BankBasicEAR`.

  Alternatively, expand **WebSphere Application Server v6.1**, right-click the **RAD7BankBasicEAR** project and select **Remove**.

Developing the static Web resources

In this section we create the content for the four static pages of our sample with the objective of highlighting some of the features of Page Designer. Page Designer facilitates the building of HTML pages by allowing the user to add HTML elements from the Pallet view using drag and drop. HTML fragments can also be imported directly into the source tab and this options is also demonstrated.

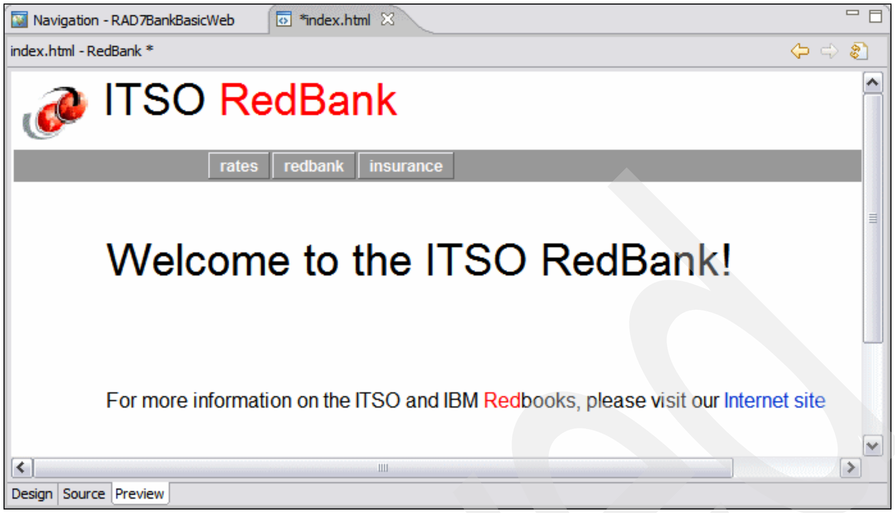
The topics in this section are as follows:

- Create the index.html page content (text, links)
- Create the rates.html page content (tables)
- Import the insurance.html page contents
- Import the redbank.html page contents

Create the index.html page content (text, links)

The RedBank home page is index.html. The links to the child pages are included as part of the header and footer of the our page template. In the following example, we describe how to add static text to the page, and add a link to the page to the IBM Redbooks Web site.

- Open the **index.html** file in Page Designer.
- Select the **Design** tab.
- Insert the welcome message text.
  - In the Context Area, right click, and select **Insert** → **Paragraph** → **Heading 1**.
  - Click the cursor in the table area of the table within the content where you want to insert the heading.
  - You will see a cell marked on the page. Resize the cell as desired.
  - Enter the text `Welcome to the ITSO RedBank!`.
- Insert a Link to the IBM Redbooks Web site.
  - Make sure the cursor is not in the H1 element created above. Click elsewhere in the Content area if required.
  - From the menu bar, select **Insert** → **Paragraph** → **Normal**.
  - Click the cursor in the content area immediately below the welcome message.
  - You will see a cell marked on the page. Resize the cell as desired.
  - Enter the text `For more information on the ITSO and IBM Redbooks, please visit our Internet site` into the new area.
  - Highlight the text **Internet site**, right-click and select **Insert Link**.
  - In the Insert Link dialog, select **HTTP**, enter `http://www.ibm.com/redbooks` in the URL field, and click **OK**.
- Customize the text font face, size, and color. This is done through the properties tab.
  - Select the word **Red** from Redbooks from the text created in the previous step (use the keyboard Shift and arrow keys).
  - In the **Text** tab in the Properties view select the color **Red** to make this partial word stand out a little.
- Save the page.
- Select the **Preview** tab and the page is displayed (Figure 12-23).



Welcome to the ITSO RedBank!

For more information on the ITSO and IBM Redbooks, please visit our Internet site

Figure 12-23 Preview of index.html

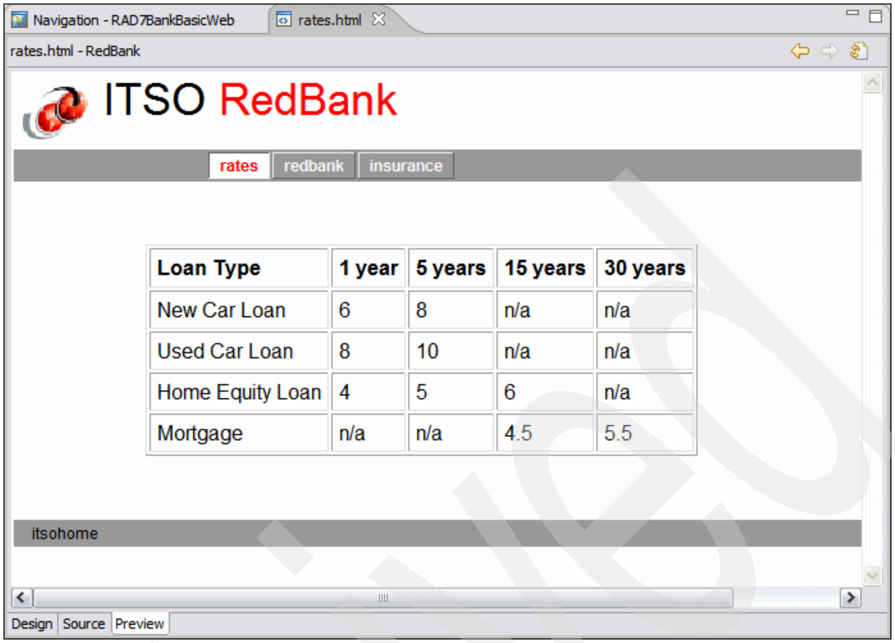
Create the rates.html page content (tables)

In this example we demonstrate how to add a static table containing interest rates using the Page Designer.

- Open the **rates.html** file in Page Designer and select the **Design** tab.
- Expand **HTML Tags** in the Palette.
- Select and drag a **Table** from the Palette to the content area.
- In the Insert Table dialog, enter 5 for Rows and 5 for Columns and also 5 for Padding inside cells, then click **OK**.
- Resize the table as desired.
- Enter the descriptions and rates (as seen in Figure 12-24) into the table.
- Select the heading cells and in the Properties view, and click the Styles icon to set the Font style weight to bold.

> **Note:** Additional table rows and columns can be added and deleted with the Table menu option.

- Save the page.
- Select the **Preview** tab and the page is displayed (Figure 12-24).



Loan Type	1 year	5 years	15 years	30 years
New Car Loan	6	8	n/a	n/a
Used Car Loan	8	10	n/a	n/a
Home Equity Loan	4	5	6	n/a
Mortgage	n/a	n/a	4.5	5.5

Figure 12-24 Preview the rates.html page

Import the insurance.html page contents

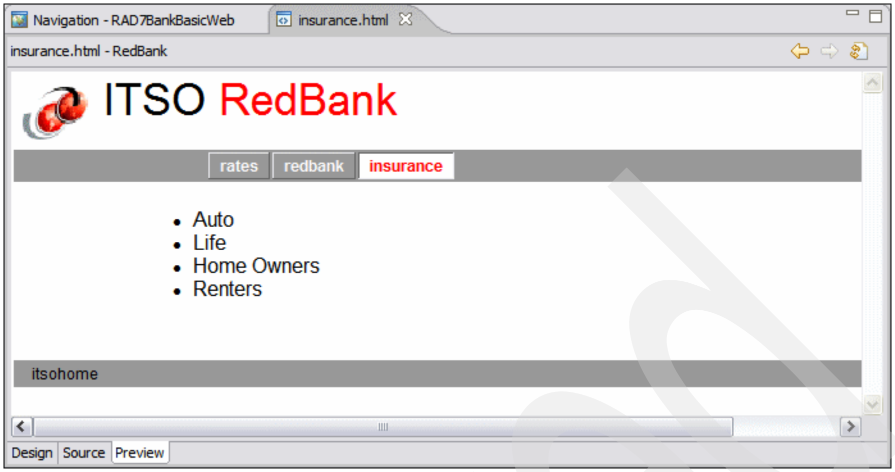
In this section, we import the body of the `insurance.html` file:

- Locate the file `c:\7501code\webapps\html\SnippetForInsuranceHTML.txt` and open it in a simple text editor (for example Notepad).
- Open `insurance.html` in Page Designer and select the **Source** tab.
- Select all the content between the tags:

```
<!-- tpl:put name="bodyarea" -->
<!-- /tpl:put --></td>
```

  Note that this area is differentiated from the rest of the content by being in color rather than gray. Delete the text.

- Insert the text from the `SnippetForInsuranceHTML.txt` file.
- Save the file and switch to the **Preview** tab (Figure 12-25).



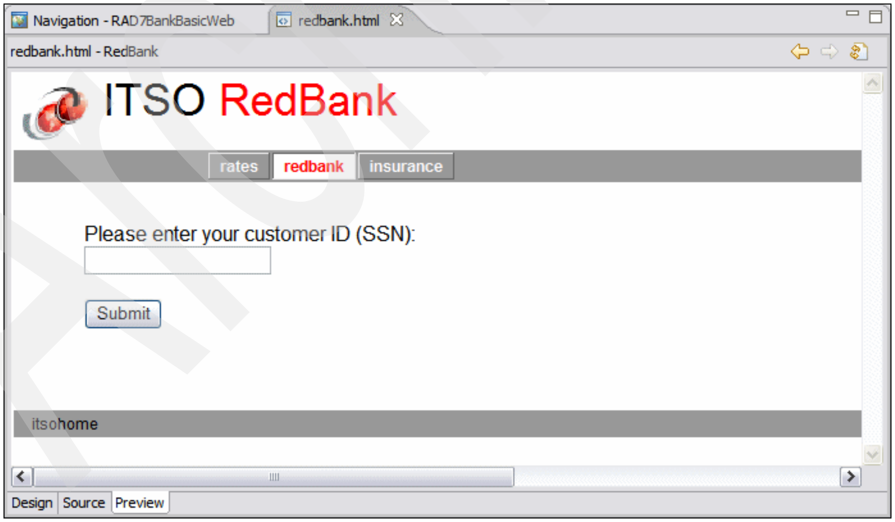
- Auto
- Life
- Home Owners
- Renters

Figure 12-25 Preview of Insurance.html

Import the redbank.html page contents

Repeat the import of the body of the `redbank.html` file:

- ► Locate the file `c:\7501code\webapps\html\SnippetForBankHTML.txt`.
- ► Replace existing content area text with the text from the snippet.
- ► Switch to the **Preview** tab (Figure 12-26).



Please enter your customer ID (SSN):

Figure 12-26 Preview of redbank.html

The static HTML pages for the RedBank application are now complete.

Developing the dynamic Web resources

In addition to the tools created for building HTML content and designing the flow and navigation in a Web application. Application Developer also provides several wizards to help you quickly build JavaServer Pages (JSPs) and Java servlets, even if you are not an expert programmer. The products of these wizards can be used as-is, or modified to fit specific needs.

The wizards not only support the creation of servlets and JSPs, they also compile the Java code and store the class files in the correct folders for publishing to your application servers. Finally, as the wizards generate project resources, the deployment descriptor file (web.xml) is updated automatically with the appropriate configuration information for the servlets that are created.

In the previous section we described how to create each of the static Web pages. In this section we demonstrate the process of creating and working with servlets. The example servlets are first built using the wizards, then the code contents are imported from the sample solution. In the next section “Working with JSPs” on page 521, the JSP pages are created which invoke the logic in these servlets.

Working with servlets

As described in the “Introduction to J2EE Web applications” on page 466, servlets are flexible and scalable server-side Java components based on the Sun Microsystems Java Servlet API, as defined in the Sun Microsystems Java Servlet Specification. For J2EE Version 1.4, the supported API is Servlet 2.4, which is used by Application Developer V7.0.

Servlets generate dynamic content by responding to Web client requests. When an HTTP request is received by the application server, the Web Server determines, based on the request URI, which servlet is responsible for answering that request and forwards the request to that servlet. The servlet then performs its logic and builds the response HTML that is returned back to the Web client, or forwards the control to a JSP.

Application Developer provides the features to make servlets easy to develop and integrate into your Web application. From the Workbench it is possible to develop, debug, and deploy servlets. It is also possible to set breakpoints within servlets, and step through the code in a debugger, and finally any changes made are dynamically folded into the running Web application, without having to restart the server each time.

In the sections that follow, we implement the `ListAccounts`, `UpdateCustomer`, `AccountDetails`, and `Logout` servlets. Then the command or action pattern is applied in Java to implement the `PerformTransaction` servlet.

Adding RAD7Java as a Web Library project

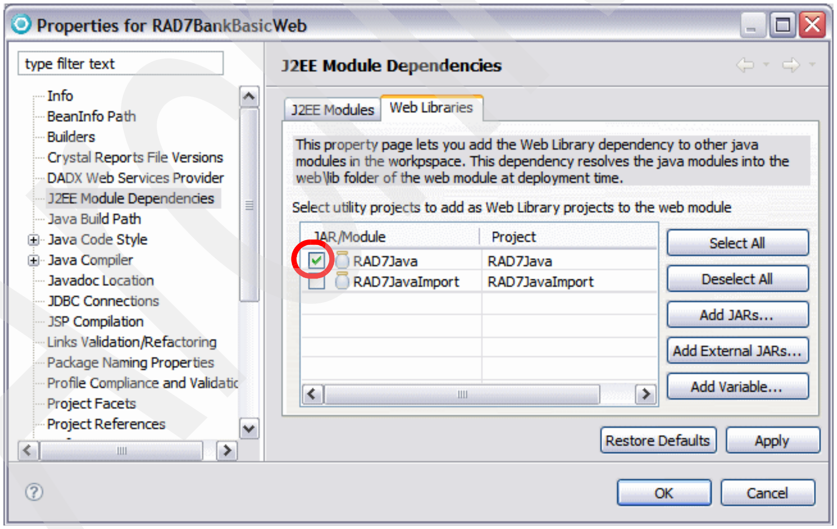
Before the implementation of the servlet classes can proceed, we must add a reference from the **RAD7BankBasicWeb** project to the **RAD7Java** project, because the servlets call the methods from classes in this project.

This uses a facility within Application Developer known as **Web Library** projects, where a Java project can be associated with a Web project so that the Java resources in the Web project can call those in the Java project. Also, when the WAR file is built, a JAR file representing the Java project is automatically added to the `WEB-INF/lib` directory.

> **Note:** In Application Developer V6.0, this feature used to exist on the deployment descriptor for a Web project. For V7.0, this has moved to the project properties.

To add `RAD7Java` as a Web Library project, perform the following steps:

- Select the **RAD7BankBasicWeb** project, select **Properties**, and select **J2EE Module Dependencies**.
- Select the **Web Libraries** tab.
- Select **RAD7Java** and click **OK** (Figure 12-27).



Properties for RAD7BankBasicWeb

type filter text

- Info
- BeanInfo Path
- Builders
- Crystal Reports File Versions
- DADX Web Services Provider
- J2EE Module Dependencies
- Java Build Path
- Java Code Style
- Java Compiler
- Javadoc Location
- JDBC Connections
- JSP Compilation
- Links Validation/Refactoring
- Package Naming Properties
- Profile Compliance and Validatic
- Project Facets
- Project References

J2EE Module Dependencies

J2EE Modules | Web Libraries

This property page lets you add the Web Library dependency to other java modules in the workpspace. This dependency resolves the java modules into the web\lib folder of the web module at deployment time.

Select utility projects to add as Web Library projects to the web module

JAR/Module	Project
☑ RAD7Java	RAD7Java
☐ RAD7JavaImport	RAD7JavaImport

Figure 12-27 J2EE Module Dependencies dialog

The classes within **RAD7Java** can now be accessed by Java code in **RAD7BankBasicWeb**.

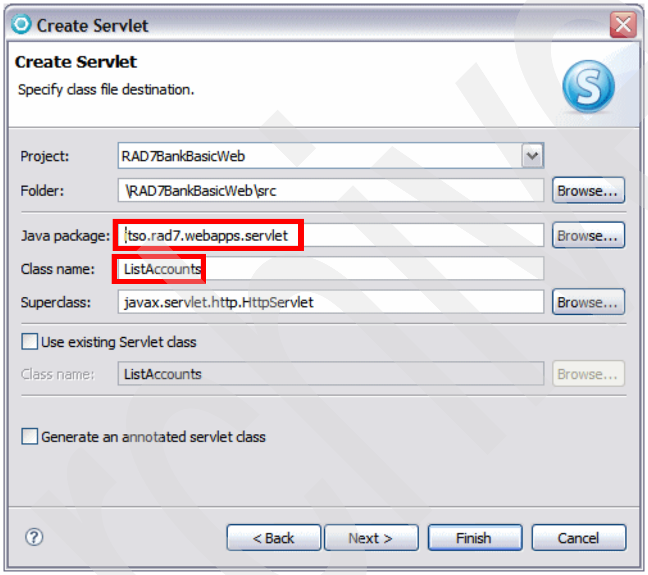
Adding the ListAccounts servlet to the Web project

Application Developer provides a servlet wizard to assist you in adding servlets to your Web applications. Follow these steps:

- Select **File** → **New** → **Other** → **Web** → **Servlet** and click **Next**.

> **Tip:** The Create Servlet wizard can also be accessed by right-clicking the project and selecting **New** → **Servlet**.

- The first page of the Create Servlet wizard opens. Enter `ListAccounts` as the class name and `itso.rad7.webapps.servlet` as the package (Figure 12-28). Click **Next**.



Create Servlet

Create Servlet

Specify class file destination.

Project: RAD7BankBasicWeb

Folder: \RAD7BankBasicWeb\src

Java package: itso.rad7.webapps.servlet

Class name: ListAccounts

Superclass: javax.servlet.http.HttpServlet

☐ Use existing Servlet class

Class name: ListAccounts

☐ Generate an annotated servlet class

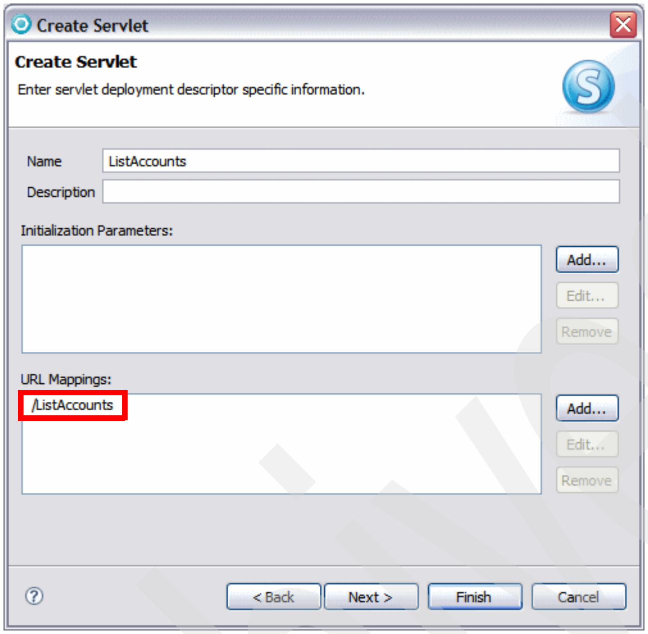
Figure 12-28 New Servlet wizard (1)

- The second page (Figure 12-29) provides space for the name and description of the new servlet.

  The page also allows the addition of servlet initialization parameters, which are used to parameterize a servlet. Servlet initialization parameters can be changed at runtime from within the WebSphere Application Server administrative console.

  The wizard will automatically generate the URL mapping `/ListAccounts` for the new servlet. If a different, or additional URL mappings are required, these can be added here.

In our sample, we do not require additional URL mappings or initialization parameters. Click **Next**.



Create Servlet

Create Servlet

Enter servlet deployment descriptor specific information.

Name: ListAccounts

Description:

Initialization Parameters:

URL Mappings:

/ListAccounts

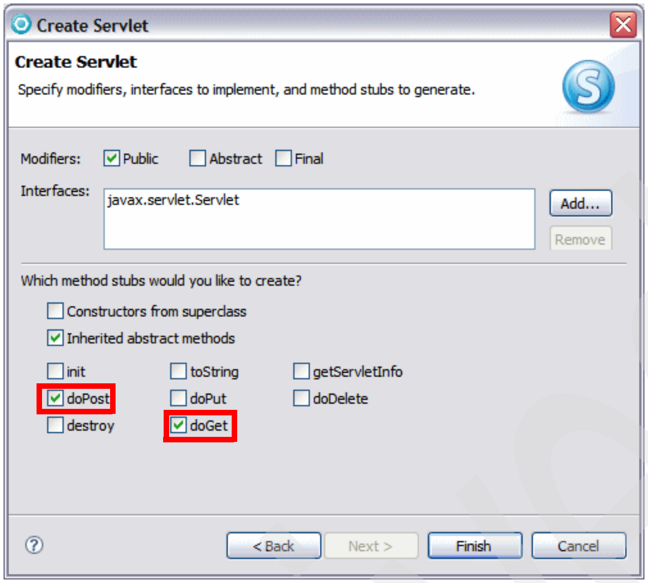
Figure 12-29 New Servlet wizard (2)

- The third and final page gives the option to have the wizard create stubs methods for methods available from the `HttpServlet` interface. The `init` method is called at start-up and `destroy` is called at shutdown. The `doPost`, `doGet`, `doPut`, and `doDelete` methods are called when a http request is received for this servlet. All of the do methods have two parameters, namely a `HttpServletRequest` and a `HttpServletResponse`. It is the job of these methods is to extract the pertinent details from the request and populate the response object.

  For the `ListAccounts` servlet, only `doGet` and `doPost` should be selected. Usually, HTTP gets are used with direct links, when no information has to be sent to the server. HTTP posts are typically used when information in a form has to be sent to the server.

  There is no initialization required for our new servlet and so the `init` method is not selected.

  Ensure that the options are selected as shown in Figure 12-30, and click **Finish**.



Create Servlet

Create Servlet

Specify modifiers, interfaces to implement, and method stubs to generate.

Modifiers: ☑ Public □ Abstract □ Final

Interfaces: javax.servlet.Servlet

Which method stubs would you like to create?

□ Constructors from superclass

☑ Inherited abstract methods

□ init □ toString □ getServletInfo

☑ doPost □ doPut □ doDelete

□ destroy ☑ doGet

Figure 12-30 New Servlet wizard page (3)

The servlet is generated and added to the project. The source code can be found in the Java Resources folder of the project, while the configuration for the servlet is found in Servlets tab of the Web deployment descriptor.

Now open the deployment descriptor for the RAD7BankBasicWeb (immediately under the project in the Project Explorer), and click on the **Servlets** tab. Note that the `ListAccounts` servlet is listed.

Implementing the ListAccounts servlet

A skeleton servlet now exists but does not perform any actions when it is invoked. We now have to add code to the servlet in order to implement the required behavior.

- The **ListAccounts.java** code of the servlet is already opened. Otherwise open the code from the `itso.rad7.webapps.servlet` package.
- Locate the file:

  `c:\7501code\webapps\itso.rad7.webapps.servlet\ListAccounts.java`
- Replace the contents of the `ListAccounts.java` with the sample file. This should compile successfully with no errors.
- Look at the source code for the `ListAccounts.java` servlet. This class implements the `doPost` and `doGet` methods, both of which call the `performTask` method.

  The `performTask` method does the following tasks:

  - First the method deals with the HTTP request parameters supplied in the request. This servlet expects to either receive a parameter called `customerNumber` or none at all. If the parameter is passed, we store it in the HTTP session for future use. If it is not passed, we look for it in the HTTP session, because it might have been stored there earlier.
  - Next the method implements the control logic. Access to the Bank facade is obtained through the `ITSOBank.getBank()` method and it is used to get the `customer` object and the array of `accounts` for that customer.
  - The third section adds the customer and account variables to the `HttpRequest` object so that the presentation renderer (`listAccounts.jsp`) gets the parameters it requires to perform its job. The control of processing the request is then passed through to `listAccounts.jsp` using the `RequestDispatcher.forward` method, which builds the response to be shown on the browser.
  - The final part of the method is the error handler. If an exception is thrown in the previous code, the catch block will ensure that control is passed to the `showException.jsp` page.

  See Figure 12-8 on page 485 for a sequence diagram of the design of this class.

- The `ListAccounts` servlet is now complete. The changes should be saved and the source editor closed.

Implementing the UpdateCustomer servlet

The `UpdateCustomer` servlet is used for updating the customer information and is invoked from the `ListAccounts` JSP servlet.

The servlet requires that the SSN of the customer that is to be updated is already placed on the session (as should be done in the `ListAccounts` servlet). It extracts the **title**, **firstName** and **lastName** parameters from the `HttpRequest` object, calls the `bank.getCustomer(String customerNumber)` method and then uses the simple setters on the `Customer` class to update the details.

Follow the procedures described in “Adding the ListAccounts servlet to the Web project” on page 515 and “Implementing the ListAccounts servlet” on page 517 for building the servlet, including the `doGet` and `doPost` methods. The name of this servlet class should be `UpdateCustomer`.

The code to use for this class is in:

```
c:\7501code\webapps\itso.rad7.webapps.servlet\UpdateCustomer.java
```

Implementing the AccountDetails servlet

The AccountDetails servlet retrieves the account details and forwards control to the accountDetails.jsp page to show these details. The servlet expects the parameter accountId in the request which specifies the account for which data should be shown. The servlet then calls the bank.getAccount(..) method which returns an Account object and adds it as a variable to the request. It then uses the RequestDispatcher to forward the request onto the accountDetails.jsp.

> **Note:** A real-life implementation would perform security and authorization, where the current user has the required access rights to the requested account. This can be implemented using the Security Editor tool as described in “Security Editor” on page 478.

Follow the procedures described in “Adding the ListAccounts servlet to the Web project” on page 515 and “Implementing the ListAccounts servlet” on page 517 for building the servlet. The name of the class should be AccountDetails.

The code to use for this class is in:

```
c:\7501code\webapps\itso.rad7.webapps.servlet\AccountDetails.java
```

Implementing the Logout servlet

The Logout servlet is used for logging the customer off from the RedBank application. The servlet requires no parameters, and the only logic performed in the servlet is to remove the SSN from the session, simulating a log off action. This is done by calling the session.removeAttribute and session.invalidate methods. Finally it uses the RequestDispatcher class to forward the browser to the index.html page.

Follow the procedures described in “Adding the ListAccounts servlet to the Web project” on page 515 and “Implementing the ListAccounts servlet” on page 517 for building the servlet. The name of the class should be Logout.

The code to use for this class is in:

```
c:\7501code\webapps\itso.rad7.webapps.servlet\Logout.java
```

Implementing the PerformTransaction command classes

In the PerformTransaction servlet, a command design pattern is used to implement it as a front controller class that forwards control to one of the four command objects namely Deposit, Withdraw, Transfer, and ListTransactions.

The design for this was presented in “Controller layer” on page 484, and this implementation is based on the sequence diagram (Figure 12-9 on page 486).

Import the code for the commands package. The source is located in the folder:

```
C:\7501code\webapps\itso.rad7.webapps.command
```

- Create the package `itso.rad7.webapps.command`. In the Project Explorer, right-click the **Java Resources: src** folder and select **New** → **Package**.
- Enter `itso.rad7.webapps.command` as the package name and click **Finish**.
- From the context menu of the new package, select **Import**, then **General** → **File system**. Click **Next**.
- Click **Browse** and navigate to the folder:

  ```
  C:\7501code\webapps\itso.rad7.webapps.command
  ```

- Click **OK**.
- Select the five Java files and click **Finish**:

  ```
  Command.java
  DepositCommand.java
  ListTransactionsCommand.java
  TransferCommand.java
  WithdrawCommand.java
  ```

The command classes perform the operations on the RedBank model classes (from the `RAD7Java` project) through the Bank facade. They also return the file-name for the next screen to be shown after the command has been executed.

Implementing the PerformTransaction servlet

Now that all the commands for the `PerformTransaction` framework have been realized, the `PerformTransaction` servlet can be created. The servlet uses the value of the transaction request parameter to determine what command to execute.

The Create Servlet wizard can be used to create a servlet named `PerformTransaction`. The servlet class should be placed in the package `itso.rad7.webapps.servlet`.

Follow the procedures described in “Implementing the ListAccounts servlet” on page 517 for preparing the servlet, including the doGet and doPost methods. The name of the class should be `PerformTransaction`.

The code to use for this class is in:

```
c:\7501code\webapps\itso.rad7.webapps.servlet\PerformTransaction.java
```

`PerformTransaction` stores a `HashMap` of the action strings (deposit, withdraw, transfer, and list) to instances of Command classes. Both the `doGet` and `doPost` methods call `performTask`. In the `performTask` method the `execute` method is called on the appropriate `Command` class that performs the transaction on the

Bank classes. After the execute is completed, the `getForwardView` method is called on the `Command` class, which returns the next page to display, and `PerformTransaction` uses the `RequestDispatcher` to forward the request to the next page.

> **Note about refactoring servlets:** It is possible to rename an existing servlet by selecting **Refactor** → **Rename** from the context menu. However, the default options given on the renaming dialog do not update references to a servlet in the `web.xml` file. When renaming a servlet, select **Update fully qualified names in non-Java text files** option and enter `*.xml` in the File name patterns field. This will then inform the user that `web.xml` is updated with the renaming and give them the opportunity to cancel the operation.

Working with JSPs

JSP files are edited in Page Designer, the same editor used to edit the HTML page. When working with a JSP page in Page Designer, the Palette view has additional elements (JSP tags) that can be used, such as JavaBean references, Java Standard Template Language (JSTL) tags, and scriptlets containing Java code.

In this section, the implementation of `listAccounts.jsp` is described in detail and the other JSPs (`accountDetails.jsp`, `listTransactions.jsp`, and `showException.jsp`) are imported from the solution.

Implement the List Accounts JSP

Customizing a JSP file by adding static content is done in the Page Designer tool in the same way that an HTML file can be edited. It is also possible to add the standard JSP declarations, scriptlets, expressions, and tags, or any other custom tag developed or retrieved from the internet.

In this example, the `listAccounts.jsp` file is built up using page data variables for customer and accounts (and array of Account classes for that customer). These variables are added to the page by the `ListAccounts` servlet and are accessible to the Java code and tags used in the JSP.

To complete the body of the `listAccounts.jsp` file, perform the following steps:

- Open the `listAccounts.jsp` file in Page Designer and select the **Design** tab.
- Add the customer and accounts variables to the page data meta information in the Page Data View (by default on the bottom left of the window). These variables are added to the request object in the `ListAccounts` servlet, as discussed in "Implementing the ListAccounts servlet" on page 517. Page Designer needs to be aware of these variables:

- In the Page Data view expand **Scripting Variables**, right-click **requestScope**, and select **New → Request Scope Variable**.
- In the Add Request Scope Variable dialog, enter the following information and click **OK**:
  - Variable name: `customer`
  - Type: `itso.rad7.bank.model.Customer`

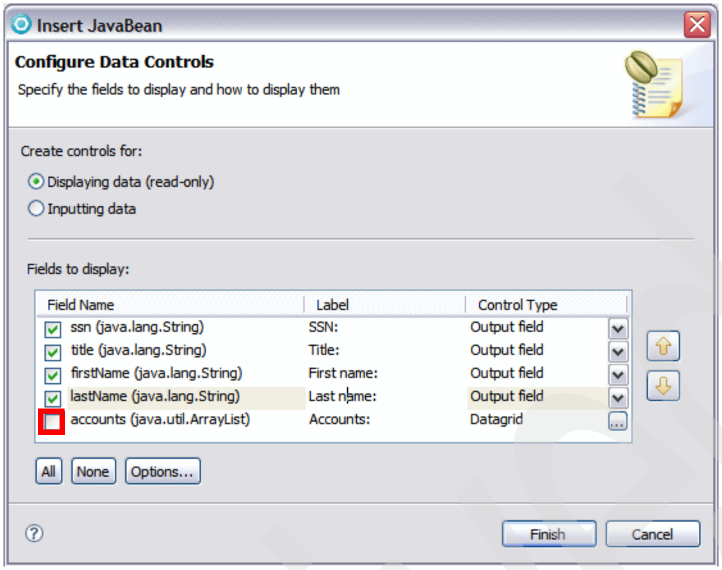
  > **Tip:** You can use the browse-button (marked with an ellipsis) to find the class using the class browser.

- Repeat this procedure to add the following request scope variable:
  - Variable name: `accounts`
  - Type: `itso.rad7.bank.model.Account[]`

  > **Important:** Note the square brackets—the variable *accounts* is an array of accounts.

- In the Palette view, select **Form Tags → Form** and click anywhere on the JSP page in the content table. A dashed box appears on the JSP page, representing the new form. Resize the box to take up most of the width of the content area.
- In the Properties view for the new **Form** element, enter the following items:
  - Action: `UpdateCustomer`
  - Method: Select **Post**.

  Note that you can use the Outline view to navigate to the form tag quickly.
- Add a table with customer information:
  - In the Page Data view, expand and select **Scripting Variables → requestScope → customer (`itso.rad7.java.model.Customer`)**.
  - Select and drag the customer object to the top half of the area inside the form that was previously created.
  - In the Object Type dialog, accept Object and click **OK**. We could set the type of the `accounts` field.
  - In the Insert JavaBean dialog (Figure 12-31), do these steps:
    - Select **Displaying data (read-only)**.
    - Use the arrow up and down buttons to arrange the fields in the order shown, and overtype the labels.
    - Clear the `accounts` field (we do not display the accounts).
    - Click **Finish**.



Insert JavaBean

Configure Data Controls

Specify the fields to display and how to display them

Create controls for:

● Displaying data (read-only)

○ Inputting data

Fields to display:

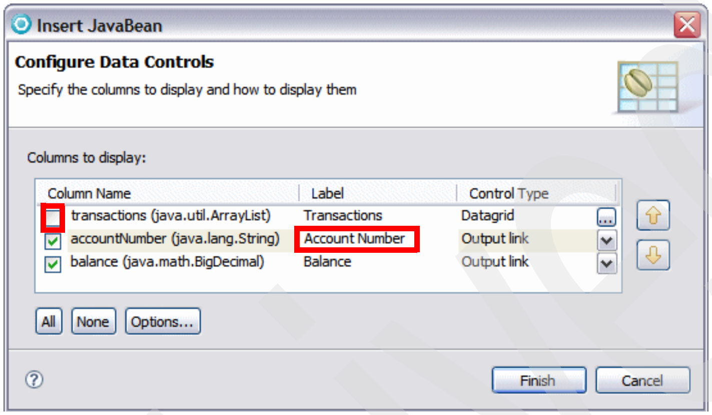
	Field Name	Label	Control Type
☑	ssn (java.lang.String)	SSN:	Output field
☑	title (java.lang.String)	Title:	Output field
☑	firstName (java.lang.String)	First name:	Output field
☑	lastName (java.lang.String)	Last name:	Output field
□	accounts (java.util.ArrayList)	Accounts:	Datagrid

Figure 12-31 Inserting the Customer JavaBean

> **Note:** The newly created table with customer data is changed in a later stage to use input fields for the title, first name, and last name fields. At the time of writing, this was not possible to achieve through the use of the available wizards.

- Right-click the last row of the newly created table (select the LastName cell) and select **Table** → **Add Row Below**.
- In the Palette view select **Form Tags** → **Submit Button** and click in the right-hand cell of the new row. Enter `Update` in the Label field and click **OK**. The Name field can be left empty.
- In the Palette view, select **HTML Tags** → **Horizontal Rule** and click in the area immediately below the form. Resize the line to be the same size as the form above it.
- In the Page Data view, expand and select **Scripting Variables** → **requestScope** → **accounts (itso.rad7.java.model.Account[])**.
- Select and drag the accounts object to area below the horizontal rule. Click **OK** in the Object Type dialog.

- In the Insert JavaBean wizard (Figure 12-32), do these steps:
  - Clear `transactions` (we do not display the transactions).
  - For both fields, select **Output link** in the Control Type column.
  - In the Label field for the `accountNumber` field, enter `Account Number`.
  - Ensure that the order of the fields is `accountNumber` and `balance`.
  - Click **Finish**, and the `accounts` Bean is added to the page and is displayed as a list.



Insert JavaBean

Configure Data Controls

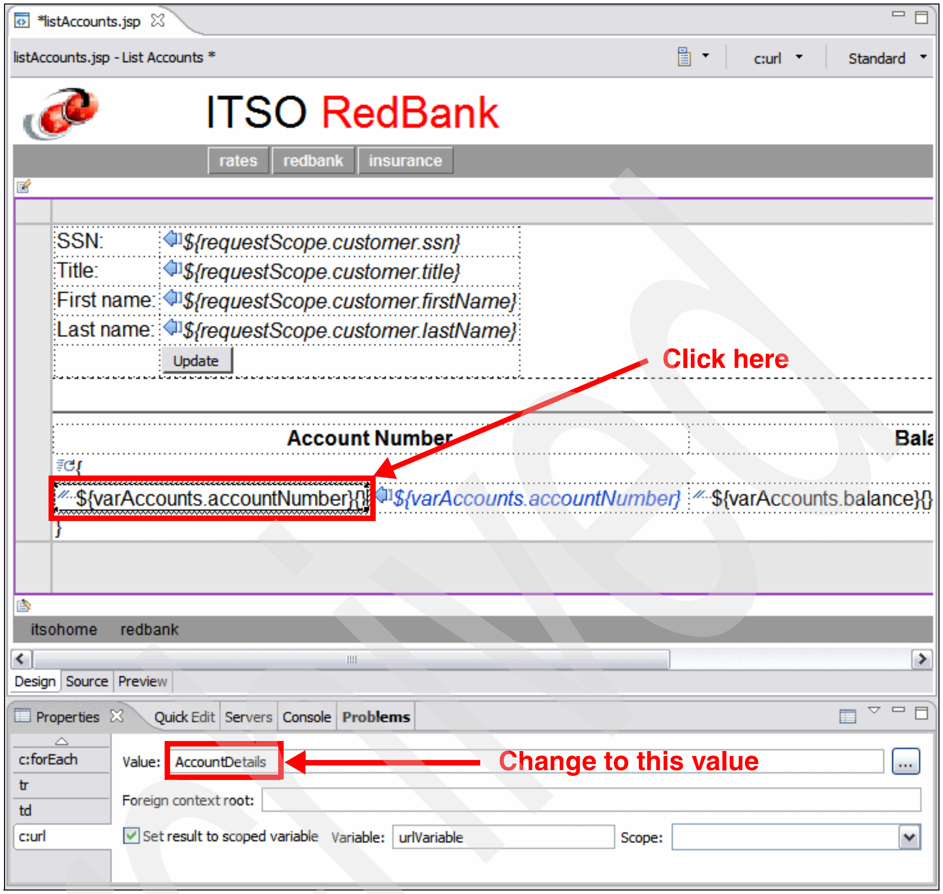
Specify the columns to display and how to display them

Columns to display:

	Column Name	Label	Control Type
☐	transactions (java.util.ArrayList)	Transactions	Datagrid
☑	accountNumber (java.lang.String)	Account Number	Output link
☑	balance (java.math.BigDecimal)	Balance	Output link

Figure 12-32 Inserting the accounts JavaBean

- The wizard inserts a JSTL `c:forEach` tag and an HTML table with headings, as entered in the Insert JavaBean window. Because we selected **Output link** as the Control Type for each column, corresponding `c:url` tags have been inserted. We now have to edit the URL for these links to make sure they are identical and to pass the `accountId` variable as a URI parameter.
  - Select the first c:url tag under the heading **Account Number**, which has the text **${varAccounts.accountNumber}{}**. In the Properties view, enter `AccountDetails` in the Value field (Figure 12-33).
  - The tag changes to `AccountDetails{}`.



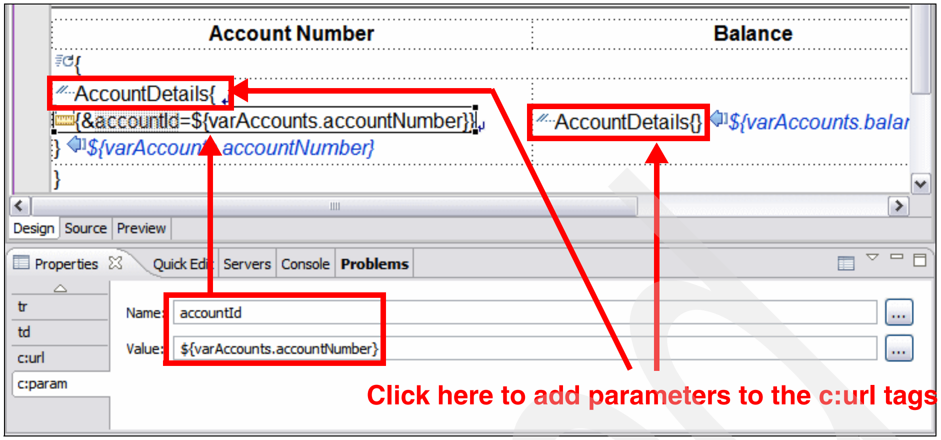
SSN: ${requestScope.customer.ssn}
Title: ${requestScope.customer.title}
First name: ${requestScope.customer.firstName}
Last name: ${requestScope.customer.lastName}

${varAccounts.accountNumber}{} ${varAccounts.accountNumber} ${varAccounts.balance}{}

Value: AccountDetails
Foreign context root:
☑ Set result to scoped variable Variable: urlVariable Scope:

Figure 12-33 Configuring the AccountDetails URL

- Select the second c:url tag: under the heading **Balance**, which has the text **${varAccounts.balance}{}**. In the Properties view, enter `AccountDetails` into the Value field. This specifies the target URL for the link, which in this case maps to the `AccountDetails` servlet.
- Now we must add a parameter to this URL, so ensure the link goes to the correct account. In the Palette view select **JSP Tags** → **Parameter** and and click on the first `c:url` in the **Account Number** column. This has the text **AccountDetails{} (**Figure 12-34).
- In the Properties view, for the **c:param** tab, enter `accountId` in the Name field and `${varAccounts.accountNumber}` in the Value field. This adds a parameter to the account url with a name of `accountId` and the value of the `accountNumber` request variable.



Name: accountId
Value: ${varAccounts.accountNumber}

Click here to add parameters to the c:url tags

Figure 12-34 Adding parameters to c:url tags

- – Repeat the two previous steps to add a parameter to the second `c:url` tag in the **Balance** column, showing the text **Account Number{}**. Note the field values are the same as used in step c, `accountId` in the Name field and `${varAccounts.accountNumber}` in the Value field.

- ► Click anywhere in the Balance column, and select the **td** tag in the Properties view. Select **Right** in the Horizontal alignment list box. This makes the contents of the Balance cells right-justified.

- ► Select the Source tab and compare the code to display the accounts to Example 12-1. This JSP code displays the accounts as a list, using the `c:forEach` tag to loop through each account, and the `c:out` tag to reference the current loop variable. The `c:url` tag builds a URL to `AccountDetails` and the `c:param` tag adds the `accountId` parameter (with account number value) to that URL.

Example 12-1 JSP code with JSTL tags to display accounts (formatted)

```
<c:forEach var="varAccounts" items="${requestScope.accounts}">
   <tr>
      <td>
         <c:url value="AccountDetails" var="urlVariable">
            <c:param name="accountId"
                     value="${varAccounts.accountNumber}"></c:param>
         </c:url>
         <a href="<c:out value='${urlVariable}' />">
            <c:out value="${varAccounts.accountNumber}"></c:out>
         </a>
      </td>
      <td align="right">
         <c:url value="AccountDetails" var="urlVariable">
```

```
            <c:param name="accountId"
                     value="${varAccounts.accountNumber}"></c:param>
         </c:url>
         <a href="<c:out value='${urlVariable}' />">
            <c:out value="${varAccounts.balance}" />
         </a>
      </td>
   </tr>
</c:forEach>
```

- In the Palette view, select **HTMLTags** → **Horizontal Rule** and click in the area below the account details table. You might have to increase the size of the content area.
- Add a logout form:
  - In the Palette view, select **Form Tags** → **Form** and click below the new horizontal rule. A dashed box will appear on the JSP page, representing the new form.
  - In the Properties view for the new form tag, enter the following items:
    - Action: `Logout`
    - Method: Select **Post**.
  - In the Palette view, select **Form Tags** → **Submit Button** and click in the right-hand cell of the new form. When the Logout button is clicked, the `doPost` method is called on the Logout servlet.
  - In the Insert Submit Button dialog, enter `Logout` in the Label field and click **OK**.
  - Click in the space above the form and button (this selects the <td> tag) and set the Horizontal alignment to **Center**.
- The remaining part is to change the title, first name and last name to be entry fields, so that the user can update the customer details.

  Do the following steps to convert the Title, First Name, and Last Name text fields to allow text entry:
  - Select the **${requestScope.customer.title}** field.
  - Select the **Source** tab and you can see the code:

    ```
    <td><c:out value="${requestScope.customer.title}" /></td>
    ```
  - Change the code to:

    ```
    <td><input type="text" name="title"
          value="<c:out value='${requestScope.customer.title}' />" /></td>
    ```
  - Repeat this for the first name and last name fields:

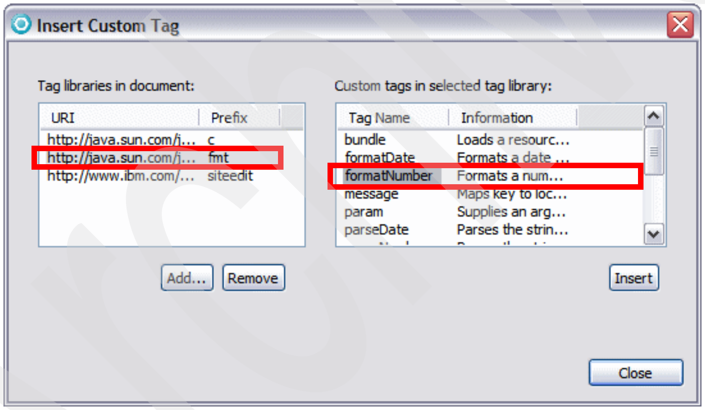
    ```
    <td><input type="text" name="firstName"
    ```

```
      value="<c:out value='${requestScope.customer.firstName}' />" /></td>
......
<td><input type="text" name="lastName"
      value="<c:out value='${requestScope.customer.lastName}' />" /></td>
```

This changes the customer fields from display only fields to be editable, so that the details can be changed.

You can increase the length of the fields.

- The width of the content areas can be change in the source code as well.
- The balance is a `BigDecimal` and must formatted, otherwise it displays with many digits:
  - Click on the balance field.
  - Select **JSP** → **Insert Custom**.
  - In the Insert Custom Tag dialog, click **Add** to add another tag library.
  - Locate and select the **http://java.sun.com/jsp/jstl/fmt** URI and click **OK**.
  - Select the new tag library and select **formatNumber** as the custom tag (Figure 12-35). Click **Insert** and **Close**.


Insert Custom Tag
Tag libraries in document:
Custom tags in selected tag library:
Add...
Remove
Insert
Close

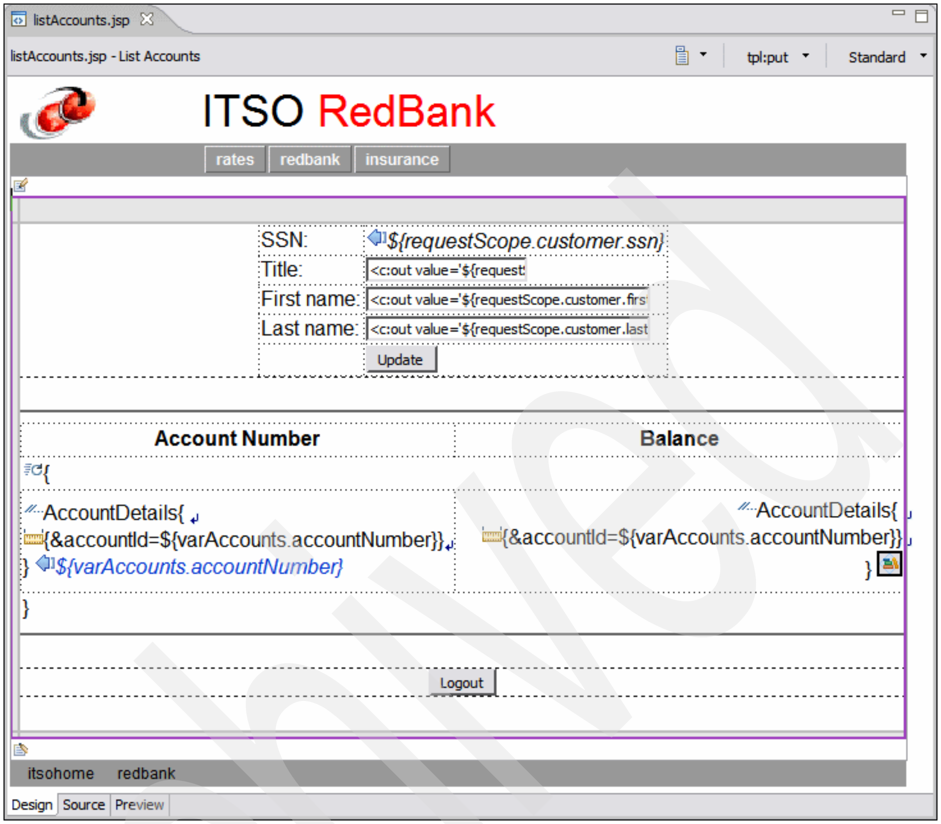
URI	Prefix
http://java.sun.com/j...	c
http://java.sun.com/j...	fmt
http://www.ibm.com/...	siteedit

Tag Name	Information
bundle	Loads a resourc...
formatDate	Formats a date ...
formatNumber	Formats a num...
message	Maps key to loc...
param	Supplies an arg...
parseDate	Parses the strin...

Figure 12-35 Inserting a custom tag

  - Select the **Source** tab.
  - Select the **`<fmt:formatNumber>`** tag and in the Properties view set `maxFractionDigits` and `minFractionDigits` to 2. For the `value` type `${varAccounts.balance}`.
  - Remove the `<c:cout value=.... >` and `</c:cout>` tags.
- Save the file.

The JSP is shown in Figure 12-36.



SSN:	${requestScope.customer.ssn}
Title:	<c:out value='${request
First name:	<c:out value='${requestScope.customer.firs
Last name:	<c:out value='${requestScope.customer.last
	Update

Account Number	Balance
{	

AccountDetails{
{&accountId=${varAccounts.accountNumber}}
} ${varAccounts.accountNumber}
	AccountDetails{
	{&accountId=${varAccounts.accountNumber}}
	}

}	

Logout

Figure 12-36 listAccounts.jsp finished

The jSP code is provided in `c:\7501code\webapps\jsp\listAccounts.jsp`. You can import the code into the WebContent folder, or copy/paste from Windows Explorer directly into Application Developer.

Implement the other JSPs

The other JPSs have already been created as part of the model solution. These were built by a similar process of adding request beans to the JSPs and building HTML and JSP elements around them.

To import the other JSP files, perform the following steps:

- Select the **WebContent** folder and **Import**, then select **General** → **File System.** Click **Next**.
- Click **Browse** and navigate to `c:\7501code\webapps\jsp`.

- Select all the JSP files except **listAccounts.jsp** (which has already been completed). Click **Finish**.
- When prompted whether to override the existing files, click **Yes to All**.

It is now possible to view the other JSP files in Page Designer. Unfortunately, these pages will not have the associated request beans showing in the Page Data view. These are maintained in the **.jspPersistence** file immediately under the Web project directory, but have to be specifically added to the PageData View for Application Developer to be aware of them.

For completeness, you can add the required variables to the appropriate Page Data view. Open the JSP file and in the Page Data view, select **Scripting Variables** → **New** → **Request Scope Variable** (Table 12-1).

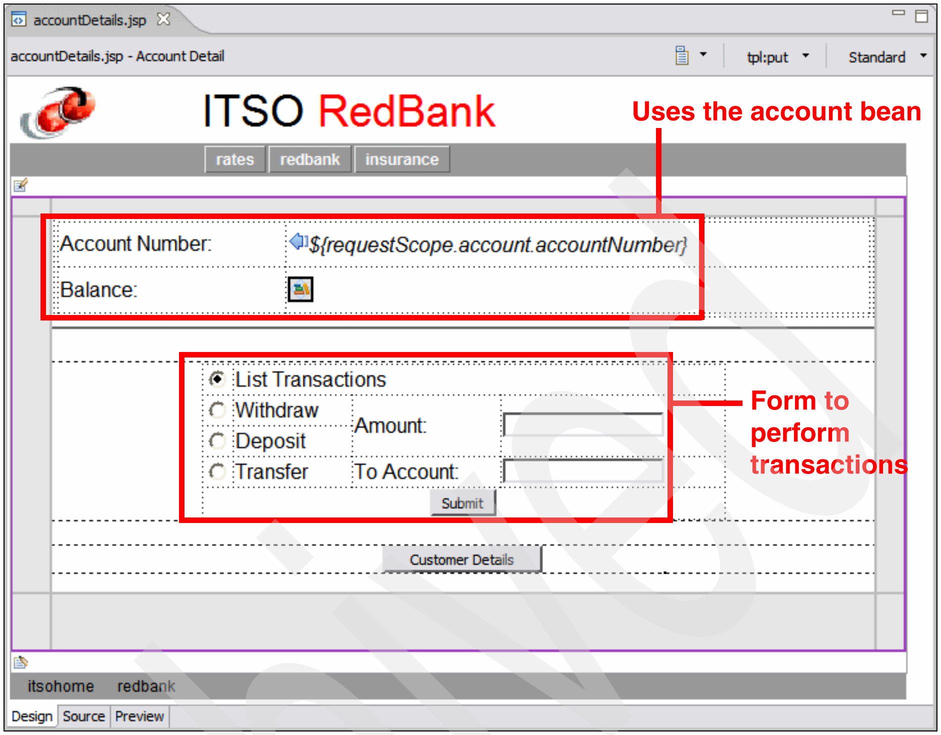
Table 12-1 Request scope variables for each JSP

JSP File	variable	type
`accountDetails.jsp`	`account`	`itso.rad7.bank.model.Account`
`listTransaction.jsp`	`transactions`	`itso.rad7.bank.model.Transaction[]`
`listTransaction.jsp`	`account`	`itso.rad7.bank.model.Account`
`showException.jsp`	`message`	`java.lang.String`
`showException.jsp`	`forward`	`java.lang.String`

Account Details JSP

The `acountDetails.jsp` shows the details for a particular customer account and gives options to execute a transaction:

- The JSP uses a single request variable called `account` to populate the top portion of the body of the page, which shows the account number and balance.
- The middle section is a simple static form, which provides fields for the details of a transaction (transaction type, amount, and destination account) and posts the request to the `PerformTransaction` servlet for processing.
- The Customer Details button navigates the user to the `listAccounts.jsp` page.
- The Page Designer view of this page is shown Figure 12-37.



Account Number: ${requestScope.account.accountNumber}

Balance:

● List Transactions

○ Withdraw

○ Deposit

○ Transfer

Amount:

To Account:

Figure 12-37 Completed accountDetails.jsp in design view

List Transactions JSP

The `listTransactions.jsp` shows a read only view of the account including the account number and balance plus a list of all transactions:

- The JSP uses two request variables called `account` and `transactions`. The first section of the page uses the `account` request bean to populate a table showing the account number and balance.
- The middle section uses the `transactions[]` request bean to show a list of transactions. This uses the JSTL tag library to iterate through the transaction list and build up an HTML representation of the transaction history.
- The Account Details button returns the browser to the `accountDetails.jsp` page.
- The Page Designer view of this page is shown Figure 12-38.

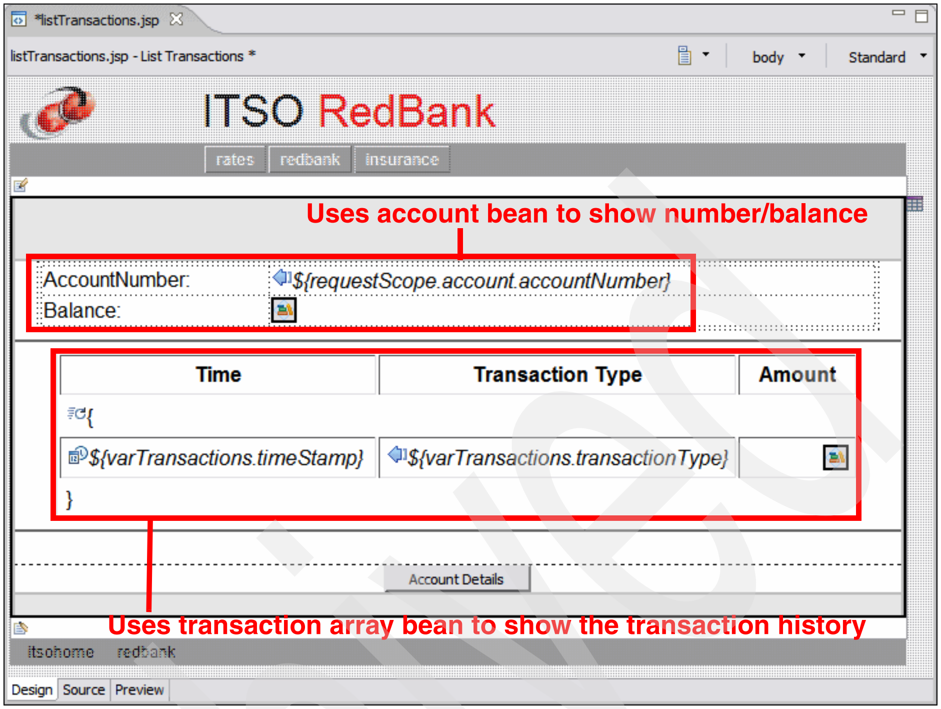
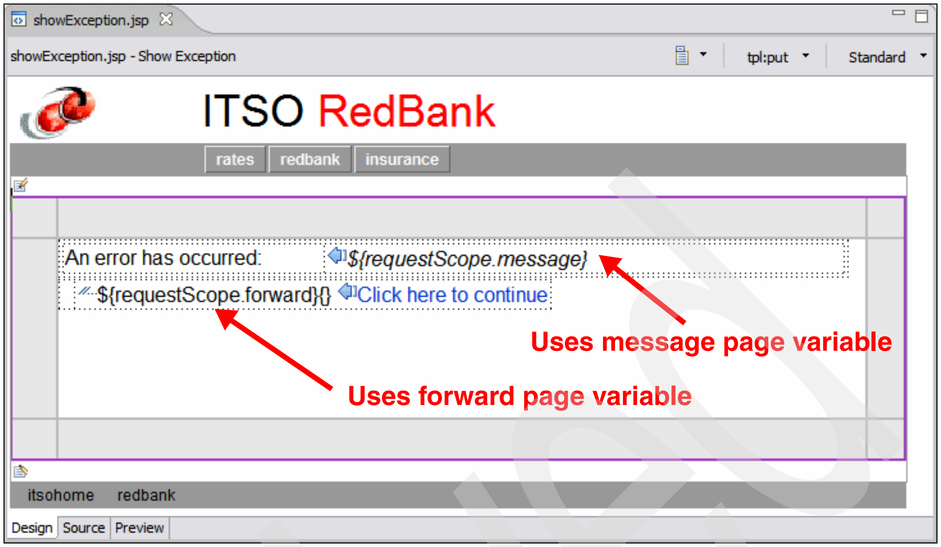

*listTransactions.jsp
listTransactions.jsp - List Transactions *
body
Standard
ITSO RedBank
rates
redbank
insurance
Uses account bean to show number/balance
AccountNumber:
${requestScope.account.accountNumber}
Balance:
Time
Transaction Type
Amount
{
${varTransactions.timeStamp}
${varTransactions.transactionType}
}
Account Details
Uses transaction array bean to show the transaction history
itsohome
redbank
Design
Source
Preview

Figure 12-38 Completed listTransactions.jsp in design view

Show Exception JSP

The `showException.jsp` is displayed when an exception occurs in the processing of a request:

- The JSP shows a simple error message and gives a link to another page within the RedBank application to allow the user to continue.
- There are two request beans used on this page; `message` stores the text to display to the user, and `forward` stores a URL for the next page to continue. The URL is hidden behind the text Click here to continue.
- Figure 12-39 shows this page in the design view of Page Designer.



An error has occurred: ${requestScope.message}

${requestScope.forward}{} Click here to continue

Uses message page variable

Uses forward page variable

Figure 12-39 Completed showException.jsp in the design view

The RedBank application is finished and ready to be tested.

Testing the Web application

This section demonstrates how to run the sample RedBank application, built in the previous sections.

Prerequisites to run the sample Web application

To run the RedBank application, you must do one of the following actions:

- Complete the sample following the procedures described in “Implementing the RedBank application” on page 486.
- Import the completed project interchange file from:

  ```
  c:\7501code\zInterchangeFiles\webapps\RAD7BankBasicWeb.zip
  ```

  Refer to “Importing sample code from a project interchange file” on page 1310 for details.

Running the sample Web application

To run the RedBank Web application in the test environment, do these steps:

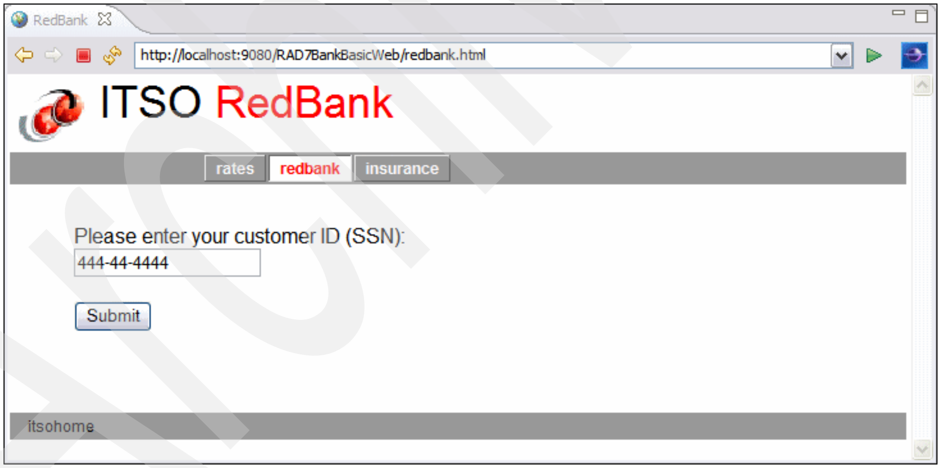
- Right-click **RAD7BankBasicWeb** in the Project Explorer and select **Run As** → **Run on Server**.
- In the Server Selection dialog, select **Choose an existing server**, select **WebSphere Application Server v6.1**, and click **Finish**.

The main page of the Web application should be displayed in a Web browser inside Application Developer.

Verifying the RedBank Web application

Once you have launched the application by running it on the test server, there are some basic steps that can be taken to verify the Web application is working properly.

- From the main page, select the **redbank** menu option.
- In the RedBank page type a customer social security number, for example, 444-44-4444 (Figure 12-40).

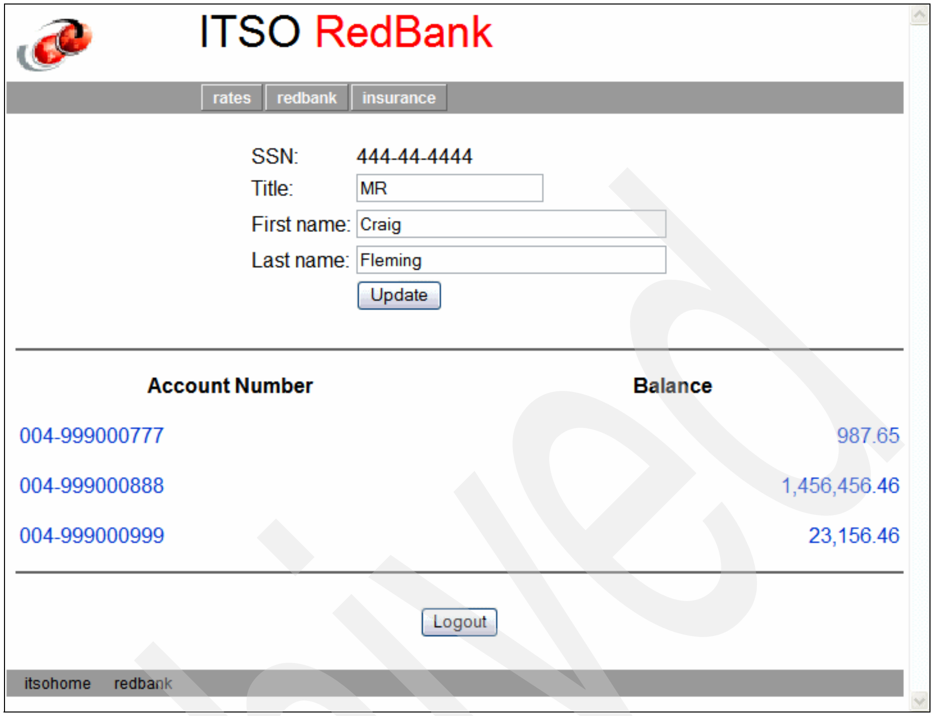


Please enter your customer ID (SSN):

444-44-4444

Figure 12-40 ITSO RedBank Login page

- Click **Submit**, and the customer and the accounts are listed (Figure 12-41).



SSN: 444-44-4444

Title: MR

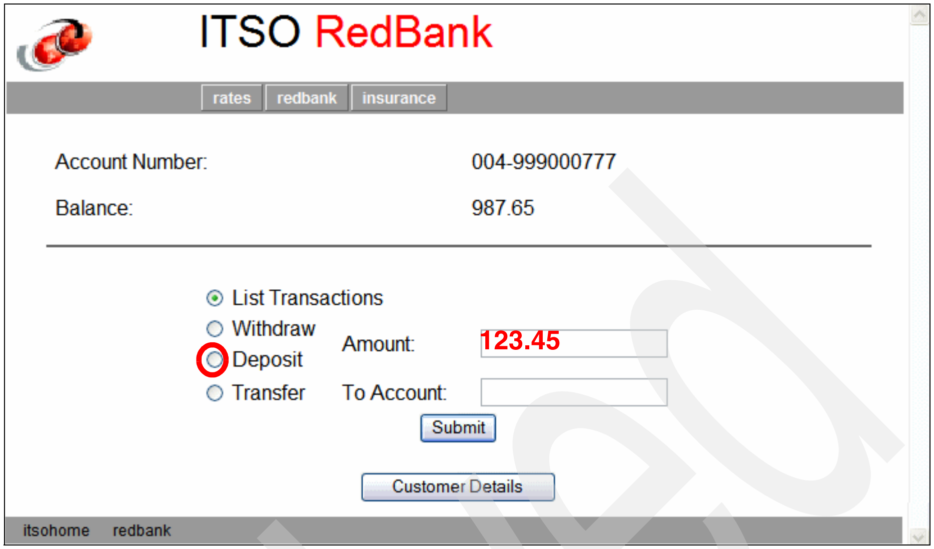
First name: Craig

Last name: Fleming

Account Number	Balance
004-999000777	987.65
004-999000888	1,456,456.46
004-999000999	23,156.46

Figure 12-41 Display of customer accounts

- These actions are supported:
  - Change the customer title or name fields and click **Update**. This performs the `doPost` method of the `UpdateCustomer` servlet. For example, change the title to Sir and then click **Update**.
  - Clicking **Logout** performs a logout and returns to the Login page.
  - Clicking on any account displays the account information (Figure 12-42).



Account Number: 004-999000777

Balance: 987.65

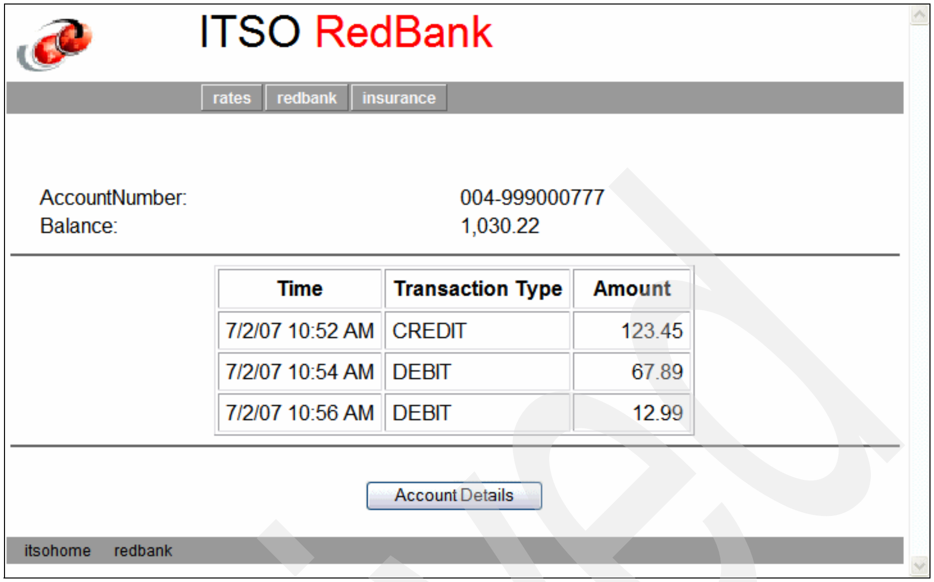
● List Transactions
○ Withdraw
[✓ ○ Deposit]
○ Transfer

Amount: 123.45

To Account:

Figure 12-42 Details for a selected account

- These actions are supported:
  - Select **List Transactions** and click **Submit**. There are no transactions yet.
  - Select **Deposit** or **Withdraw**, enter an amount and click **Submit** to execute a banking transaction. The page is redisplayed with the balance updated.
  - Select **Transfer**, enter an amount and a target account, and click **Submit** The page is redisplayed with the balance updated.
  - Click **Customer Details** to return to the account listing.
- Run a few transactions (deposit, withdraw, transfer), then select **List Transactions** and click **Submit**. The transaction listing is displayed (Figure 12-43).

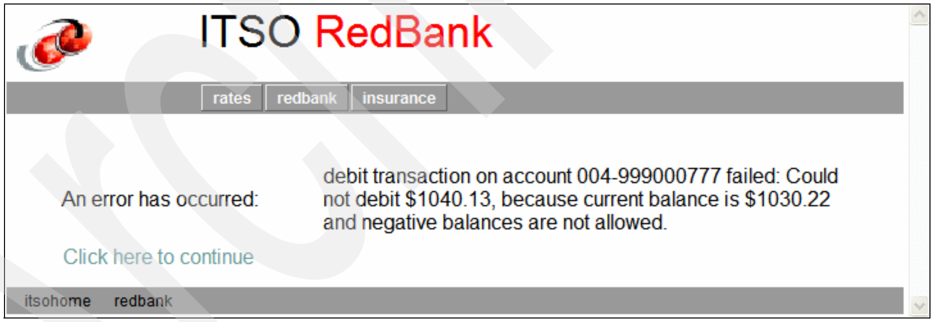


AccountNumber: 004-999000777
Balance: 1,030.22

Time	Transaction Type	Amount
7/2/07 10:52 AM	CREDIT	123.45
7/2/07 10:54 AM	DEBIT	67.89
7/2/07 10:56 AM	DEBIT	12.99

Figure 12-43 List of transactions for an account

- Try a withdraw of an amount greater than the balance. The Show Exception JSP is displayed with an error message (Figure 12-44).



An error has occurred: debit transaction on account 004-999000777 failed: Could not debit $1040.13, because current balance is $1030.22 and negative balances are not allowed.

Click here to continue

Figure 12-44 Withdraw over the limit error

RedBank Web application conclusion

We hope that all of the foregoing demonstrations worked successfully for you. The example has demonstrated the following features, which are just a subset of the tools available within Application Developer for Web development:

- Basic servlets
- Basic JSPs
- Page Designer
- Page templates
- Web Site Navigation editor
- CSS editor
- JSP tags
- New Servlet wizard
- New Web Project wizard
- Web Library projects
- Navigation tabs
- JSTL tag library

More information

There are many ways that the RedBank application can be improved, by adding features or using other technologies. Some of these are covered in subsequent chapters of this book, including these:

- Using EJBs to store the model—Chapter 16, “Develop Web applications using EJBs” on page 719
- Using a database rather than HashMaps—Chapter 9, “Develop database applications” on page 355
- Using Struts to handle the requests—Chapter 13, “Develop Web applications using Struts” on page 541
- Using JSF components rather than JSTL—Chapter 14, “Develop Web applications using JSF and SDO” on page 587
- Debugging the application—Chapter 22, “Debug local and remote applications” on page 1041

The Help feature provided with Application Developer has a large section on Developing Web sites and applications. It contains reference information for all the features presented in this chapter and further information on topics only covered briefly here, including JSP tag libraries, Security, and use of the Web Diagram Editor.

Finally, the following URLs provide further information for the topics covered in this chapter:

- **Sun Java Servlet Technology Home page**—Contains links to the specification, API Javadoc, and articles on servlets.

  http://java.sun.com/products/servlet/index.jsp

- **Sun JavaServer Pages Technology Home page**—Home page for technical information on JSPs:

  http://java.sun.com/products/jsp/jstl/

- **Sun JavaServer Pages Standard Tag Library (JSTL)**—Home page for technical information on JSTL:

  http://java.sun.com/products/jsp/jstl/

- **Online Color Scheme**—Useful for figuring out the hex code for a particular color:

  http://www.colorschemer.com/online.html

- **JSP and Servlets best practices**—This is an old article, but it articulates clearly the different ways of applying an MVC pattern to JSPs and servlets:

  http://java.sun.com/developer/technicalArticles/javaserverpages/servlets_jsp/

- **Experience J2EE! Using WebSphere Application Server V6.1, SG24-7297**

  This book has an excellent section on implementing J2EE security in WebSphere Application Server:

  http://www.redbooks.ibm.com/abstracts/sg247297.html

13

Develop Web applications using Struts

Apache Struts is an instance of a servlet/JSP model-view-controller (MVC) framework. The goal of Apache Struts is to provide an open source framework useful in building Web applications with Java servlet and JavaServer Pages (JSP) technology. In addition, Struts encourages application architectures based on the MVC design paradigm.

In this chapter, we explore the tooling support found in Rational Application Developer Version 7.0 for Apache Struts.

The chapter is organized into the following sections:

- Introduction to Struts
- Preparing for the sample application
- Developing a Web application using Struts
- Running the Struts Bank Web application
- Developing a Struts Web application using Tiles

Introduction to Struts

The Struts framework control layer uses technologies such as servlets, JavaBeans, and XML. The view layer is implemented using JSPs and tag libraries. The Struts architecture encourages the implementation of the concepts of the model-view-controller (MVC) architecture pattern. By using Struts, you can get a clean separation between the presentation (view) and business logic (model) layers of your application.

Struts also speeds up Web application development by providing an extensive JSP tag library, parsing and validation of user input, error handling, and internationalization support.

The focus of this chapter is on the Application Developer tooling used to develop Struts-based Web applications. Although we do introduce some basic concepts of the Struts framework, we recommend that you refer to the following sites for further in-depth information:

- Apache Struts home page:

  `http://struts.apache.org/`

- Apache Struts User Guide:

  `http://struts.apache.org/userGuide/introduction.html`

> **Note:** Since the prior version of Application Developer (Version 6.x), Struts has forked into three distinct frameworks:
>
> - Struts Classic (which is the original Struts framework)
> - Struts 2 (which is Struts + Webwork)
> - Struts Shale (which is a JSF version of Struts, now known as *Shale*
>
> Application Developer V7.0 includes support for only Struts Classic. The versions supported by Application Developer are Struts Version 1.0.2 and 1.1. At the time of writing this book, the latest version of the Struts Classic framework was V1.3.8.

Model-view-controller (MVC) pattern with Struts

In "Model-view-controller (MVC) pattern" on page 471, we described the general concepts and architecture of the MVC pattern. Figure 13-1 shows the Struts components in relation to the MVC pattern:

- **Model**: Struts does not provide model classes. Specifically, Struts does not provide the separation between the controller and model layers. The separation must be provided by the Web application developer as a facade, service locator, EJB, or Java Bean.
- **View**: Struts provides action forms (or form beans) in which data is automatically or manually collected from HTTP requests with the purpose to pass data between the view and controller layers. In addition, Struts provides custom JSP tag libraries that assist developers in creating interactive form-based applications using JSPs. Application resource files hold text constants and error message, translated for each language, that are used in JSPs.
- **Controller**: Struts provides an `ActionServlet` (controller servlet) that populates action forms from JSP input fields and then delegates work to an action class where the application developer implements the logic to interface with the model.

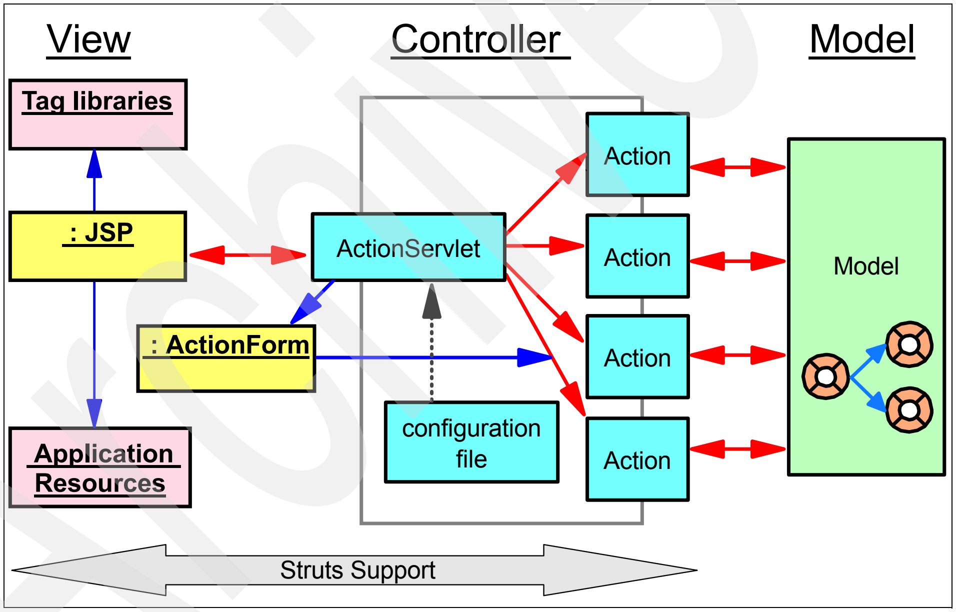


Figure 13-1 Struts components in the MVC architecture

A typical Struts Web application is composed of the following components:

- **Action servlet**—A single servlet (extending `org.apache.struts.action.`**`ActionServlet`**) implements the primary function of mapping a request `URI` to an action class. Before calling the action class, it populates the action form associated to the action with the fields from the input JSP. If specified, the action servlet also requests the action form to validate the data. It then calls the action class to carry out the requested function. If action form validation fails, control is returned to the input JSP so the user can correct the data. The action servlet is configured in the Web deployment descriptor (`web.xml`). The action servlet controls and manages the relationship between other Struts components which is configured in the Struts configuration file (`struts-config.xml`).

- **JSPs**—Multiple JSPs that provide the end-user view. Struts includes an extensive tag library to make JSP coding easier. The JSPs display the information prepared by the action classes and requests new information from the user.
- **Action classes**—Multiple action classes (extending any one of the Struts action classes like `org.apache.struts.action.`**`Action`**) that interface with the model. When an action has performed its processing, it returns an action forward object, which determines the view that should be called to display the response (or alternatively forward to another action class). The action class prepares the information required to display the response, usually as an action form (although, it is not recommended), and makes it available to the JSP. Usually the same action form that was used to pass information to the action is used also for the response, but it is also common to have special view beans tailored for displaying the data. An action forward has properties for its name (logical mapping), path(URI), and a flag specifying if a forward or a send redirect call should be made. The address to an action forward is usually externalized in the Struts configuration file, but can also be generated dynamically by the action class.
- **Action forms**—Multiple action forms (extending one of the Struts action form classes like `org.apache.struts.action.ActionForm`) to help facilitate transfer form data from JSPs. The action forms are generic JavaBeans with getters and setters for the input fields available on the JSPs. Usually there is one form bean per Web page, but you can also use more coarse-grained form beans holding the properties available on multiple Web pages (this fits very well for wizard-style Web pages). If data validation is requested (a configurable option) the form bean is not passed to the action until it has successfully validated the data. Therefore the form beans can act as a sort of firewall between the JSPs and the actions, only letting valid data into the system.
- **Resource files**—One application resource file per language supported by the application holds text constants and error messages and makes internationalization easy.

Figure 13-2 shows the basic flow of information for an interaction in a Struts Web application.

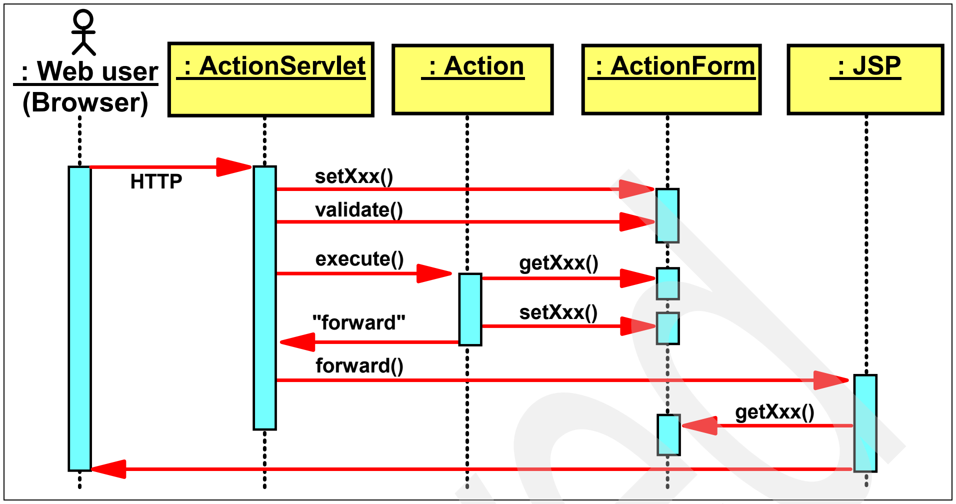


Figure 13-2 Struts request sequence

- A request from a Web browser is first received by the Struts action servlet.
- If the action that handles the request has a form bean associated with it, Struts creates the action form and (and if specified, automatically) populates it with the data from the input form.
- It then calls the validate method of the action form. If validation fails, the user is returned to the input page to correct the input. If validation succeeds, Struts calls the action's execute method.
- The action retrieves the data from the form bean and performs the appropriate logic. The action often call session EJBs to perform the business logic.
- When done, the action either creates a new action form (or other appropriate view bean) or reuses the existing one, populates it with new data, and stores it in the request (or session) scope.
- The action then returns a forward object to the action servlet, which forwards to the appropriate output JSP (or alternatively forwards to another action).
- The JSP uses the data in the action form to render the result.

Application Developer support for Struts

Application Developer provides the following support for Struts-based Web applications:

- A Web project can be configured for Struts. This adds the Struts runtime (and dependent JARs), tag libraries, and the action servlet to the project, and creates skeleton Struts configuration and application resources files. Application Developer provides support for Struts 1.1, selectable when setting up the project. This field is selectable, because at the time of this writing, support for Struts 1.2.x is being added to Application Developer.

- A set of *Struts Component Wizards* allows you to define `action form` classes, `action` classes with action forwarding information, and JSP skeletons with the tag libraries included.
- The *Struts Configuration Editor* is provided to maintain the control information for the action servlet.
- The *Web Diagram Editor* provides a graphical design tool to edit a graphical view of the Web application from which components (action forms, actions, JSPs) can be created using the wizards. The Web Diagram Editor provides top-down development (developing a Struts application from scratch), bottom-up development (that is, you can easily diagram an existing Struts application that you may have imported), and meet-in-the-middle development (that is, enhancing or modifying an existing diagrammed Struts application). The Web Diagram Editor is now written on top of the Graphical Modeling Framework (GMF). Improved support allows direct/in-sync manipulation of actual components/artifacts to reflect changes in the Web Diagram Editor.

Note: The Web Diagram in Application Developer, which supports the creation of various Struts components, can also be applied in the creation of Struts components when using the IBM Struts Portlet framework.

Specifically, when working with a Portlet project and the project has been enabled to use the IBM Struts Portlet framework, the same palette options of the Web Diagram are available to the Portlet project.

- The *Project Explorer* view provides a hierarchical (tree-like) view of the application. This view shows the Struts artifacts (such as actions, action forms, global forwards, global exceptions, and Web pages). You can expand the artifacts to see their attributes. For example, an action can be expanded to see the Action Forms, and forwards and local exceptions associated with the selected Action. This is useful for understanding specific execution paths of your application. The Project Explorer view is available in the Web perspective.
- The *JSP Page Designer* provides support for rendering the Struts tags, making it possible to properly view Web pages that use the Struts JSP tags. This support is customizable using Application Developer's Preferences settings.

- *Validators* to validate the Struts XML configuration file and the JSP tags used in the JSP pages.

Preparing for the sample application

This section describes the tasks that have to be completed prior to developing the Web application using Struts.

> **Note:** A completed version of the ITSO RedBank Web application built using Struts can be found in the project interchange file:
>
> `C:\7501code\zInterchangeFiles\struts\RAD7Struts.zip`
>
> If you do not want to develop the sample yourself, but want to see it run, follow the procedures described in “Importing the final sample application” on page 585.

Setting up the sample database

The `ITSOBANK` database can be set up using either DB2 or Derby.

The instructions for creating the `ITSOBANK` database are given in “Setting up the ITSOBANK database” on page 1312.

Configuring the data source in the WebSphere Server 6.1

The instructions for configuring the data source are given in “Configuring the data source in WebSphere Application Server” on page 1313.

Activating Struts development capabilities

To activate the Struts development capabilities, such as the Struts Configuration Editor, we must update the Application Developer preferences:

- Select **Window** → **Preferences** → **General** → **Capabilities**.
- Select **Web Developer (advanced)** and click **Advanced**.
- In the Advanced dialog, expand Web Developer (advanced) and select **Struts Development**.
- Click **OK** twice to close the dialogs.

ITSO Bank Struts Web application overview

We use the ITSO Bank as the theme of our sample Web application using Struts. Similar samples were developed in the following chapters using other Web application technologies:

- Chapter 12, “Develop Web applications using JSPs and servlets” on page 465
- Chapter 14, “Develop Web applications using JSF and SDO” on page 587

The ITSO Bank sample application allows a customer to enter a customer ID (social security number), select an account to view detailed transaction information, or perform a deposit or withdrawal on the account. The model layer of the application is implemented within the action classes using Java beans for the sake of simplicity.

In the banking sample, we use the Struts framework for the controller and view components of the Web application. We implement the model using JavaBeans and the Derby database (or DB2).

Figure 13-3 displays the Struts Web Diagram for the sample banking application. The basic description and flow of the banking sample application are as follows:

- The `logon.jsp` page is displayed as the initial page of the banking sample application. The customer is allowed to enter her or his social security number. In our case we use simple validation of the Struts framework to check for an empty value. If the customer does not enter a valid value, the Struts framework returns to the `logon.jsp` page and display the appropriate message to the user.
- The `logon` action logs in the user, and on successful logon retrieves the customer and the account information and lists all the accounts associated with the customer using the `customerlisting.jsp` Web page.
- In the `customerlisting.jsp`, the customer can select to see details of an account using the `accountDetails` action, or perform a transaction on an account using the `transact.jsp`.
- In the `accountDetails.jsp`, the details of the account are displayed, including the transactions that have been performed on the account. The customer can select to perform another transaction using the `transact.jsp`.
- The `transact.jsp` invokes the `performTransaction` action. After a successful transaction the `accountDetails` action is invoked to redisplay the `accountDetails.jsp`.
- The customer can log off using the logoff link, which invokes the `logoff` action.

- In case of errors, an `error.jsp` is displayed.

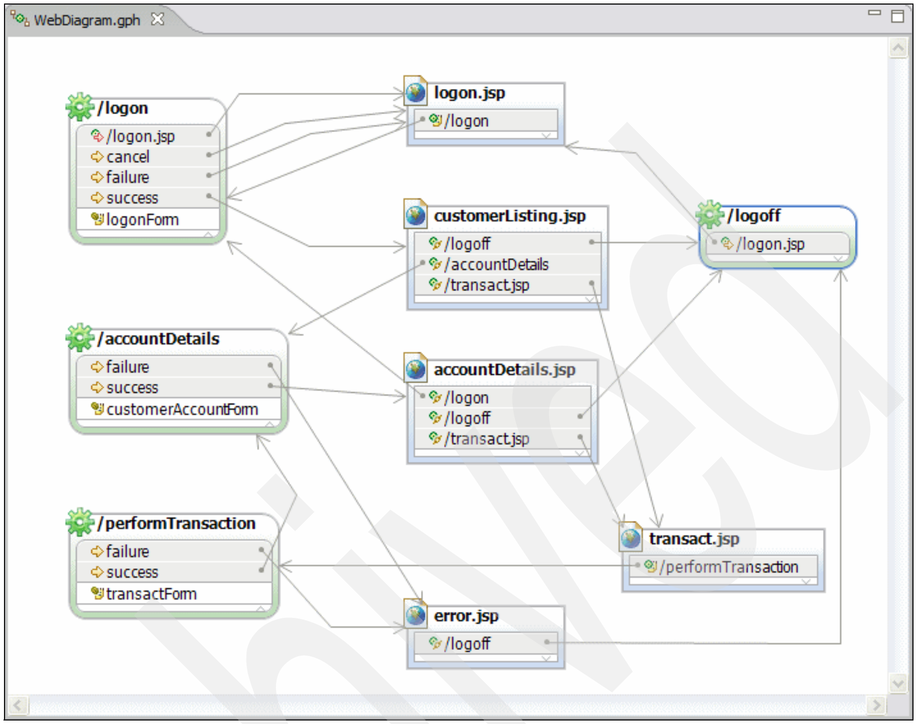


Figure 13-3 Struts Web Diagram: ITSO Bank sample

In this section we focus on creating the various Struts components, including the Struts controller, actions, action forms, and Web pages, and relate these components together. We implement the following steps to demonstrate the capabilities of Application Developer:

- Create a dynamic Web application with Struts support: In this section the process of creating a dynamic Web application with Struts support and the wizard generated support for Struts is be described.
- Create Struts components: In this section we focus on creating Web pages, actions, action forms, exceptions (local and global), and forwards using the Web Diagram, and modify the properties of Struts components using the Struts Configuration Editor.
- Struts Configuration Editor: In this section we focus on creating Struts components using the Struts Configuration Editor that provides a visual editor to modify the Struts configuration file `struts-config.xml`.

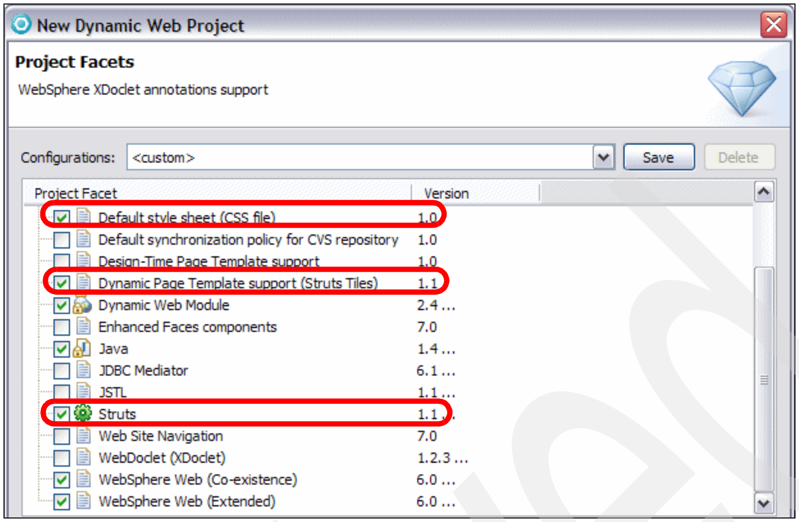
- Import the complete sample banking Web application: In the previous sections we created various Struts components. Here we import the complete banking sample implemented as shown in Figure 13-3 on page 549.
- Run the sample banking application: In this section we verify the data source configurations in the extended application descriptor of the imported sample and then run and test the application in the WebSphere V6.1 Test Environment.

> **Note:** Because this chapter focuses on Application Developer's Struts tools and wizards (more than on the architecture and best practices of a Struts application), we try to use the Struts tools and wizards as much as possible when creating our application.
>
> After having used the wizards to create some components (JSPs, form beans, actions), you might find it faster to create new components by copying and pasting from your existing components than by using all the wizards.

Creating a dynamic Web project with Struts support

To create a dynamic Web project with Struts support, do these steps:

- Open the Web perspective.
- Select **File** → **New** → **Dynamic Web Project** and click **Next**.
- Type **`RAD7StrutsWeb`** in the Project Name field, `RAD7StrutsEAR` for the EAR Project Name field and click **Next**.
- In the Project Facets dialog, select **Struts** and **Struts Tiles** (both with Version 1.1), and click **Next (**Figure 13-4).



New Dynamic Web Project

Project Facets

WebSphere XDoclet annotations support

Configurations: <custom>

Project Facet	Version
☑ Default style sheet (CSS file)	1.0
☐ Default synchronization policy for CVS repository	1.0
☐ Design-Time Page Template support	1.0
☑ Dynamic Page Template support (Struts Tiles)	1.1
☑ Dynamic Web Module	2.4 ...
☐ Enhanced Faces components	7.0
☑ Java	1.4 ...
☐ JDBC Mediator	6.1 ...
☐ JSTL	1.1 ...
☑ Struts	1.1 ...
☐ Web Site Navigation	7.0
☐ WebDoclet (XDoclet)	1.2.3 ...
☑ WebSphere Web (Co-existence)	6.0 ...
☑ WebSphere Web (Extended)	6.0 ...

Figure 13-4 Create a Struts-based dynamic Web project: Facets

- Accept the defaults in the Web Module dialog for Context Root, Content Directory and Java Source Directory, and click **Next**.
- Accept the defaults in the Struts Settings dialog for Resource Bundle and click **Finish**.

> **Note:** At the time of writing, Struts V1.0.2 and 1.1 are the only versions supported and available from the Struts list box.

At this point, a new dynamic Web project with Struts support (`RAD7StrutsWeb`) and an enterprise application project (`RAD7StrutsEAR`) have been created, and the Web Diagram Editor is open.

The following Struts-specific artifacts are created and Web application configurations are modified by the wizard related to Struts when a new dynamic Web application is created with Struts support (Figure 13-5):

- Struts configuration file `struts-config.xml` in `WebContent/WEB-INF`.
- Web deployment descriptor `web.xml` with two servlets:
  - The Struts `ActionServlet` and a servlet mapping for the action servlet to handle all client requests matching the regular expression pattern `*.do`, as shown in Figure 13-5.
  - The `TilesServet` for Struts Tiles support, without a servlet mapping.

Note: The Struts `ActionServlet` is configured (in `web.xml`) to intercept all requests with a URL ending in .do (the servlet mapping is *.do). This is common for Struts applications, but equally common is using a servlet mapping of `/action/*` to intercept all URLs beginning with `/action`.

To see the servlet mapping, open the Web deployment descriptor.

- `ApplicationResources.properties` in the package specified in the Struts Settings dialog in the creation of a dynamic project. This is a property file used in Struts to externalize messages and form field labels to accommodate for a language and locale independent fashion (or internationalization - i18n).
- Default Struts module with the name `<default Module>` under which all the Struts components are created.
- Default Web diagram `WebDiagram.gph` used by the Struts Web Diagram Editor is used to create Struts components in a top-down design approach.

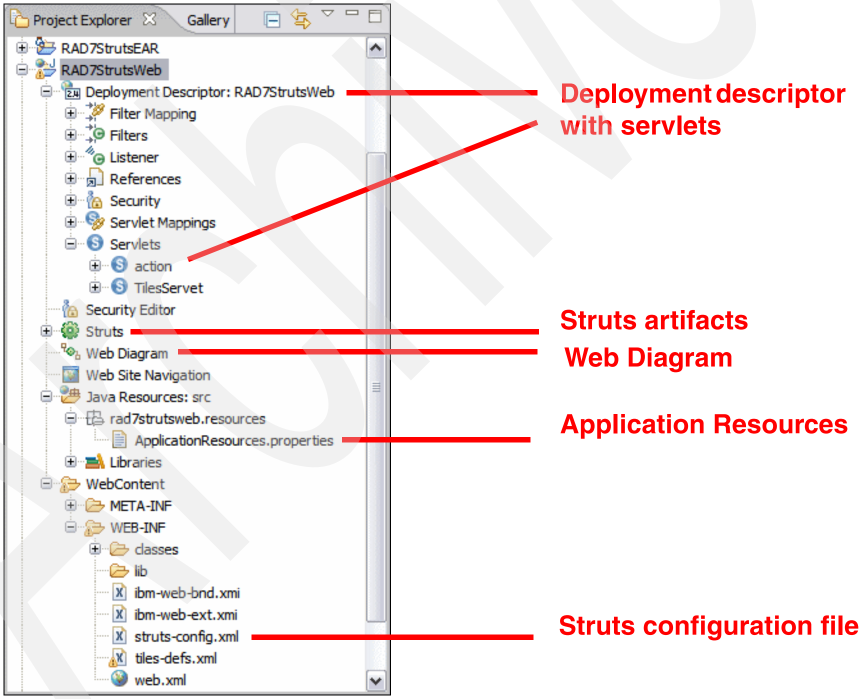


Figure 13-5 Web project with Struts support

Developing a Web application using Struts

This section describes how to develop a Web application using Struts with the tooling provided by Application Developer.

The section is organized into the following tasks:

- Creating the Struts components
- Modify application resources
- Using the Struts validation framework
- Page Designer and the Struts tag library
- Using the Struts Configuration Editor

> **Important:** This section demonstrates how to develop a Web application using Struts with the tooling included with Application Developer. We do not cover the details for all of the sample code. A procedure to import and run the completed Struts Bank Web application sample can be found in "Developing a Struts Web application using Tiles" on page 580.

Creating the Struts components

There are several ways to create Struts components:

- In the Project Explorer, expand and right-click **RAD7StrutsWeb** → **Struts** → **<default module>** and select **New** to create a Struts **Module**, **Action Mapping**, **Form Bean**, **Global Forward**, and **Global Exception**.
- Using the Struts Configuration Editor: Application Developer provides a Struts Configuration Editor, which is used to create Struts components and to modify the Struts configuration file `struts-config.xml`. We describe how to use the Struts Configuration Editor in detail in "Using the Struts Configuration Editor" on page 569.
- In the Struts Web Diagram Editor, use the Struts Palette to create Struts components.

In this chapter, we take a top-down approach to design the Web application by laying out all the components in the Web diagram using the Web Diagram Editor.

Note that we do not build the entire application using the Web Diagram. We just show examples of building Struts components.

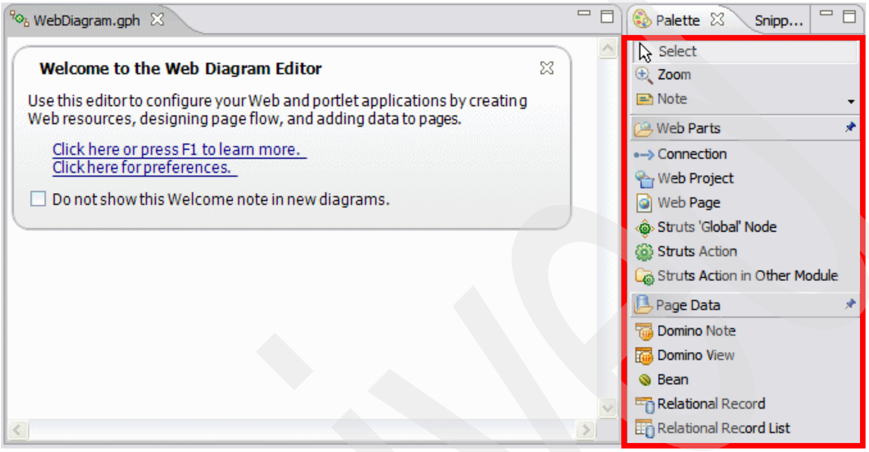
This section is organized into the following tasks:

- Start the Web Diagram Editor
- Create a Struts action
- Create a Struts form bean

- Create a Web page
- Create a Struts Web connection

Start the Web Diagram Editor

Open the Web Diagram under the `RAD7StrutsWeb` project (Figure 13-6).



Welcome to the Web Diagram Editor

Use this editor to configure your Web and portlet applications by creating Web resources, designing page flow, and adding data to pages.

Click here or press F1 to learn more.
Click here for preferences.

□ Do not show this Welcome note in new diagrams.

Figure 13-6 Web Diagram with the Struts drawer (Web Parts) open in the Palette

Create a Struts action

To create the Struts Action for logon, do these steps:

- In the Palette expand **Web Parts**.
- Drag and drop the Struts **Action** icon (Struts Action) from the Palette.
- Overtype the name of the action with `/logon` (Figure 13-7).

  When initialially typing in the name, the logon component appears in black and white. This is because the component has not yet been realized. Once you are done with the typing in the name, the Struts action becomes automatically realized. The Web Diagram Editor directly manipulates the underlying artifacts (creation of action class and updating of the `struts-config.xml`). As a result, the action widget is displayed in color.

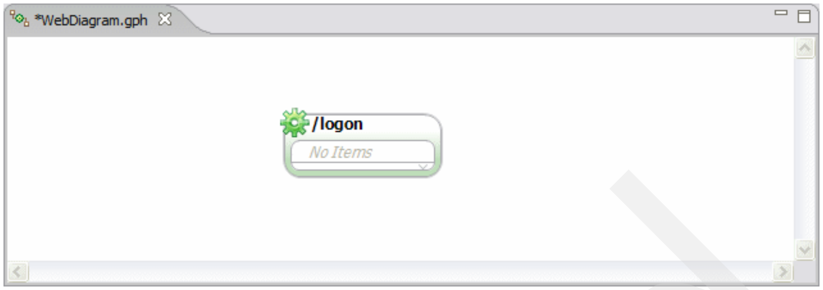


Figure 13-7 Struts Components - Create Struts action

- Notice that a `LogonAction` class is created in the `rad7strutsweb.actions` package, and the Struts configuration file is updated.

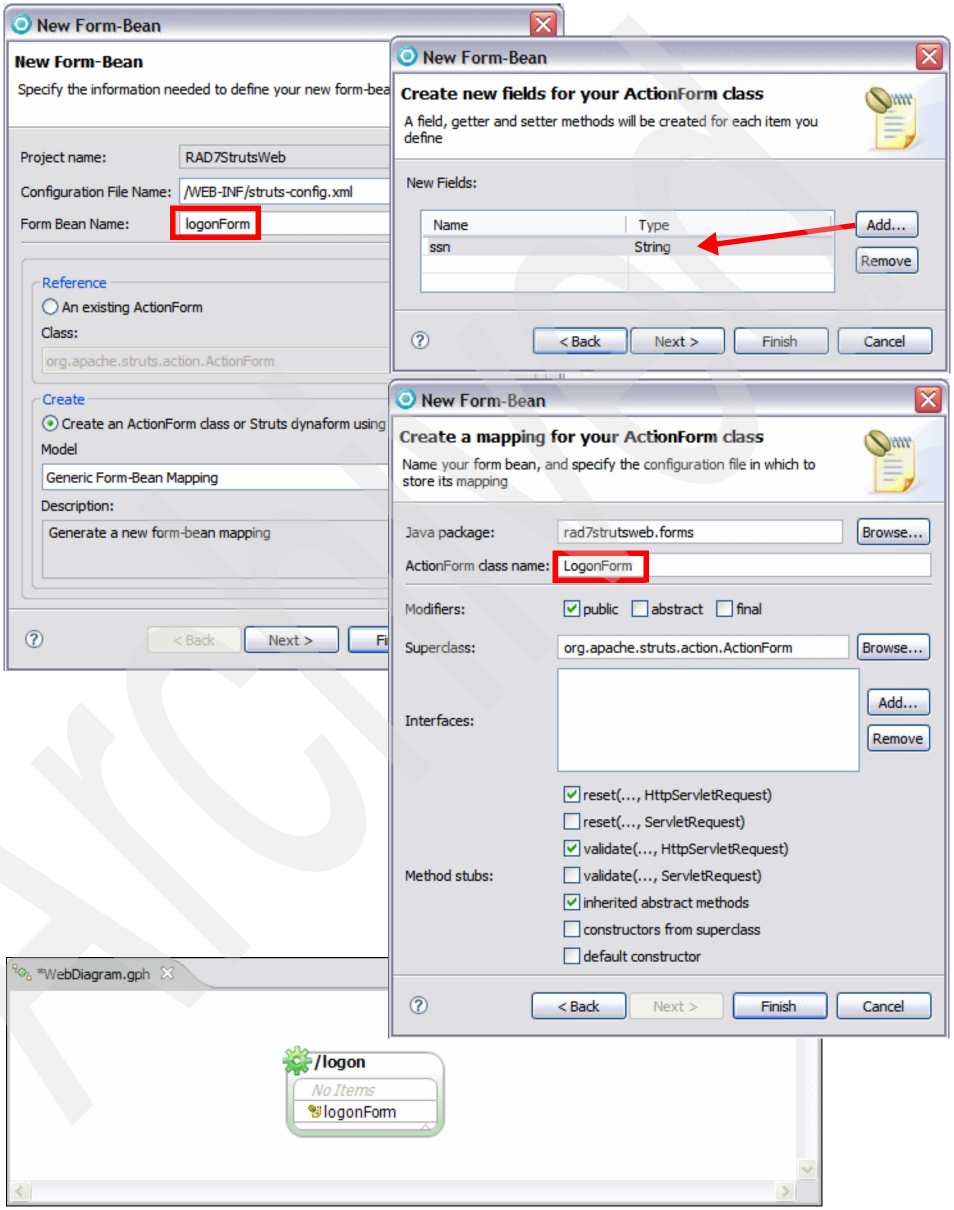
Create a Struts form bean

In the previous section we created the logon action. The logon action is invoked when the customer enters a customer ID (social security number). This information is passed to the action as a Struts form bean.

To create and associate the `logonForm` form bean to the logon action, do these steps:

- Hover with the mouse over the `/logon` action, then click the **Add Form Bean** icon ( ) from the selections presented.
- In the Form Bean Selection dialog, do these steps:
  - Click **New**.
  - In the New Form Bean dialog, type `logonForm` as the Form Bean Name and accept the defaults for the rest. Click **Next**.
  - In the Choose new fields for your `ActionForm` class dialog, you can select existing items (from projects) to be generated a field with getter and setter methods. We do not have existing fields. Click **Next**.
  - In the Create new fields for your `ActionForm` class dialog, you can add fields to the action form. Click **Add** to create a field named **ssn** of type string. Click **Next**.
  - In the Create a mapping for your ActionForm class dialog, accept the defaults for package, modifiers, superclass, and method stubs. Accept the generated name `LogonForm` for the ActionForm class name.
  - Click **Finish**.
  - The `logonForm` bean is added to the Form Bean Selection dialog. Click **OK**.

- A `LogonForm` class is created in the `rad7strutsweb.forms` package and the Struts configuration file is updated.
- Figure 13-8 shows the creation of the form bean and the result in the Web Diagram.



New Form-Bean

New Form-Bean

Specify the information needed to define your new form-bea

Project name: RAD7StrutsWeb

Configuration File Name: /WEB-INF/struts-config.xml

Form Bean Name: logonForm

Reference

○ An existing ActionForm

Class: org.apache.struts.action.ActionForm

Create

● Create an ActionForm class or Struts dynaform using

Model: Generic Form-Bean Mapping

Description: Generate a new form-bean mapping

New Form-Bean

Create new fields for your ActionForm class

A field, getter and setter methods will be created for each item you define

New Fields:

Name	Type
ssn	String

New Form-Bean

Create a mapping for your ActionForm class

Name your form bean, and specify the configuration file in which to store its mapping

Java package: rad7strutsweb.forms

ActionForm class name: LogonForm

Modifiers: ☑ public ☐ abstract ☐ final

Superclass: org.apache.struts.action.ActionForm

Interfaces:

Method stubs:
☑ reset(..., HttpServletRequest)
☐ reset(..., ServletRequest)
☑ validate(..., HttpServletRequest)
☐ validate(..., ServletRequest)
☑ inherited abstract methods
☐ constructors from superclass
☐ default constructor

Figure 13-8 Struts components: Adding a Struts form bean

Create a Web page

Thus far, we have created the `logon` action and the `logonForm` action form. We have to create the input and output pages for the logon action. The input page (`logon.jsp`) lets the customer enter a customer ID (SSN) through the input form. The form action is mapped to the logon action. The form data is passed to the action through the logonForm logon `ActionForm`.

To create the `logon.jsp` and `customerListing.jsp` Web pages, do these steps:

- From the Web Parts drawer of the Palette, drag and drop the **Web Page** icon (Web Page) into the Web diagram. Type `logon.jsp` as the page name.
- Repeat this to create the `customerListing.jsp`. Notice that empty JSPs are created in the `WebContent` folder.
- The Web Diagram is shown in Figure 13-9.

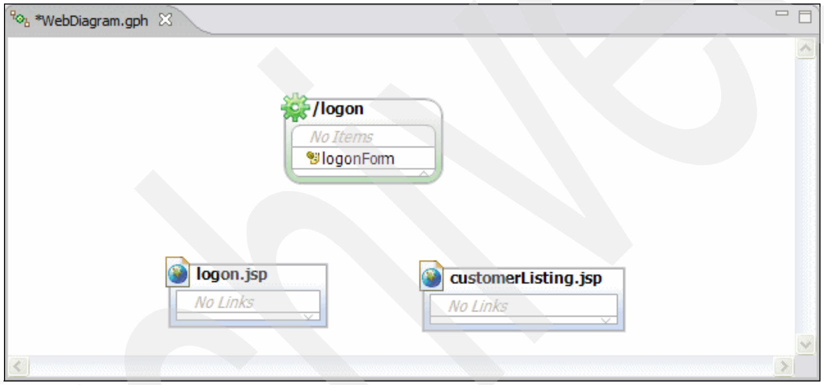


Figure 13-9 Struts components: Create Web pages

Create a Struts Web connection

A connection is typically used to connect two nodes in a Web diagram. In a Struts context, when a connection is dragged from a Struts action, a pop-up connection wizard is displayed, enabling the user to create the connection. When any of these connections are realized, the corresponding Struts action mapping entry in the Struts configuration file `struts-config.xml` is modified appropriately. As in prior sections, once the connections are drag and defined on the palette, the realization occurs automatically.

When you select **Connection** from the palette and drag it from a Struts action to any other node, you are able to create the following:

- **Local Exception**: When Local Exception is selected, the action mapping entry is modified in the `struts-config.xml` file. The handler class created during the creation is invoked when the Struts action throws the local exception.

- **Action Input**: When the Struts validation framework is used, the action needs to forward the control back to the page that invoked the action in case of validation failures. This is specified by specifying an action input. The action mapping entry is modified in the `struts-config.xml` file.
- **Include Action Mapping**: The action in the action mapping entry in `struts-config.xml` is configured as an include.
- **Forward Action Mapping**: The action in the action mapping entry in `struts-config.xml` is configured as a forward.
- **Global Forward**: When Global Forward is selected, a global forward entry is added in the `struts-config.xml` configuration file.
- **Local Forward**: When Local Forward is selected, a local forward entry is added within the corresponding action mapping entry instead of globally in the `struts-config.xml` configuration file.
- **Global Exception**: When Global Exception is selected, a global exception entry is added in the `struts-config.xml` configuration file.

In our sample, when a user enters an invalid customer ID (SSN), the logon action fails and then forwards the user back to the logon page to enable the user to re-enter this information. Likewise, if the logon action succeeds, the customer has to be forwarded to the `customerListing.jsp` that displays the customer's account information.

To create the local forwards for success and failures for the logon action, do these steps:

- Create a connection from the `logon.jsp` to the logon action. Click **Connection** in the Palette, select `logon.jsp`, and drag the connection to the logon action. Select **Struts Form** when prompted.

  This is done to indicate that the action that is to be invoked when the form is submitted is logon.
- Create an Action Input that is used by the Struts validation framework for validating the form data in `logon.jsp` to redirect the user back to the `logon.jsp` page.

  Select **Connection** in the Palette, select the logon action, and drag it to the `logon.jsp`. Select **Action Input** when prompted.
- Create a local forward back to `logon.jsp`. This is used to forward the user back to the logon page when business exceptions occur in the logon action.
  - Create a connection from the logon action to the `logon.jsp`.
  - Select **Local Forward** when prompted.
  - Rename the generated forward named success to `failure`.

- Create a local forward to the `customerListing.jsp`.
  - Create a connection from the logon action to the `customerListing.jsp`.
  - Select **Local Forward** when prompted.
  - Rename the forward to `success` (it should be success by default).
- The resulting Web Diagram is shown in Figure 13-10.

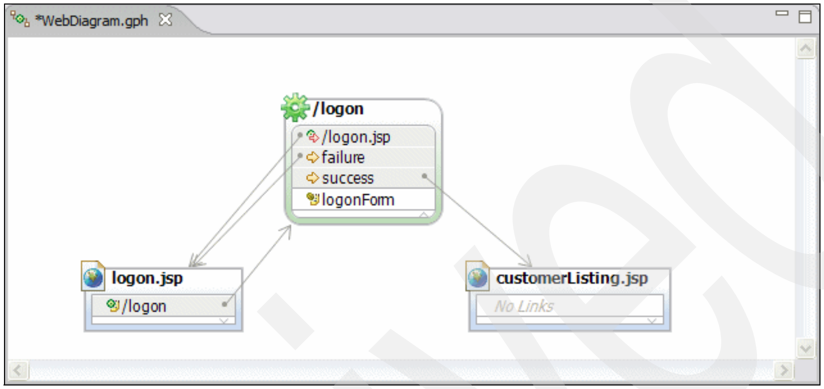


Figure 13-10 Struts components: Creating connections

- Save the Web Diagram.

> **Tip:** You can select a component in the Web diagram and overtype the name and the forward names.
>
> To improve the layout of the application flow, you can drag components to another spot. You can rearrange connections by dragging their middle point.

Realizing the Struts components

In Application Developer V7, the Web Diagram Editor automatically realizes the components drawn onto the palette. In terms of Struts components, this realization has differing consequences for each type of component. Table 13-1 describes the item generated and artifacts updated, given a realization of a specific component.

Table 13-1 Struts components realization resultst

Object	Resulting action when realized
Struts Form Beans	Creating a new form bean from within an action creates a new form bean class. In addition, it also updates the `struts-config.xml` file with a form bean mapping and form bean to action association.
Struts Actions	Dragging a new action onto the palette creates a new action class. In addition, it also updates the `struts-config.xml` with an action mapping.
Local/Global Forwards, Exceptions, Action Input Connections	Dragging a connection onto the palette updates the `struts-config.xml` file with the appropriate configuration.
JSP	Dragging a Web page onto the palette creates a new Web page JSP.

Modify application resources

The wizard created an empty `ApplicationResources.properties` file in the `rad7strutsweb.resources` package for us and we have to update it with the texts and messages for our application.

While developing Struts applications, you usually find yourself having this file open, because you typically add messages to it as you go along writing your code. Again, this properties file is meant to externalize messages and more importantly support i18n. Example 13-1 shows a completed `ApplicationResources.properties` file.

Example 13-1 ApplicationResources.properties snippet

```
# Optional header and footer for <errors/> tag.
#errors.header=<ul>
#errors.footer=</ul>
errors.prefix=<li>
errors.suffix=</li>

form.ssn=SSN
form.accountId=Account Id
form.balance=Balance
form.amount=Amount

form.accountDetails.transactionId=Transaction ID
form.accountDetails.transactionType=Transaction Type
form.accountDetails.transactionTime=Transaction Date-Time
form.accountDetails.transactionAmount=Transaction Amount
form.transaction.amount=Amount

errors.required={0} is a required Field
error.ssn=Verify that the customer ssn entered is correct.
error.amount=Verify that the amount entered is valid.
error.timeout=Your session has timed out. Please login again.
errors.systemError=The system is currently unavailable. Please try again later.
```

Initially this file only contains optional header and footer for errors. At this point you can open the file and add two lines:

```
form.ssn=SSN
errors.required={0} is a required field.
```

We use this message to validate—using the Struts Validation Framework—the logon form in `logon.jsp` to ensure that the user enters a value for the customer (SSN).

Using the Struts validation framework

The Struts validation framework provides automatic validation of forms using configuration files. The `validation.xml` and `validator-rules.xml` are the two configuration files used by the Struts validation framework to validate forms.

> **Note:** More information on the architecture and further documentation of Struts validation framework can be found at:
>
> `http://struts.apache.org`

To validate the `logonForm` using the Struts validation framework, do these steps:

- We have provided a `validation.xml` and `validation-rules.xml` as part of the sample code.

  Import the `validation.xml` and `validator-rules.xml` from the `C:\7501code\struts\WebContent\WEB-INF` directory into the `RAD7StrutsWeb\WebContent\WEB-INF` directory of the workspace.
- Add the Struts validator plug-in and required property to the plug-in indicating the location of the validation configuration files.
  - Expand **RAD7StrutsWeb** → **WebContent** → **WEB-INF**.
  - Open the `struts-config.xml` file in the Struts Configuration Editor.
  - Select the **Plug-ins** tab in the Struts Configuration Editor.

- Click **Add** for Plug-ins, select the **ValidatorPlugIn** in the Class Selection wizard, and click **OK**. The Struts validator plug-in has now been added.
- Add the required parameter by clicking **Add** in the Plug-in Mapping Extension field.
- Set the Property field to `pathnames` and the Value field to `/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml`.

- Save and close the Struts configuration file.

The `validation.xml` file contains all the Struts form beans and the fields within the form bean that is validated, and the rule that is applied to validate the bean. The snippet that validates the `logonForm` is shown in Example 13-2.

Example 13-2 validation.xml snippet - LogonForm

```
<form-validation>
   <formset>
      <form name="logonForm">
         <field property="ssn" depends="required">
            <arg0 key="form.ssn" />
         </field>
      </form>
   </formset>
   .......
   .......
</form-validation>
```

The `validator-rules.xml` file contains the rule configurations for all the rules defined in the `validation.xml` file. In our example above, the rule that is defined is *required* for the field ssn, as shown in Example 13-2. The snippet for the *required* rule is shown in Example 13-3.

Example 13-3 validator-rules.xml snippet - Rule configuration for the required rule

```
<form-validation>
   <global>
      <validator name="required"
         classname="org.apache.struts.validator.FieldChecks"
         method="validateRequired"
         methodParams="java.lang.Object,
                       org.apache.commons.validator.ValidatorAction,
                       org.apache.commons.validator.Field,
                       org.apache.struts.action.ActionErrors,
                       javax.servlet.http.HttpServletRequest"
         msg="errors.required" />
   </global>
</form-validation>
```

Page Designer and the Struts tag library

The Struts framework provides a tag library to help in the development of Struts-based Web applications. The Page Designer is used to design and develop HTML and dynamic Web pages within Application Developer. The features of the Page Designer are explained in detail in Chapter 12, “Develop Web applications using JSPs and servlets” on page 465.

Stuts tag library overview

In this section we explore the support for the Struts tag libraries within the Page Designer. The Page Designer supports the Struts tag libraries by allowing dropping of tags from the Page Designer Palette and into the design view of the Page Designer.

Open the `logon.jsp` to see the Palette in Page Designer (Figure 13-11):

- **Struts HTML tags**: Struts provides tags to render HTML content. Examples of the Struts HTML tags are button, cancel, checkbox, form, errors, etc. These tags can be dragged and dropped into a page in the page designer’s design perspective. We drag and drop the errors tag to the `logon.jsp` and display a message to the user if no ssn is entered later on in this section.
- **Struts Bean tags**: The bean tag library provides tags to access bean properties, request parameters, create page attributes, and so forth.
- **Struts Logic tags**: The Logic tag library provides tags to implement conditional, looping, and control related functionality.
- **Struts Nested tags**: The Nested tag library provides tags to access complex beans with nested structures.
- **Struts Template tags**: The Template tag library provides tags for creation of dynamic JSP templates.
- **Struts Tiles Tags**: The Tiles tag library facilitates development of dynamic Web applications in a tiled fashion where the tile can be reused throughout the application.

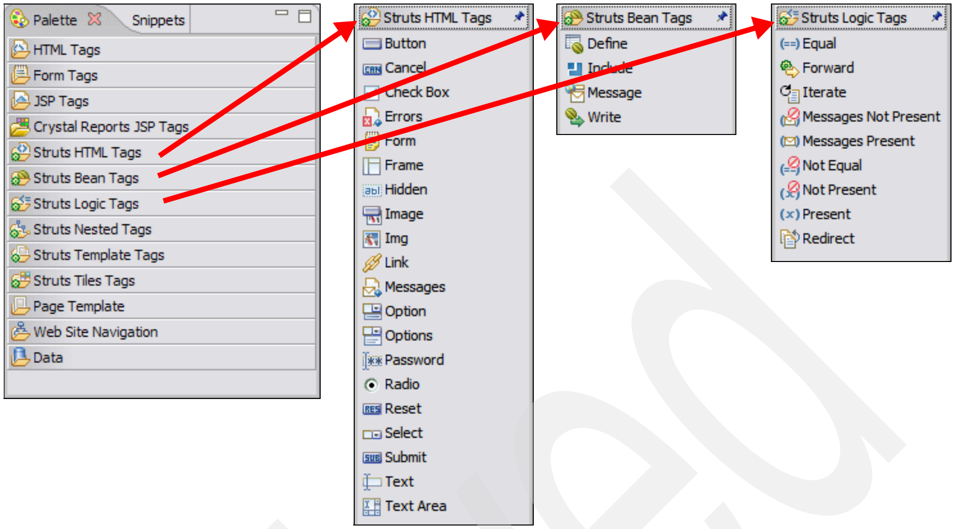


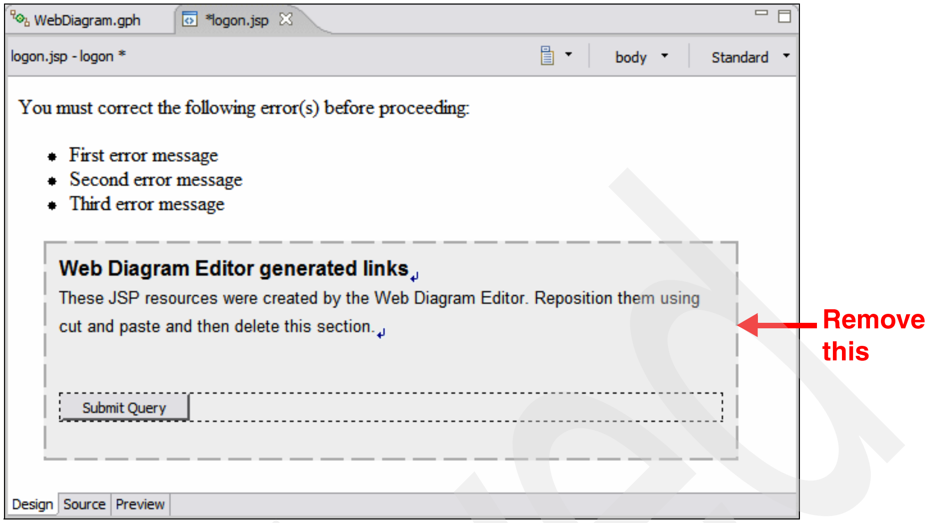
Figure 13-11 Struts tag library support: Page Designer Struts tag drawers

> **Tip:** If you do not see all the Struts tag libraries in the palette, right-click the palette and select **Customize**. From the Customize Palette dialog box, select all drawers you want to have available in the palette. Note that you must be in the process of editting a JSP in order to see the Struts tag libraries options.

Completing the logon JSP

We now add a simple HTML error tag to `logon.jsp`, which displays an HTML error message that occurs due to the validation check by the Struts Validation Framework. To add the HTML error tag to `logon.jsp`, do these steps:

- Open the `logon.jsp` in Page Designer.
- Select the **Design** tab of the Page Designer.
- Select the **Errors** icon from the Struts HTML Tags drawer and drop it at the top of the page.
- The Page Designer renders the HTML for the tag in the design view as shown in Figure 13-12.



You must correct the following error(s) before proceeding:

- First error message
- Second error message
- Third error message

Web Diagram Editor generated links

These JSP resources were created by the Web Diagram Editor. Reposition them using cut and paste and then delete this section.

Remove this

Figure 13-12 Struts tags: Struts errors tag rendered by Page Designer

Creating the input form

The input form contains the customer SSN and a submit button. The form with the button was created when we linked the JSP to the action. Now we have to add the fields of the form:

- Select the shaded area in the Outline view (`<div>` tag) and select **Remove**. We will generate the content using the action form.
- In the Page Data view select **Struts Form Beans** → **logonForm** and drag it into the JSP form above the button (Figure 13-13).

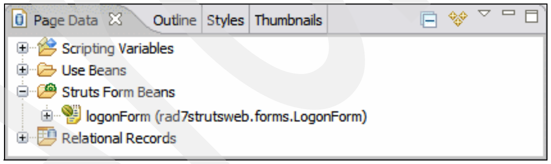
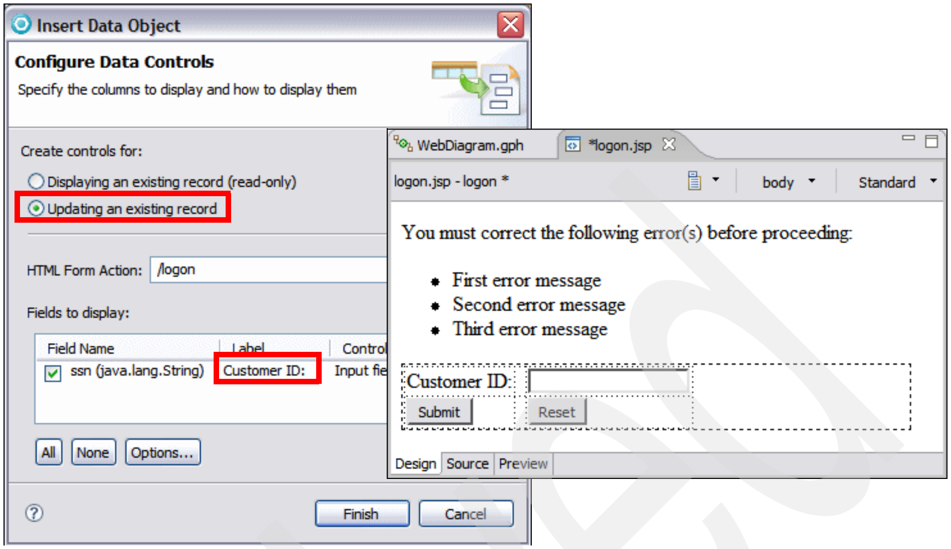


Figure 13-13 Page Data view with form bean

- In the Configure Data Controls dialog (Figure 13-14), select **Updating an existing record** and change the label of the ssn field to **Customer ID:**. The HTM Form Action is prefilled with /logon.
- Click **Options** and select **Submit** and clear **Delete**. Click **OK**.
- Click **Finish** and the form is generated (Figure 13-14).


Insert Data Object

Configure Data Controls

Specify the columns to display and how to display them

Create controls for:

○ Displaying an existing record (read-only)

● Updating an existing record

HTML Form Action: /logon

Fields to display:

Field Name	Label	Control
☑ ssn (java.lang.String)	Customer ID:	Input fie

You must correct the following error(s) before proceeding:

- First error message
- Second error message
- Third error message

Customer ID:

Figure 13-14 Creating the logon JSP with a form

- The source for the rendered logon.jsp is shown in the Example 13-4. Notice the `<html:errors />` and the form that the wizard has added for rendering the action form.

Example 13-4 Struts tags:- Logon.jsp snippet of the tag in the source view

```
......
<html:errors />
<html:form action="/logon">
   <TABLE>
      <TBODY>
         <TR>
            <TD align="left">Customer ID:</TD>
            <TD width="5">&nbsp;</TD>
            <TD><html:text property="ssn"></html:text></TD>
         </TR>
         <TR>
            <TD align="left"><html:submit property="Submit"
                  value="Submit"></html:submit></TD>
            <TD width="5">&nbsp;</TD>
            <TD><html:reset /></TD>
         </TR>
      </TBODY>
   </TABLE>
</html:form>
......
```

Completing the logon action

The skeleton code for the `LogonAction` class was created from the Web Diagram and is shown in Example 13-5.

Example 13-5 Skeleton action class (compressed)

```
package rad7strutsweb.actions;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class LogonAction extends Action {

   public ActionForward execute(ActionMapping mapping, ActionForm form,
                   HttpServletRequest request, HttpServletResponse response)
            throws Exception {
      ActionErrors errors = new ActionErrors();
      ActionForward forward = new ActionForward(); // return value
      try {
            // do something here
      } catch (Exception e) {
            // Report the error using the appropriate name and ID.
            errors.add("name", new ActionError("id"));
      }

        // If a message is required, save the specified key(s)
        // into the request for use by the <struts:errors> tag.
        if (!errors.isEmpty()) {
            saveErrors(request, errors);
            // Forward control to the appropriate 'failure' URI (....)
            //forward = mapping.findForward(failure");
        } else {
            // Forward control to the appropriate 'success' URI (....)
            forward = mapping.findForward("success");
        }
        // Finish with
        return (forward);
    }
}
```

We have to complete the execute method with our logic. The final code is shown in Example 13-6.

Example 13-6 Completed execute method of the logon action class

```
public class LogonAction extends Action {

public ActionForward execute(ActionMapping mapping, ActionForm form,
              HttpServletRequest request, HttpServletResponse response)
            throws Exception {

   ActionErrors errors = new ActionErrors();
   ActionForward forward = new ActionForward(); // return value

   LogonForm logonForm = (LogonForm)form;
   String ssn = logonForm.getSsn();

   try {
      CustomerBean customerBean = new DatabaseAccess().getCustomer(ssn);
      request.setAttribute("customer", customerBean);
      request.getSession().setAttribute("ssn", logonForm.getSsn());

   } catch (InvalidCustomerException e){
      errors.add("ssn", new ActionError("error.ssn"));
   } catch (Exception e) {
      errors.add("error", new ActionError("errors.systemError"));
   }
   // If a message is required, save the specified key(s)
   // into the request for use by the <struts:errors> tag.
   if (!errors.isEmpty()) {
      saveErrors(request, errors);
      // Forward control to the appropriate 'failure' URI (........)
       forward = mapping.findForward("failure");
   } else {
      // Forward control to the appropriate 'success' URI (.......)
       forward = mapping.findForward("success");
   }
   // Finish with
   return (forward);
}
}
```

Let us understand some of the action class coding:

- The ActionForm parameter is cast to the correct LogonForm class and the ssn value is extracted.
- The customer and accounts are retrieved from the database using the DatabaseAccess class.

- The ssn is stored as session data and the customer information is stored in the request block. The session data is available to all actions and JSPs.
- If exceptions are thrown by the database access, an ActionError is added to the Struts errors list. The error text ("error.ssn" or "errors.systemError") comes from the ApplicationResources file (see Example 13-1 on page 560).
- An ActionForward of either "failure" or "success" is returned. Struts will find the appropriate resulting action or JSP in the configuration file.

Helper classes

The action methods are supported by helper classes provided in:

```
C:\7501code\struts\bank\beans
C:\7501code\struts\bank\database
```

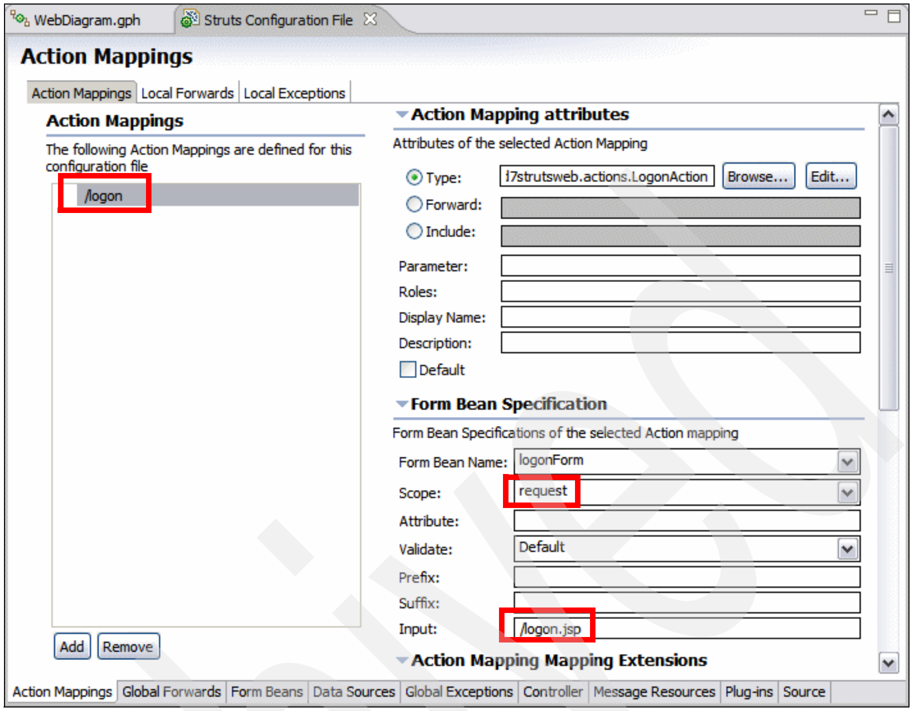
- JavaBean data transfer objects: CustomerBean, AccountBean, and TransactionBean (rad7strutsweb.beans package). These classes are simple Java beans with a few properties.
- Database retrieve and update: DatabaseAccess and DataSourceFactory (rad7strutsweb.database package). The DatabaseAccess class provides methods to retrieve the customer with accounts and an account with transactions, and to update the account balance and create a transaction record.

  For example, to retrieve the customer with accounts for the logon action:

  - A data source connection to the database is created using the DataSourceFactory utility class. This factory can be used by all actions.
  - The SQL statement (CUSTOMER_STMT) is executed to retrieve customer and account information using a join.
  - The result set is retrieved. Customer information is stored in the CustomerBean and account information is stored in an accountList (of AccountBean objects), which is then added to the CustomerBean.
  - An exception is thrown if the customer is not found or if any other SQL exception occurs.

Using the Struts Configuration Editor

Application Developer provides an editor for the Struts struts-config.xml configuration file. This editor is yet another way you can add new form beans and actions and customize their attributes. You can also directly edit the XML source file, should you prefer to do this manually instead of by using the wizards. The Struts Configuration Editor is shown in Figure 13-15.


WebDiagram.gph
Struts Configuration File
Action Mappings
Action Mappings
Local Forwards
Local Exceptions
Action Mappings
The following Action Mappings are defined for this configuration file
Add
Remove
Action Mapping attributes
Attributes of the selected Action Mapping
Browse...
Edit...
Form Bean Specification
Form Bean Specifications of the selected Action mapping
Action Mapping Mapping Extensions
Action Mappings
Global Forwards
Form Beans
Data Sources
Global Exceptions
Controller
Message Resources
Plug-ins
Source

/logon

● Type: d7strutsweb.actions.LogonAction
○ Forward:
○ Include:
Parameter:
Roles:
Display Name:
Description:
□ Default

Form Bean Name: logonForm
Scope: request
Attribute:
Validate: Default
Prefix:
Suffix:
Input: /logon.jsp

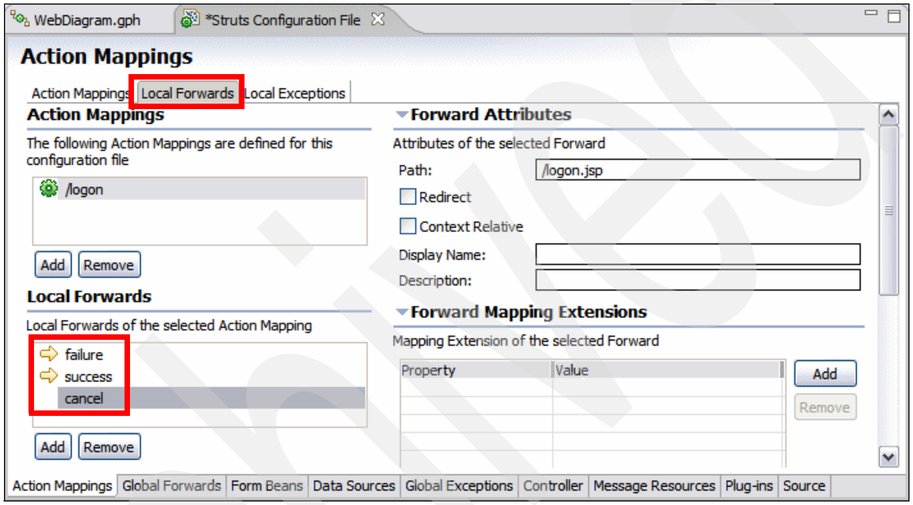
Figure 13-15 Struts Configuration Editor

We use this editor to add a local forward called *cancel* to the logon action. This forward can be used by the logon action's execute method to forward the user to the `logon.jsp` page, as we map this forward to the `logon.jsp` page.

We also specify the input attribute for all our actions. This attribute specifies which page should be displayed if the validate method of a form bean or the Struts validation framework fails to validate. Usually you want to display the input page where the user entered the data so they can correct their entry.

- Open the **struts-config.xml** file under **RAD7StrutsWeb** → **WebContent** → **WEB-INF**.
- The editor has tabs at the bottom of the screen to navigate between the different Struts artifacts it supports.
  - In the **Action Mappings** tab select the **/logon** action. You can see the `logonForm` (Form Bean Name) and the `logon.jsp` (Input).
  - Notice the Scope value of `request`; an alternative would be `session`.

- Create a forward:
  - Select the **Local Forwards** tab found at the top of the Action Mappings page. Select the **/logon** action. We already have two forwards named failure and success.
  - Click **Add** to specify in the local forwards section. A new forward with the name `<no name>` is created.
  - Overtype the name with `cancel` and enter `/logon.jsp` in the Path field in the Forward Attributes (Figure 13-16).



The following Action Mappings are defined for this configuration file

/logon

Local Forwards of the selected Action Mapping

failure
success
cancel

Attributes of the selected Forward

Path: /logon.jsp

□ Redirect

□ Context Relative

Display Name:

Description:

Mapping Extension of the selected Forward

Figure 13-16 Struts Configuration Editor: Creating new forward

> **Note:** The Redirect check box allows you to select if a redirect or forward call should be made. A forward call keeps the same request with all attributes it contains and just passes control over to the path specified. A redirect (or send redirect) call tells the browser to make a new HTTP request, which creates a new request object (and you lose any attributes set in the original request).
>
> A forward call does not change the URL in the browser's address field, as it is unaware that the server has passed control to another component. With a redirect call, however, the browser updates the URL in its address field to reflect the requested address.
>
> You can also redirect or forward to other actions. It does not necessarily have to be a JSP.

- Save the file.

- Select the **Source** tab to look at the Struts configuration file. Example 13-7 shows parts of the `struts-config.xml` XML source.

Example 13-7 Struts configuration file: struts-config.xml

```
<?xml version="1.0"........>
<struts-config>
   <!-- Data Sources -->
   ......
   <!-- Form Beans -->
   <form-beans>
      <form-bean name="logonForm" type="rad7strutsweb.forms.LogonForm">
      </form-bean>
   </form-beans>
   <!-- Global Exceptions -->
   ......
   <!-- Global Forwards -->
   ......
   <!-- Action Mappings -->
   <action-mappings>
      <action path="/logon" type="rad7strutsweb.actions.LogonAction"
               name="logonForm" scope="request" input="/logon.jsp">
         <forward contextRelative="false" name="failure" path="/logon.jsp">
         </forward>
         <forward contextRelative="false" name="success"
                     path="/customerListing.jsp">
         </forward>
         <forward name="cancel" path="/logon.jsp">
         </forward>
      </action>
   </action-mappings>

   <!-- Message Resources -->
   <message-resources
         parameter="rad7strutsweb.resources.ApplicationResources"/>
   <plug-in className="org.apache.struts.tiles.TilesPlugin">
      <set-property property="definitions-config"
            value="/WEB-INF/tiles-defs.xml"/>
      <set-property property="definitions-parser-validate" value="true"/>
      <set-property property="moduleAware" value="true"/>
   </plug-in>
   <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
      <set-property property="pathnames"
         value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
   </plug-in>
</struts-config>
```

As you can see, the Application Developer Struts tools have defined the `logonForm` bean in the `<form-beans>` section and the `logon` action in the

<action-mappings> section. The JSPs, however, are not specified in the Struts configuration file. They are completely separate form the Struts framework (only the forwarding information is kept in the configuration file). At the end of the file is the name of the application resources file where our texts and error messages are stored.

The Struts Configuration Editor does round-trip editing, so if you edit something in the XML view it is reflected in the other views.

The forwards we use are local to each action, meaning that only the action associated with the forward can look it up. In the <global-forwards> section you can also specify global forwards that are available for all actions in the application. Normally you have a common error page to display any severe error messages that may have occurred in your application and that prevents it from continuing. Pages like these are good targets for global forwards, and so are any other commonly used forward. Local forwards override global forwards.

- Close the configuration file editor.

Notes: We recommend that all requests for JSPs go through an action class so that you have control over the flow and can prepare the view beans (form beans) necessary for the JSP to display properly. Struts provides simple forwarding actions that you can use to accomplish this.

In our example we do not perform any customization on the Struts action servlet (org.apache.struts.action.ActionServlet). If you have to do any custom life cycle processing needed to be executed, you would want to create your own action servlet, extending the ActionServlet class, and overriding the appropriate ActionServlet methods. You would then also modify the \WEB-INF\web.xml file and replace the name of the Struts ActionServlet with the name of your action servlet class.

Completing the application

Completing the application as shown in Figure 13-3 on page 549 would be quite complex. Here are abbreviated instructions if you want to try this yourself.

Alternatively, import the solution by using the instructions in “Importing the final sample application” on page 585. Then study the code described in this section.

Complete the Web Diagram

This is how we complete the Web Diagram:

- Open the Web Diagram and rearrange the tree existing components to the layout of Figure 13-3 on page 549.

- Add three actions: logoff, accountDetails, and performTransaction.
- Add three Web pages (JSPs): accountDetails.jsp, transact.jsp, and error.jsp.
- Create two action forms:
  - In the accountDetails action create the customerAccountForm with two attributes (String ssn and String accountId).
  - In the performTransaction action create the transactForm with three attributes (String accountId, String amount, java.math.BigDecimal balance).
- Create connections according to Table 13-2.

Table 13-2 Web Diagram connections

From	To	Type of connection
/logoff	logon.jsp	Static Forward
customerListing.jsp	/logoff	Struts Link
	/accountDetails	Struts Link
	transact.jsp	Struts Link
/accountDetails	accountDetails.jsp	Local Forward success
	error.jsp	Local Forward failure
accountDetails.jsp	/logon	Struts Link
	/logoff	Struts Link
	transact.jsp	Struts Link
transact.jsp	/performTransaction	Struts Form
/performTransaction	/accountDetails	Local Forward success
	error.jsp	Local Forward failure
error.jsp	/logoff	Struts Link
/logon	logon.jsp	Local Forward cancel

Import the Java code and the JSPs

The Java source code for the actions. forms, beans, exceptions, and utility classes are provided in folders under:

```
C:\7501code\struts\bank
```

Note: You can drag a source file from Windows Explorer into the appropriate package in Application Developer:

- Copy the database access classes `DatabaseAccess` and `DataSourceFactory` into the `rad7stutsweb.database` package.
- Copy the `Applicationresources.properties` file into the `rad7stutsweb.resources` package.
- Copy the action forms (`CustomerAccountForm`, `LogonForm`, `TransactForm`) into the `rad7stutsweb.forms` package.
- Copy the data transfer beans (`AccountBean`, `CustomerBean`, `TransactBean`) into the `rad7stutsweb.beans` package.
- Copy the exceptions (`InvalidCustomerException`, `InvalidTransactionException`) into the `rad7stutsweb.exceptions` package.
- Copy the actions (`AccountDetailsAction`, `LogoffAction`, `LogonAction`, `PerformTransactionAction`) into the `rad7stutsweb.actions` package.
- Copy the `rbhome.gif` image into the `theme` folder. This image is used by the Web pages.
- Copy the JSPs (`accountDetails`, `customerListing`, `error`, `logon`, `transact`) into the `WebContent` folder.
- Optionally import the `struts-config.xml` file into `WebContent\WEB-INF`.
- Optionally import the `WebDiagram.gph` file into the project. When you open the Web Diagram the action forms may not be visible in the actions. Double-click in the small space at the bottom to display the action form (Figure 13-17).

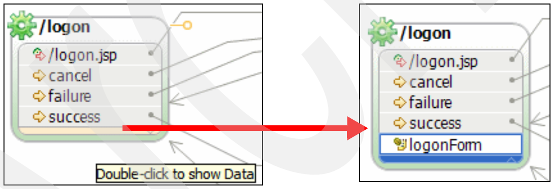


Figure 13-17 Web Diagram action with and without the action form

Study the sample code

Study the sample code, especially the actions and the JSPs:

- The data transfer beans and action forms are easy to understand; they are standard Java beans.
- The `DatabaseAccess` class provides methods to retrieve customer, account, and transaction information from the database, and to update the account balance and create a transaction record.

- The DataSourceFactory class retrieves a data source with the JNDI name of jdbc/itsobank. This data source must be defined in the server.
- The AccountDetailsAction retrieves one customer account and the transactions of that account and stores the data in the data transfer beans.
- The PerfomTransactionAction adds or subtracts the transaction amount from the balance, with a possible exception if the amount is too big for a withdraw. Then the database is updated: The ACCOUNT table is updated with the new balance, and a transaction record is added to the TRANSACTIONS table.
- The customerListing.jsp displays the customer name and the list of accounts. Notice a number of Struts tags are used:
  - Additional tag library:

    ```
    <%@taglib uri="http://jakarta.apache.org/struts/tags-logic"
                                                    prefix="logic"%>
    ```

  - <bean:write>: display data from a JavaBean:

    ```
    <bean:write name="customer" scope="request" property="firstName" />
    ```

  - <html:link>: An HTML link to another Web page or action:

    ```
    <html:link action="/logoff">logoff</html:link>
    <html:link page="/transact.jsp" name="queryParms"> Deposit/Withdraw
                                                          </html:link>
    ```

    Parameters for the link (accountId, balance) are stored in a HashMap.

  - <logic:present>: Verify and retrieve session and request data:

    ```
    <logic:notPresent name="ssn" scope="session">
       <logic:redirect page="/logon.jsp" />
    <logic:present name="ssn" scope="session">
    <logic:present name="customer" property="accounts" scope="request">
    ```

  - <logic:iterate>: Loop over a property:

    ```
    <logic:iterate name="customer" property="accounts" scope="request"
                                                 id="account">
    ```

- The accountDetails.jsp displays the account information and the list of transactions of that account:
  - <bean:message>: Retrieve a text from the application resource file:

    ```
    <bean:message key="form.accountId" />
    ```

  - <nested> tags: Nested loops

    ```
    <nested:iterate name="customer" property="accounts" id="account">
    <nested:write name="account" property="balance"/>
    <nested:present name="account" property="transactions">
    ```

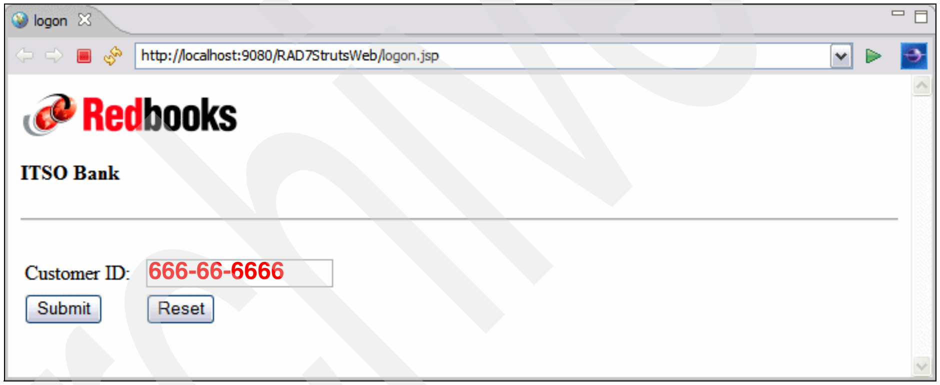
- The transact.jsp displays the account information and a form to submit a transaction amount. The error.jsp displays the Struts errors.

Running the Struts Bank Web application

In this section we run the sample application and explore the functionality built using Application Developer and its support for rapid development of Struts-based Web applications.

To run the sample ITSO Bank application, do these steps:

- Start the WebSphere Application Server v6.1 if it is not running.
- Expand **RAD7StrutsWeb** → **WebContent**, right-click the `logon.jsp` and select **Run As** → **Run on Server**.
- In the Server Selection wizard select **Choose and existing server**, select the **WebSphere Application Server v6.1** server, select **Set server as project default**, and click **Finish**.
- The home page `logon.jsp` is displayed (Figure 13-18).



http://localhost:9080/RAD7StrutsWeb/logon.jsp

ITSO Bank

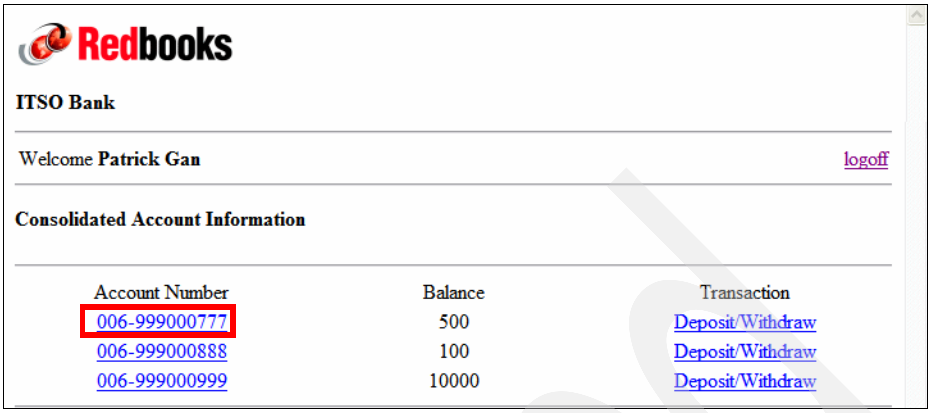
Customer ID: 666-66-6666

Figure 13-18 Running the sample: Logon.jsp

- If you click **Submit** without entering an ID, the Struts Validation Framework is activated and displays the error message added to the resources:

  ```
  Verify that the customer ssn is entered correct
  ```

- Enter a sample ID of `666-66-6666` and click **Submit**. The customer and a list of accounts is displayed (Figure 13-19). Here the logon action is responsible for retrieving this information. In a real world application, the logon action talks to the business tier to retrieve this information. The business tier architecture is described in detail in Chapter 16, “Develop Web applications using EJBs” on page 719.



Redbooks

ITSO Bank

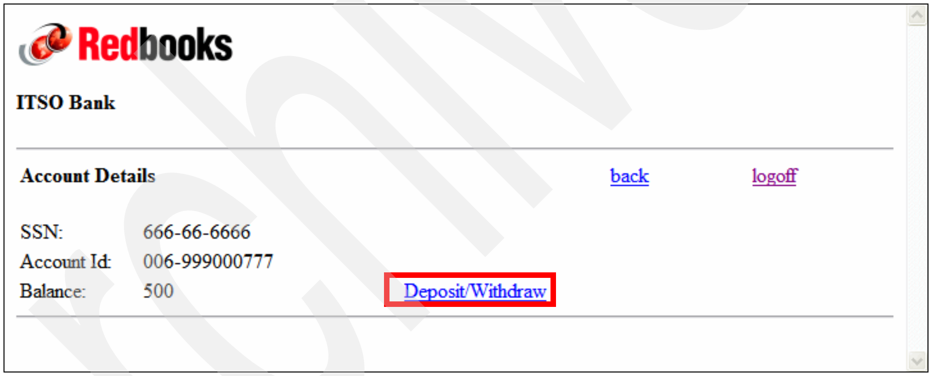
Welcome Patrick Gan logoff

Consolidated Account Information

Account Number	Balance	Transaction
006-999000777	500	Deposit/Withdraw
006-999000888	100	Deposit/Withdraw
006-999000999	10000	Deposit/Withdraw

Figure 13-19 Running the sample: Account listing

- To view details of a particular account, click the Account Number (for example, `006-999000777`) and the account information is listed. At this point there may be no transactions (Figure 13-20).



Redbooks

ITSO Bank

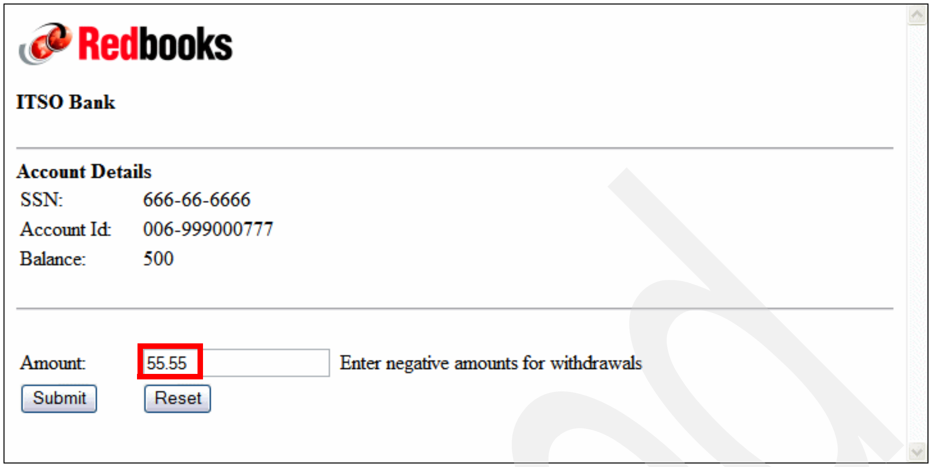
Account Details back logoff

SSN:	666-66-6666
Account Id:	006-999000777
Balance:	500

Deposit/Withdraw

Figure 13-20 Running the sample: Account details

- To perform a transaction on the account, click **Deposit/Withdraw**. The link is also available in the account listing (click **back** to get to the listing again).
- The transaction page is displayed. Enter a positive number for a deposit and a negative number for a withdrawal. We enter in `55.55` to indicate a deposit and click **Submit** (Figure 13-21).



ITSO Bank

Account Details

SSN: 666-66-6666

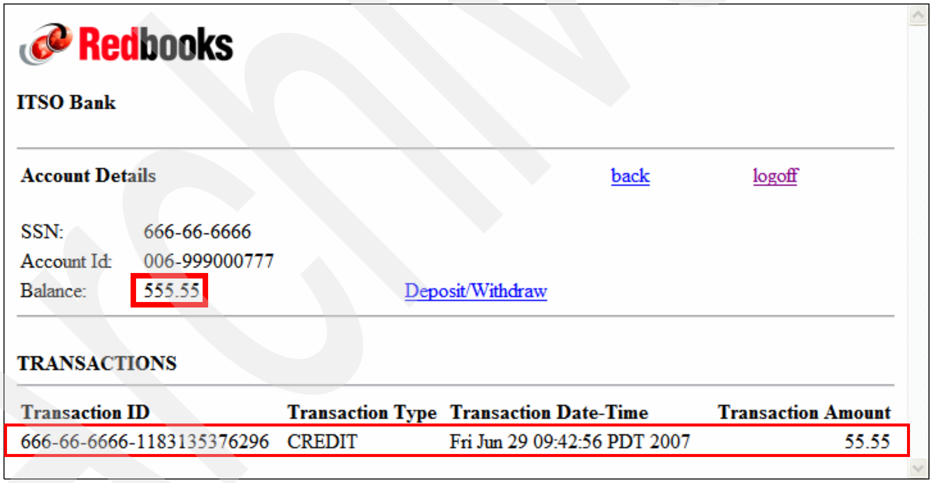
Account Id: 006-999000777

Balance: 500

Amount: 55.55 Enter negative amounts for withdrawals

Figure 13-21 Running the sample: Performing transactions

- The Account Details page is redisplayed with an entry made for the last transaction (Figure 13-22).



ITSO Bank

Account Details

SSN: 666-66-6666

Account Id: 006-999000777

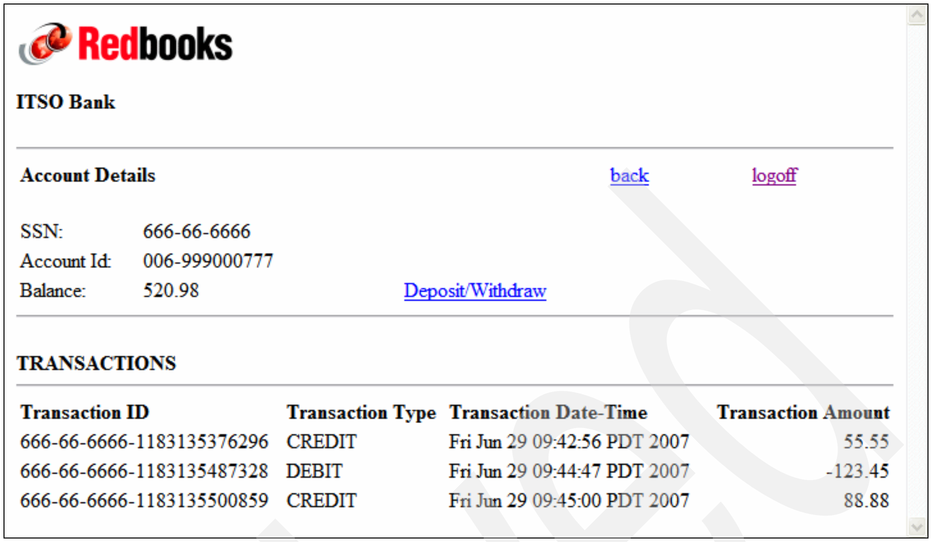
Balance: 555.55

TRANSACTIONS

Transaction ID	Transaction Type	Transaction Date-Time	Transaction Amount
666-66-6666-1183135376296	CREDIT	Fri Jun 29 09:42:56 PDT 2007	55.55

Figure 13-22 Running the sample: Transaction listing

- Run a few more transactions and the list of transaction grows (Figure 13-23).



Redbooks

ITSO Bank

Account Details back logoff

SSN:	666-66-6666
Account Id:	006-999000777
Balance:	520.98

Deposit/Withdraw

TRANSACTIONS

Transaction ID	Transaction Type	Transaction Date-Time	Transaction Amount
666-66-6666-1183135376296	CREDIT	Fri Jun 29 09:42:56 PDT 2007	55.55
666-66-6666-1183135487328	DEBIT	Fri Jun 29 09:44:47 PDT 2007	-123.45
666-66-6666-1183135500859	CREDIT	Fri Jun 29 09:45:00 PDT 2007	88.88

Figure 13-23 Running the sample: More transactions

- Enter an invalid amount (for example, alphabetic) or a withdraw amount greater than the balance, and you get the exception error:

```
Verify that the amount entered is valid.
```

- Use the **back** and **logoff** links to go back to the logon panel.

Developing a Struts Web application using Tiles

Apache Tiles is a templating framework built to simplify the development of Web application user interfaces. Apache Tiles allows application developers and page authors to define page fragments (tiles) which are assembled into a complete page at runtime. These tiles are a step up from `jsp:include` directives and can be used to reduce the duplication of common page elements or embedded within other tiles to develop a series of reusable templates. These templates streamline the development of a consistent look and feel across an entire application.

For more information on Tiles, please visit the following site:

http://tiles.apache.org/

Application Developer now provides support for Tiles framework. However, the support for Tiles is rather minimal. Nevertheless, we are going to show the extent of Tiles support.

Application Developer tooling supports Tiles in that one has the option to enable Tiles support for a Dynamic Web project. In our prior example, when creating a Dynamic Web project, we specificied to select Dynamic Page Template Support. This option enables the Web project with Tiles (by creating the Tiles configuration and updating the Struts configration to use Tiles).

The Tiles configuration file (`tiles-def.xml`) is created under `WebContent/WEB-INF`. In addition, the `struts-config.xml` file is updated with a plug-in configuration (Example 13-8).

Example 13-8 Plug-in configuration for Tiles in struts-config.xml file

```
<plug-in className="org.apache.struts.tiles.TilesPlugin">
   <set-property property="definitions-config"
                 value="/WEB-INF/tiles-defs.xml"/>
   <set-property property="definitions-parser-validate" value="true"/>
   <set-property property="moduleAware" value="true"/>
</plug-in>
```

There is also another piece of configuration that you have to put in manually. In the `struts-config.xml`, after the Action Mappings section, insert the line shown in Example 13-9.

Example 13-9 Controller configuration for Tiles

```
<controller processorClass="org.apache.struts.tiles.TilesRequestProcessor" />
```

As a reference, these two entries in the `struts-config.xml` informs Struts that Tiles is used in this Web application. Specifically, we can now use `ActionForward` to forward control onto a Tiles configuration (among other things).

Building the Tiles application extension

Now, let us build example using Tiles.

Tile actions with local forward

Open the Web Diagram, as we have done in the prior sections, use the Struts Action button (Struts Action)to create actions with names of `/contactUs` and `/aboutUs`.

For each action, create a local forward by selecting the action and **Add Link → Local Forward** (or use the arrow icon when hovering over the action). Accept the default local forward name of success (Figure 13-24).

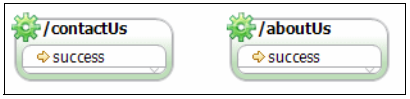
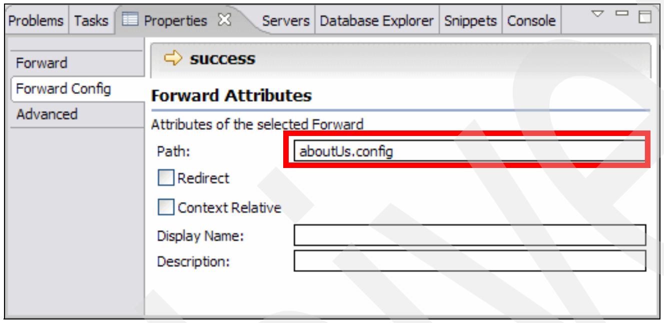


Figure 13-24 Actions for Tiles

- For `/contactUs`, have the path of the `success` local forward point to `contactUs.config`, and for `/aboutUs` to `aboutUs.config`.:
- The path value can be set in the Properties view, Forward Config tab, when selecting the `success` local forward (Figure 13-25).



Path: aboutUs.config

□ Redirect

□ Context Relative

Display Name:

Description:

Figure 13-25 Editing the Forward Config for local forwards

Add two connections from `customerListing.jsp` to the two actions, and select **Struts Link** when prompted.

Tiles configuration file

Open the Tiles configuration file `tiles-def.xml` and add these definitions (use the file `C:\7501code\struts\tiles\tiles-def.xml`):

```
<!--Define each of your definitions here-->
<definition name="contactUs.config" path="/itsoBankTemplate.jsp">
   <put name="header" value="/itsoBankHeaderTile.jsp"/>
   <put name="body" value="/itsoBankContactUsTile.jsp"/>
</definition>
<definition name="aboutUs.config" path="/itsoBankTemplate.jsp"
                                 extends="contactUs.config">
   <put name="body" value="/itsoBankAboutUsTile.jsp"/>
</definition>
```

The `contactUs` Tile page is built using a template (`itsoBankTemplate.jsp`), a header (`itsoBankHeaderTile.jsp`), and a body (`itsoBankContactUsTile.jsp`). The `aboutUs` Tile page extends the `contactUs` page with another body.

Tiles Web pages

Import the Tiles Web pages (from `C:\7501code\struts\tiles`):

```
itsoBankAboutUsTile.jsp
itsoBankContactUsTile.jsp
itsoBankHeaderTile.jsp
itsoBankTemplate.jsp
```

- The template points to the header and body pages.
- The header includes the image and `<H1>ITSO Bank</H1>`.
- The body pages contain descriptive text.

Add the links to the Struts Web page

The two Struts links have been added to the `customerListing.jsp`. Copy the two links from the `<div>` section at the top to a place before the logoff link:

- Generated code:

```
<html:link action="/contactUs">Struts Link Label</html:link><br>
<html:link action="/aboutUs">Struts Link Label</html:link><br>
```

- Final code:

```
<TD width="128" align="right"><html:link action="/contactUs">
                                Contact Us</html:link></TD>
<TD width="128" align="right"><html:link action="/aboutUs">
                                About Us</html:link></TD>
<TD width="128" align="right"><html:link action="/logoff">
                                logoff</html:link></TD>
```

Delete the `<div>` section after copying the links into the HTML table.

Tiles recapitulation

We have created and/or imported the necessary components. In essence, we have created two action classes which each (when executed successfully) forwards to a Tiles configuration. The Tiles configuration contains information on how the eventual page should render. The actual page is constructed at runtime.

The action entries in the `struts-config.xml` file have this format:

```
<action path="/contactUs" type="rad7strutsweb.actions.ContactUsAction">
   <forward name="success" path="contactUs.config">
   </forward>
</action>
<action path="/aboutUs" type="rad7strutsweb.actions.AboutUsAction">
   <forward name="success" path="aboutUs.config">
   </forward>
</action>
```

Notice the match in logical name between the path value of the success local forward within the /contactUs action mapping entry in the struts-config.xml and the contactUs.config entry in the tiles-def.xml.

The ContactUsAction and AboutUsAction classes were generated automatically when we added the actions to the Web Diagram. No code change is necessary.

Tiles runtime behavior

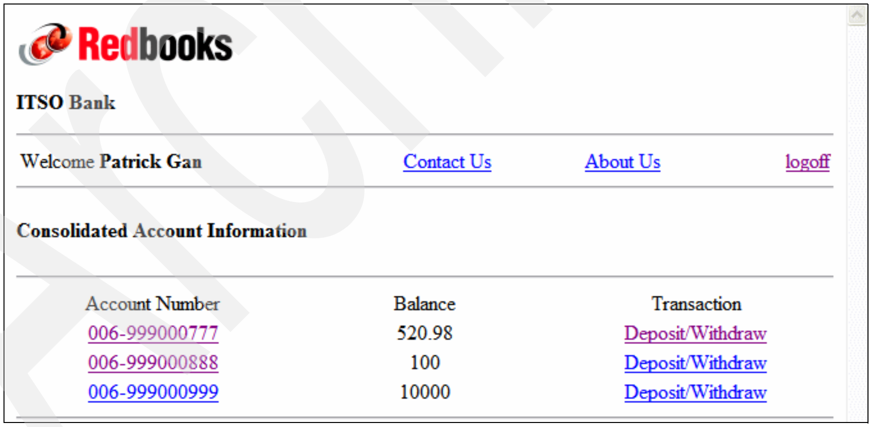
When a /contactUs request arrives, Struts invokes the ContactUsAction class, which forwards a success local forward. Struts forwards control to the contactUs.config Tiles configuration. This configuration is based on the itsoBankTemplate.jsp, with a header (itsoBankHeaderTile.jsp) and a body (itsoBankContactUsTile.jsp). The assembly of the page displayed occurs at runtime.

The processing for an /aboutUs request is similar, however, the aboutUs configuration extends the contactUs configuration and inherits the header.

Running the Tiles application

To test the Tiles application, start with the **logon.jsp** → **Run As** → **Run on Server**:

- Logon as a customer and the accounts are displayed (Figure 13-26).



ITSO Bank

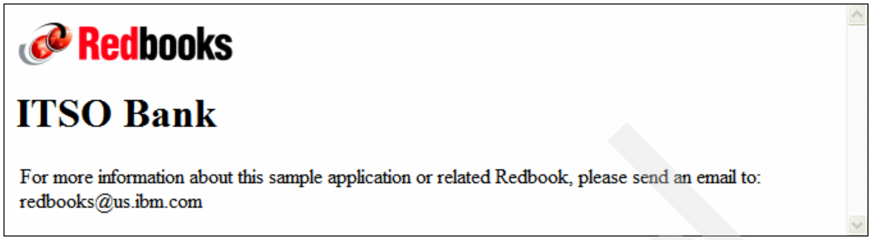
Welcome **Patrick Gan** Contact Us About Us logoff

Consolidated Account Information

Account Number	Balance	Transaction
006-999000777	520.98	Deposit/Withdraw
006-999000888	100	Deposit/Withdraw
006-999000999	10000	Deposit/Withdraw

Figure 13-26 Customer with accounts and new links

- Click **Contact Us** and the Tiles page is displayed (Figure 13-27).



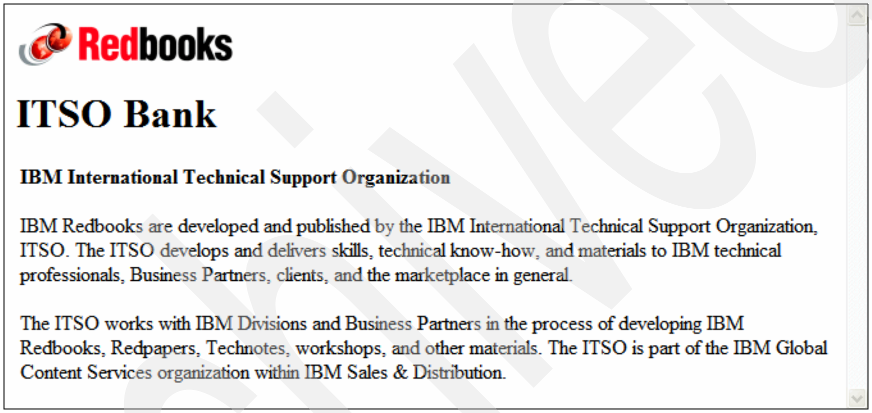
Redbooks

ITSO Bank

For more information about this sample application or related Redbook, please send an email to: redbooks@us.ibm.com

Figure 13-27 Tiles page: Contact Us

- Click **About Us** and the Tiles page is displayed (Figure 13-28).



Redbooks

ITSO Bank

IBM International Technical Support Organization

IBM Redbooks are developed and published by the IBM International Technical Support Organization, ITSO. The ITSO develops and delivers skills, technical know-how, and materials to IBM technical professionals, Business Partners, clients, and the marketplace in general.

The ITSO works with IBM Divisions and Business Partners in the process of developing IBM Redbooks, Redpapers, Technotes, workshops, and other materials. The ITSO is part of the IBM Global Content Services organization within IBM Sales & Distribution.

Figure 13-28 Tiles page: About Us

Importing the final sample application

In the previous sections we described how to build and run a Struts Web application. If you did not manage to build the application, you can import the project interchange file from:

```
C:\7501code\zInterchangeFiles\struts\RAD7Struts.zip
```

- Select **File** → **Import** → **Project Interchange** and click **Next**.
- In the Import wizard select **Other** → **Project Interchange** and click **Next**.
- Click **Browse** to locate the interchange ZIP file and click **Open**.
- Select both projects and click **Finish**.
- The `RAD7StrutsWeb` and `RAD7StrutsEAR` projects are imported.

- Open the Web Diagram in the `RAD7StrutsWeb` project (Figure 13-29). The diagram looks like Figure 13-3 on page 549, with the two Tiles action added and two more links from `cusomerListing.jsp` to the two Tiles actions.

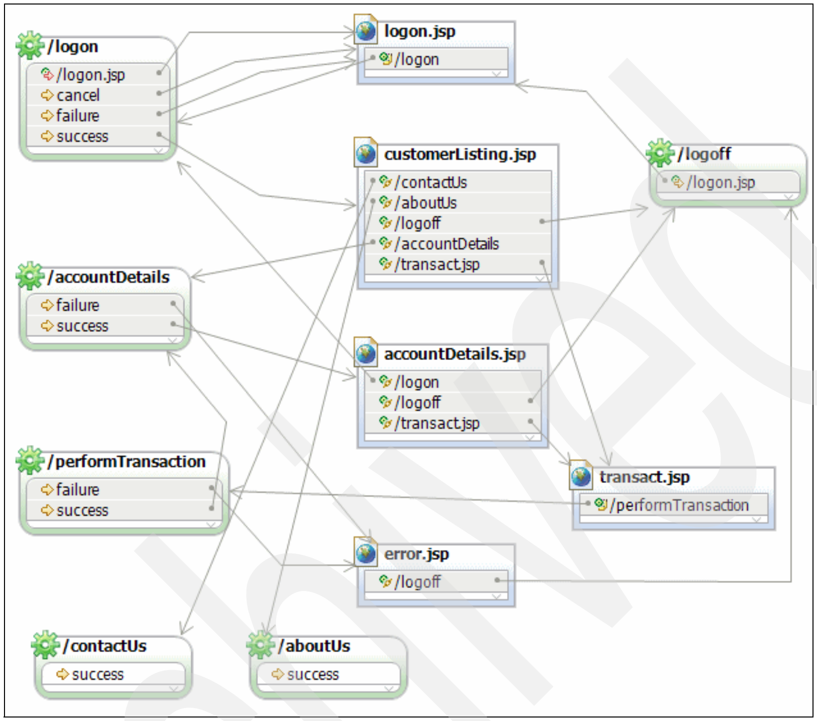


Figure 13-29 Complete Web Diagram with Tiles actions

More information

For more information about Struts and Tiles, consult these Web sites:

- Apache Struts home page:

  `http://struts.apache.org/`

- Apache Struts User Guide:

  `http://struts.apache.org/userGuide/introduction.html`

- Apache Tiles home page:

  `http://tiles.apache.org/`

- IBM developerWorks (search for Struts and Tiles):

  `http://www.ibm.com/developerworks/`

14

Develop Web applications using JSF and SDO

JavaServer Faces (JSF) is a framework that simplifies building user interfaces for Web applications. Service Data Objects (SDO) is a data programming architecture and API for the Java platform that unifies data programming across data source types.

This chapter introduces the features, benefits, and architecture of JSF and SDO. The focus of the chapter is to demonstrate the Rational Application Developer support and tooling for JSF. We also discuss the new feature Asynchronous JavaScript and XML (AJAX) for developing JSF applications. The chapter includes an example Web application using JSF, SDO, and AJAX.

The chapter is organized into the following sections:

- Introduction to JSF, SDO, and AJAX
- Developing a Web application using JSF and SDO
- Running the sample Web application
- Adding AJAX behavior to the sample Web application

Introduction to JSF, SDO, and AJAX

This section provides an introduction to JavaServer Faces (JSF), Service Data Objects (SDO), and Asynchronous JavaScript and XML (AJAX).

JavaServer Faces (JSF) overview

JavaServer Faces is a framework that simplifies building user interfaces for Web applications. The combination of the JSF technology and the tooling provided by IBM Rational Application Developer allows developers of differing skill levels the ability to achieve the promises of rapid Web application development.

This section provides an overview of the following aspects of JSF:

- JSF features and benefits
- JSF application architecture
- JSF features in Application Developer

> **Note:** Detailed information on the JSF specification can be found at:
>
> `http://java.sun.com/j2ee/javaserverfaces/download.html`

JSF features and benefits

The following list describes the key features and benefits of using JSF for Web application design and development:

- **Standards-based Web application framework**:
  - JSF is a standards-based Web application framework. JavaServer Faces technology is the result of the Java Community process JSR-127 and evolved from Struts. JSF addresses more of the model-view-controller pattern than Struts, in that it more strongly addresses the view or presentation layer though UI components, and addresses the model through managed beans. Although JSF is an emerging technology and is likely become a dominant standard, Struts is still widely used.
  - JSF is targeted at Web developers with little knowledge of Java and eliminates much of the hand coding involved in integrating Web applications with back-end systems.
- **Event driven architecture**: JSF provides server-side rich UI components that respond to client events.
- **User interface development**:
  - UI components are de-coupled from their rendering. This allows for other technologies such as WML to be used (for example, mobile devices).
  - JSF allows direct binding of user interface (UI) components to model data.

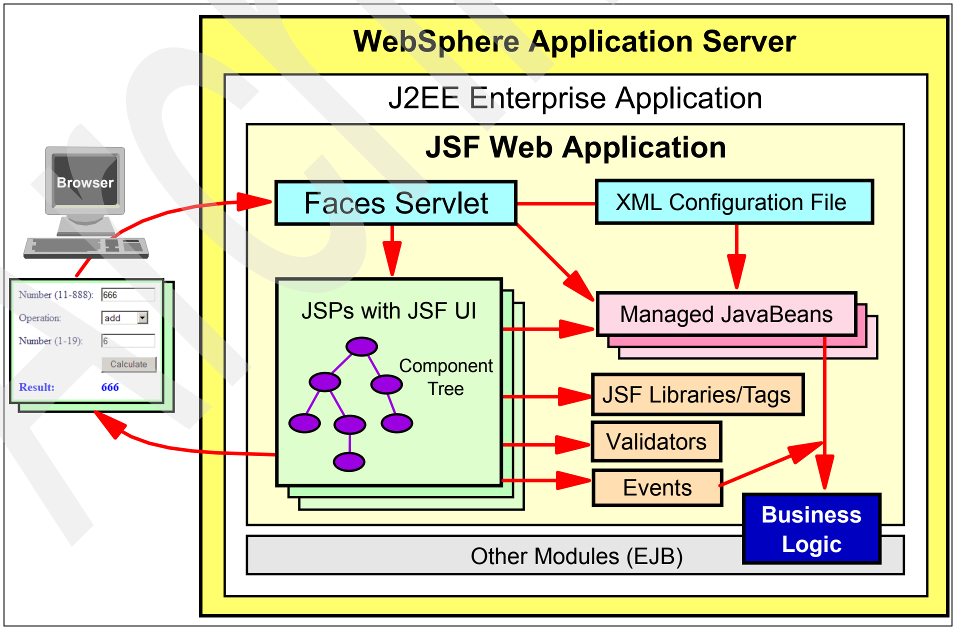
- – Developers can use extensive libraries of prebuilt UI components that provide both basic and advanced Web functionality.

- ► **Session and object management**: JSF manages designated model data objects by handling their initialization, persistence over the request cycle, and cleanup.
- ► **Validation and error feedback**: JSF allows direct binding of reusable validators to UI components. The framework also provides a queue mechanism to simplify error and message feedback to the application user. These messages can be associated with specific UI components.
- ► **Internationalization**: JSF provides tools for internationalizing Web applications, including supporting number, currency, time, and date formatting, and externalization of UI strings.

JSF application architecture

The JSF application architecture can be easily extended in a variety of ways to suit the requirements of your particular application. You can develop custom components, renderers, validators, and other JSF objects and register them with the JSF runtime.

The focus of this section is to highlight the JSF application architecture depicted in Figure 14-1.

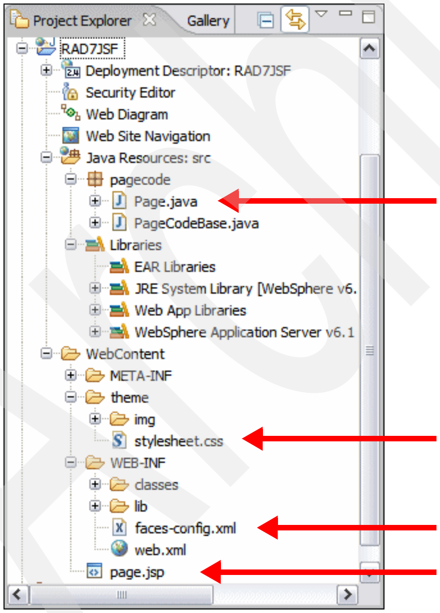


Number (11-888): 666
Operation: add
Number (1-19): 6
Calculate
Result: 666

Figure 14-1 JSF application architecture

- Faces JSP pages are built from JSF components, where each component is represented by a server-side class.
- Faces servlet: One servlet (`FacesServlet`) controls the execution flow.
- Configuration file: An XML file (`faces-config.xml`) that contains the navigation rules between the JSPs, validators, and managed beans.
- Tag libraries: The JSF components are implemented in tag libraries.
- Validators: Java classes to validate the content of JSF components, for example, to validate user input.
- Managed beans: JavaBeans defined in the configuration file to hold the data from JSF components. Managed beans represent the data model and are passed between business logic and user interface. JSF moves the data between managed beans and user interface components.
- Events: Java code executed in the server for events (for example, a push button). Event handling is used to pass managed beans to business logic.

Figure 14-2 represents the structure of a simple JSF application created in Application Developer.



JSF page server-side code

JSF stylesheet

JSF configuration file

Sample JSF Page

Figure 14-2 JSF application structure within Application Developer

JSF features in Application Developer

Application Developer includes a wide range of features for building highly functional Web applications. Application Developer includes full support to make drag-and-drop Web application development a reality.

Application Developer includes the following support and tooling for JSF Web application development:

- Visual page layout of JSF components using Page Designer
- Web Diagram Editor for defining the flow of a JSF application
- Built-in Component Property editor
- Built-in tools to simplify and automate event handling
- Page navigation defined declaratively
- Automatic code generation for data validation, formatting, and CRUD functions for data access
- Multiple faces configuration file support for different purposes, such as navigation and managed beans
- Relational database support
- EJB support
- Web services support
- Data abstraction objects for easy data connectivity (Service Data Objects)
- Data objects can be bound easily to user interface components
- Support for runtime page templates with Tiles
- Asynchronous JavaScript and XML (AJAX) support

Application Developer v7.0 has now additional rich and powerful components for JSF. These components include:

- **Menu Bar:** Displays a (hierarchical) menu bar of buttons and/or hyperlinks.
- **Panel - Dialog:** Creates a block panel that behaves like a modal or modeless dialog box. The panel floats above the main page, displays an appropriate title bar, can be moved, and normally has a pair of **OK** and **Cancel** buttons.
- **Panel - Form Box:** Creates a block panel that contains a header area and one or more form items (label/field pairs).
- **Panel - Section:** Creates a block panel that has a header that can be used to expand/collapse the display of the panel's content.
- **Select - Color:** Displays a drop-down combo box from which the user chooses a color.

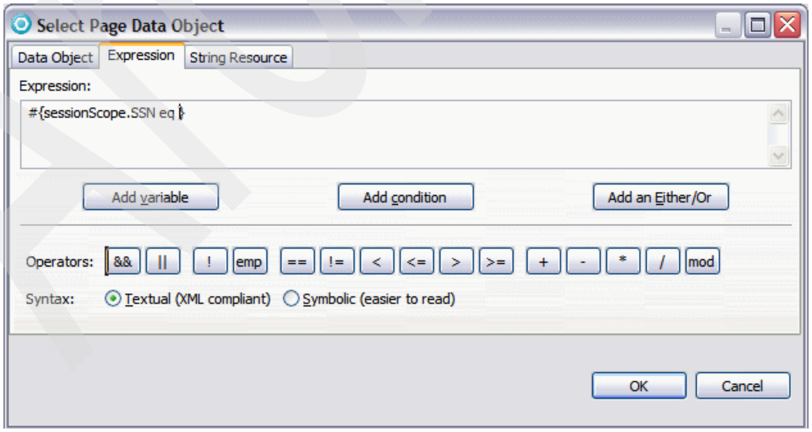
- **Select - Calendar:** Adds small calendar to the page. A user can select a date by clicking a day in the calendar.
- **Progress Bar:** Displays an animated progress bar.
- **Link - Request:** Generates an HTML link with a URL that can pass parameters and navigate to a page by passing a string to JSF navigation rules via a string returned from a JSF action or static action string.
- **Data Iterator:** Iterates over rows of model data allowing values from each row to be used in child components. For each row of data available, a set of child components are rendered. The iterator tag itself does not have any control over what is rendered, it just provides data to child components.

Application Developer v7.0 also provides additional improvements to existing JSF components. Some of these improvements are as follows:

- Command buttons can now have icons.
- A Data Table component can now have sortable columns.
- Data Table component rows can now have single select mode using the radio buttons.
- A JSF panel content can now be updated using AJAX.

In addition to new and enriched JSF components, Application Developer v7.0 introduces new tools to provide much more development efficiency and minimize development time. These new tools for JSF are:

- **VBL Expression Builder:** Attributes of JSF components can be specified by expressions. These expressions can now be created visually using the builder (Figure 14-3).



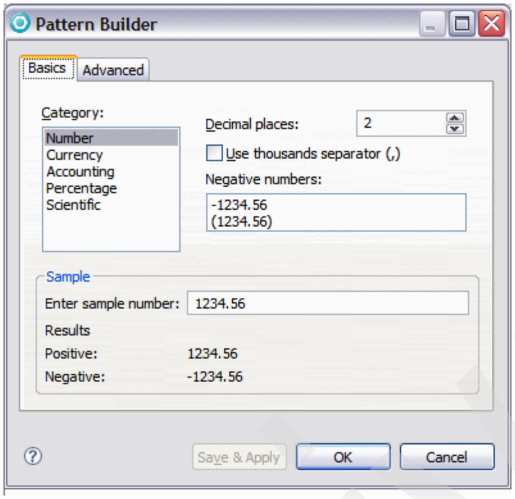
Expression:

```
#{sessionScope.SSN eq }
```

Syntax: ● Textual (XML compliant) ○ Symbolic (easier to read)

Figure 14-3 VBL Expression Builder

- **Pattern Builder:** Some JSF components that have date and number formats can use patterns. These patterns can now be created visually using the Pattern Builder (Figure 14-4).



Category:
- Number
- Currency
- Accounting
- Percentage
- Scientific

Decimal places: 2

□ Use thousands separator (,)

Negative numbers:
-1234.56
(1234.56)

Sample

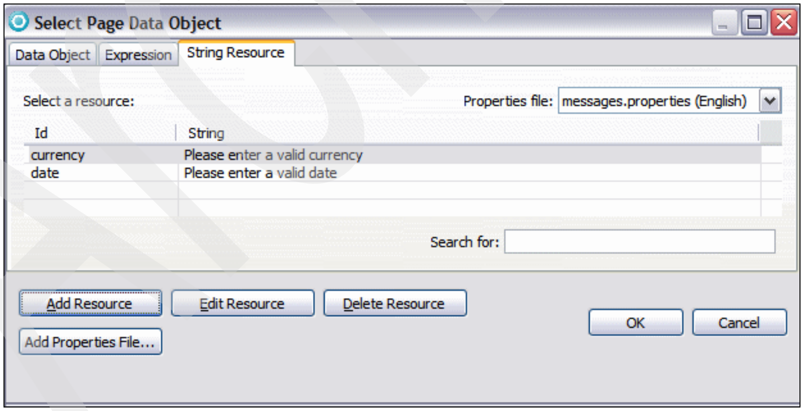
Enter sample number: 1234.56

Results

Positive:	1234.56
Negative:	-1234.56

Figure 14-4 Pattern Builder

- **Resource bundle editor:** JSF applications support localization that requires externalizing all the strings used in Faces JSP files into resource bundles. Resource bundle editor visually works with these resource bundle files so that the user can easily externalize all the strings (Figure 14-5).



Select a resource:

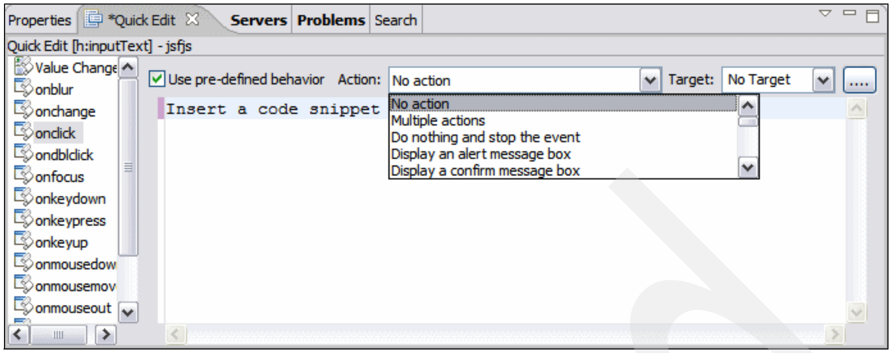
Properties file: messages.properties (English)

Id	String
currency	Please enter a valid currency
date	Please enter a valid date

Search for:

Figure 14-5 Resource bundle editor

- **Improved Quick Edit view:** Quick Edit view lets you add short scripts to your HTML and JSP files. The view now supports a library of pre-defined JavaScript functions on JSF components (Figure 14-6).



Quick Edit [h:inputText] - jsfjs

☑ Use pre-defined behavior Action: No action Target: No Target

```
Insert a code snippet
```

Figure 14-6 Improved Quick Edit view

Service Data Objects (SDO)

Service Data Objects is a data programming architecture and API for the Java platform that unifies data programming across data source types. SDO provides robust support for common application patterns, and enables applications, tools, and frameworks to more easily query, view, bind, update, and introspect data.

The Java specification request is JSR 235 for Service Data Objects (SDO) and can be found at:

http://www.jcp.org/en/jsr/detail?id=235

Note: For more detailed information on JavaServer Faces and Service Data Objects, we recommend the following Redbooks publication:

- *WebSphere Studio 5.1.2 JavaServer Faces and Service Data Objects*, SG24-6361

Asynchronous JavaScript and XML (AJAX)

Asynchronous JavaScript and XML (AJAX) refers to a group of technologies that are used to develop interactive Web applications. By combining these technologies, Web pages appear more responsive, since small amounts of data are exchanged with the server and Web pages are not reloaded each time a user makes an input change.

In classic Web applications, users have to submit forms (HTTP requests) to exchange data with the server. After some processing phase, the server returns the whole page to the client.

AJAX moves this server-side logic to the client. Instead of loading the whole page, the client can refresh the parts of page. The users' HTTP requests turn simply into JavaScript calls to the AJAX engine that is loaded by the browser. The AJAX Web application model is shown in Figure 14-7.

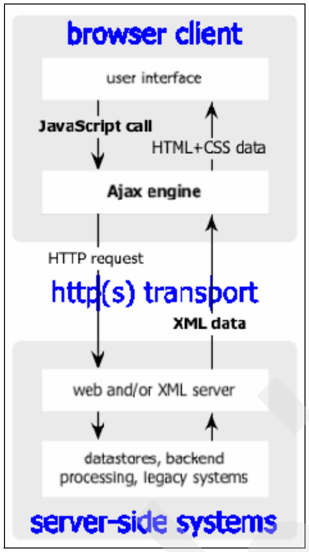


Figure 14-7 AJAX Web application model

AJAX is made up of the following technologies:

- HTML and CSS for presenting information.
- JavaScript for dynamically interacting with the information presented.
- XML, XSLT, and `HTMLHttpRequest` object to manipulate data asynchronously with the Web server.

Application Developer can now support AJAX enabled JSF components. With these components, user interactions with a page no longer have to submit/request entire pages, instead, pieces of the page can change as data is entered. Application Developer now also provides a JavaScript library that can initiate AJAX requests with the server by updating only parts of the page.

AJAX tags for JSF components are added as child tags of any of the panel tags. Their content defines an alternative content for the panel. When an action is triggered, a request is made back to the server for the new content to put in the panel. The server computes the new content, returns the content to the client where it is plugged into the panel, replacing its existing content.

Application Developer v7.0 AJAX components include these:

- **inputTypeAhead**: Extended input text component to support type ahead functionality.
- **ajaxRefreshRequest**: Defines how content from the same JSF page replaces the existing content of the parent tag. Content is received using an HTTP get request.
- **ajaxRefreshSubmit**: Defines how content from the same JSF page replaces the existing content of the parent tag. Content is received using an HTTP submit request.
- **ajaxExternalRequest**: Defines how content from a different page replaces the content of the parent tag. Content is retrieved using an HTTP get request against an arbitrary data source.

Preparing for the sample

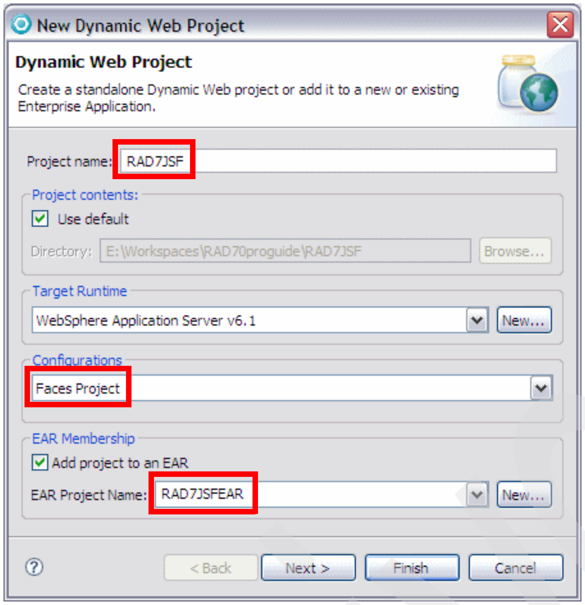
This section describes the tasks that must be completed prior to developing the JSF and SDO sample application.

> **Note:** A completed version of the Web application built using JSF and SDO can be found in the `c:\7501code\zInterchangeFiles\jsfsdo\RAD7JSF.zip` project interchange file. If you do not wish to create the sample yourself, but want to see it run, follow the procedures described in “Running the sample Web application” on page 647.

Creating a dynamic Web project

To create a dynamic Web project with support for JSF and SDO, do these steps:

- Open the Web perspective Project Explorer view.
- Select **File** → **New** → **Dynamic Web Project**.
- In the New Dynamic Web Project wizard enter the following (Figure 14-8), and click **Next**:
  - Name: **RAD7JSF**
  - Target Runtime: WebSphere Application Server v6.1 (default)
  - Configuration: Select **Faces Project**
  - EAR Project Name: **RAD7JSFEAR**



New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: RAD7JSF

Project contents:

☑ Use default

Directory: E:\Workspaces\RAD70proguide\RAD7JSF

Target Runtime: WebSphere Application Server v6.1

Configurations: Faces Project

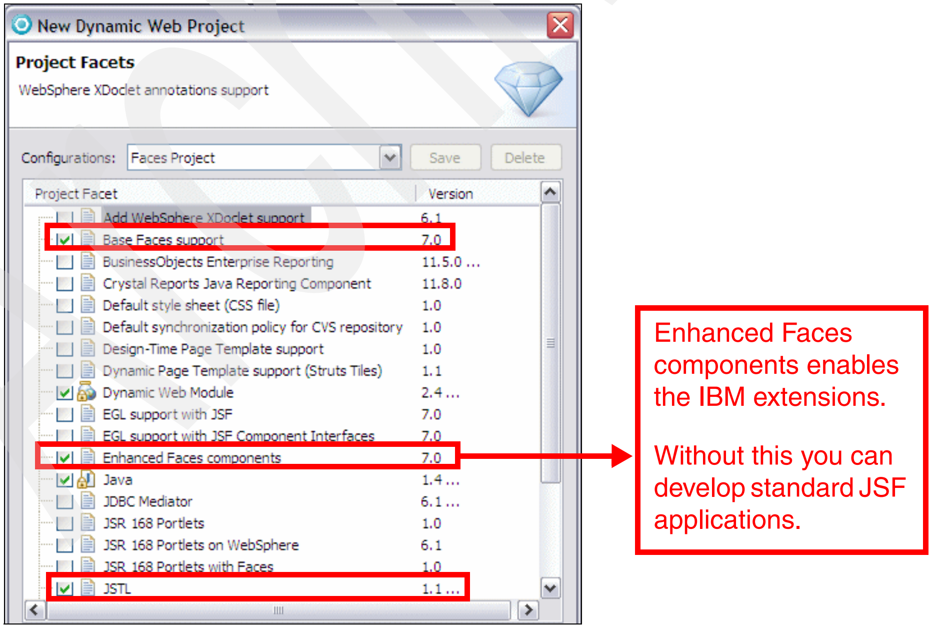
EAR Membership

☑ Add project to an EAR

EAR Project Name: RAD7JSFEAR

Figure 14-8 New Dynamic Web Project wizard

- In the **Project Facets** dialog, accept the default (Figure 14-9), and click **Next.**



New Dynamic Web Project

Project Facets

WebSphere XDoclet annotations support

Configurations: Faces Project

	Project Facet	Version
□	Add WebSphere XDoclet support	6.1
☑	Base Faces support	7.0
□	BusinessObjects Enterprise Reporting	11.5.0 ...
□	Crystal Reports Java Reporting Component	11.8.0
□	Default style sheet (CSS file)	1.0
□	Default synchronization policy for CVS repository	1.0
□	Design-Time Page Template support	1.0
□	Dynamic Page Template support (Struts Tiles)	1.1
☑	Dynamic Web Module	2.4 ...
□	EGL support with JSF	7.0
□	EGL support with JSF Component Interfaces	7.0
☑	Enhanced Faces components	7.0
☑	Java	1.4 ...
□	JDBC Mediator	6.1 ...
□	JSR 168 Portlets	1.0
□	JSR 168 Portlets on WebSphere	6.1
□	JSR 168 Portlets with Faces	1.0
☑	JSTL	1.1 ...

Enhanced Faces components enables the IBM extensions.

Without this you can develop standard JSF applications.

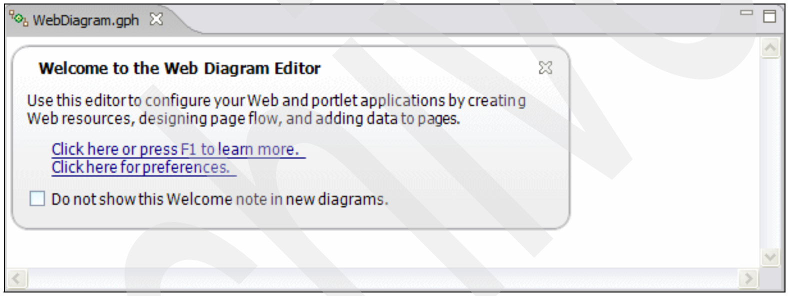
Figure 14-9 Project facets dialog for dynamic Web projects

Tip: Project facets features can also be added to a project later using the Project Facets panel of the project Properties dialog.

- In the Web Module dialog, accept the defaults and click **Finish**.

After creating the JSF application, the Web Diagram Editor automatically opens for creating Web resources, designing page flow, and adding data to pages (Figure 14-10).

The Web Diagram provides a visual way for you to create and manage the flow of a Web application. The Web Diagram Editor helps you visualize and structure the user interface of a Web application. You can create Web pages, add actions and mappings (such as inputs and outputs), add page data, and so on. In many cases, these items can be created for you automatically. Consider the Web Diagram as a way to create the logical flow and the basic user interface of a Web application.



Welcome to the Web Diagram Editor

Use this editor to configure your Web and portlet applications by creating Web resources, designing page flow, and adding data to pages.

Click here or press F1 to learn more.
Click here for preferences.

□ Do not show this Welcome note in new diagrams.

Figure 14-10 Web Diagram Editor

Close the Web Diagram for now.

Setting up the sample database

To use SDO components, we require a relational database. This section provides instructions for deploying the ITSOBANK sample database and populating the database with sample data. For simplicity we use the built-in Derby database.

Follow the instructions in “Setting up the ITSOBANK database” on page 1312 to create and load the sample ITSOBANK database.

Create a database connection

Follow the instructions in “Creating a database connection” on page 358 to create the **ITSOBANK** connection (if you do not have the connection already defined).

Configuring the data source

There are a couple of methods that can be used to configure the data source, including using the WebSphere Administrative Console or using the WebSphere Enhanced EAR, which stores the configuration in the deployment descriptor and is deployed with the application.

While developing JSF and SDO Web applications with Application Developer v7.0, the data source is created automatically when you add an SDO Relational Record or SDO Relational Record List to a Faces JSP file. The data source configuration is added to the Deployment tab of the EAR deployment descriptor.

You can also use the data source configured in the server, as described in “Configuring the data source in WebSphere Application Server” on page 1313.

Developing a Web application using JSF and SDO

Next we describe how to develop the sample Web application using JSF and SDO.

This section includes the following tasks:

- Creating a dynamic page template
- Views for editing page template files
- Customizing the page template
- Create Faces JSP pages using the Web Diagram Editor

Creating a dynamic page template

A page template contains common areas that you want to appear on all pages, and content areas that are unique on the page. They are used to provide a common look and feel for a Web project. You can create two kinds of Web page templates, design-time templates and dynamic page templates:

- Pages that you create using a **design-time template** are created immediately. Changes that you make to this template are immediately reflected in the pages that are based on the template.
- The **dynamic page template** is a new feature of Application Developer v7.0. Dynamic page templates use Struts-Tiles technology to generate pages on

the Web server at run-time. You can insert JSP fragments and configure the content areas of your template when you edit the template in Page Designer. In our sample Web application we use a dynamic page template.

To create a dynamic page template containing JSF components, do these steps:

► In Project Explorer, expand **RAD7JSF**.

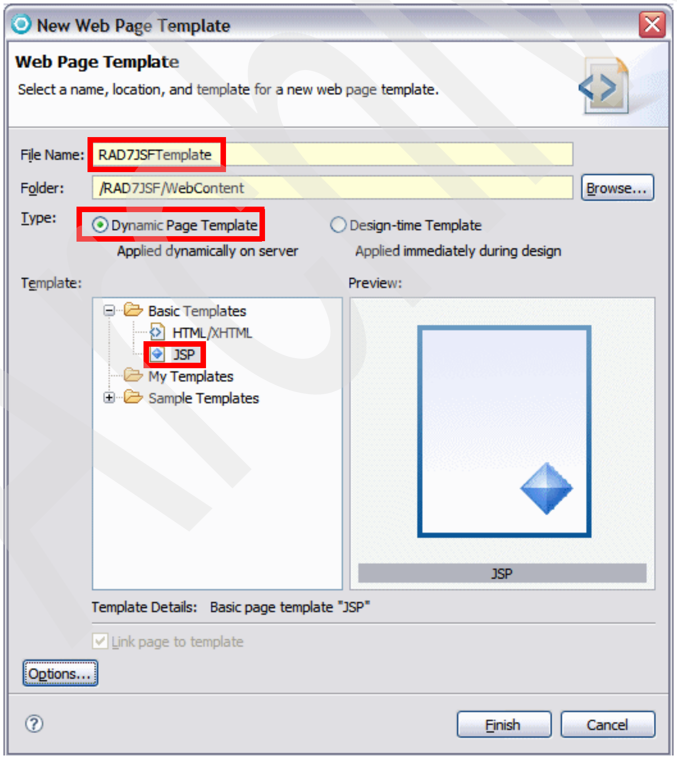
► Right-click the **WebContent** folder, and select **New** → **Web Page Template**.

► In the New Web Page Template wizard, enter the following items (Figure 14-11), and click **Finish**:

  – File name: `RAD7JSFTemplate`
  – Folder: `/RAD7JSF/WebContent`
  – Type: Select **Dynamic Page Template**
  – Template: Select **Basic Templates** → **JSP**.

  You can also select an existing template from the list of pre-defined sample templates. You can preview most of the templates in the preview box.

  Note that templates that you have previously created automatically appear under **My Templates**.



New Web Page Template

Web Page Template

Select a name, location, and template for a new web page template.

File Name: RAD7JSFTemplate

Folder: /RAD7JSF/WebContent

Type: ● Dynamic Page Template ○ Design-time Template

Applied dynamically on server — Applied immediately during design

Template:

- Basic Templates
  - HTML/XHTML
  - JSP
- My Templates
- Sample Templates

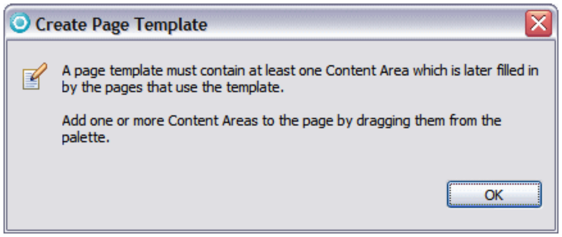
Preview:

Template Details: Basic page template "JSP"

☑ Link page to template

Figure 14-11 New Dynamic Page Template wizard design

- When prompted with the content area message (Figure 14-12), click **OK**. We add a content area as part of our page template customization in later steps.



A page template must contain at least one Content Area which is later filled in by the pages that use the template.

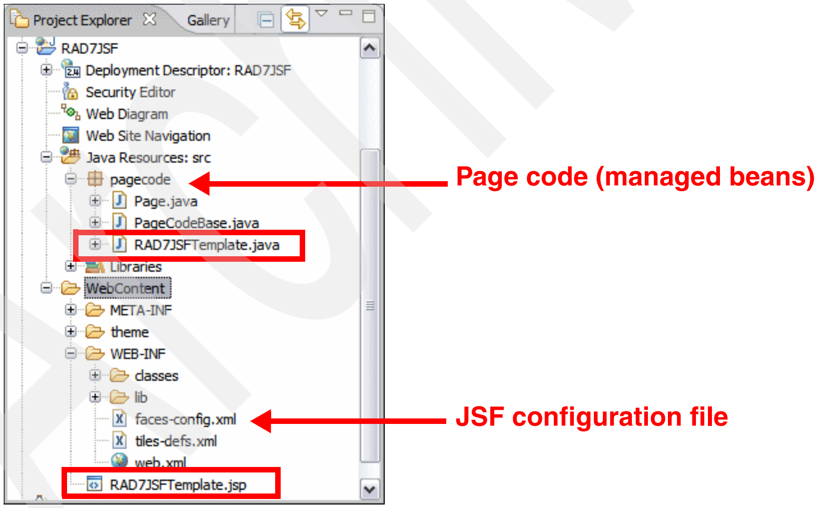
Add one or more Content Areas to the page by dragging them from the palette.

Figure 14-12 Content area warning

Review of files generated for page template

Now that we have created the page template, we would like to highlight some of the files that have been generated.

In the Project Explorer view (Figure 14-13), there are some files added to the Java Resources folder, which contains the generated managed bean classes. The `PageCodeBase.java` class file is a super class for all the generated managed bean classes. A `RAD7JSFTemplate.java` class is created as the managed bean for the template page. Each JSF page has its own managed bean class generated.



Page code (managed beans)

JSF configuration file

Figure 14-13 JSF page template files

Views for editing page template files

There are many views and tools that can be used when editing page templates. This section highlights some views that we found helpful for editing a page template.

> **Note:** This section is not required for the working example. To continue with the working example, skip to “Customizing the page template” on page 607.

Page Designer

The Page Designer is the main editor pane and is composed of three tabs: Design, Source, and Preview:

- **Design tab** (Figure 14-14): The Design tab is the main GUI editor that allows you to select the components on the page in a graphical manner. It also gives you a good representation of the resulting page. Also, you notice in the top right of Figure 14-14 that there are some drop-down options; these options allow you to select the layout mode, select an existing component on the page, and select font options for presentation, respectively. If you right-click the Design editor, the context menu allows you to be able to insert a variety of components, as well as edit embedded documents.

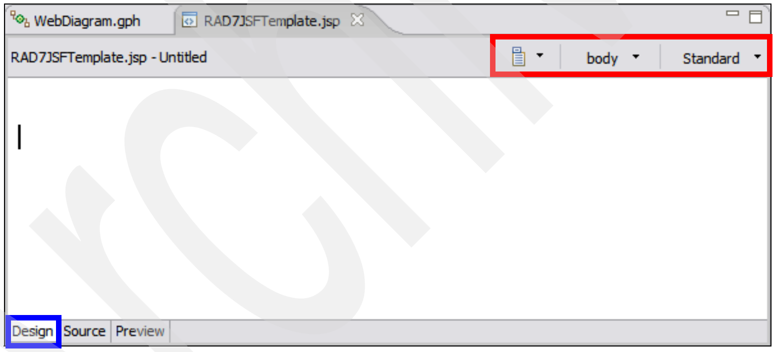
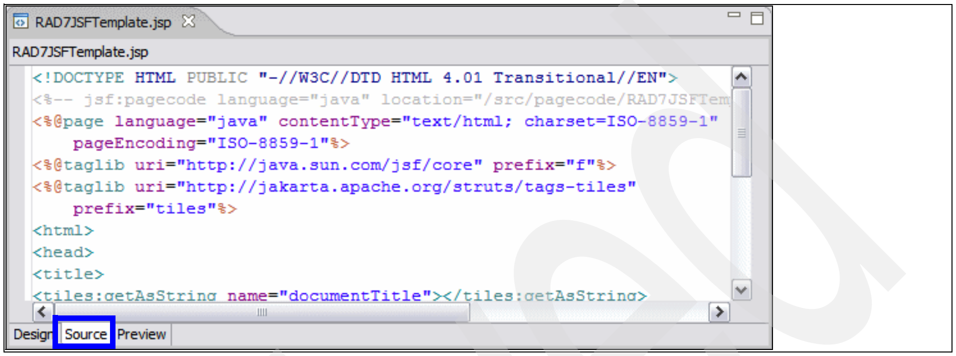


Figure 14-14 Design tab of Page Template Editor

- **Source tab** (Figure 14-15): The Source tab allows you to view the page template source. This tab allows you to tweak the template by changing the source directly. If you right-click in the source editor, you have the option to clean up the document, format the source, and refactor components:
  - The **Cleanup** option allows you to change the casing of HTML tags and attributes. It also gives you the ability to clean up tags with missing attributes, place quotes around attributes, insert missing tags, and format the source.

- The **Format** option allows you to format the whole document based on your formatting preferences, or just format the active element. To set your preferences, you can right-click the source editor and select **Preferences**. The preferences dialog is similar to selecting **Window** → **Preferences**, however, only the relevant preferences are include in the dialog.



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<%-- jsf:pagecode language="java" location="/src/pagecode/RAD7JSFTem
<%@page language="java" contentType="text/html; charset=ISO-8859-1"
    pageEncoding="ISO-8859-1"%>
<%@taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@taglib uri="http://jakarta.apache.org/struts/tags-tiles"
    prefix="tiles"%>
<html>
<head>
<title>
<tiles:getAsString name="documentTitle"></tiles:getAsString>
```

Figure 14-15 Source tab of the Page Template editor

- **Preview tab** (Figure 14-16): The Preview tab allows you to preview the resulting JSF page template. It presents the page template as it would appear in a browser. Notice in the top right corner that there are buttons similar to the navigation buttons in a browser. They allow you to go to the previous page, next page, or reload the page.

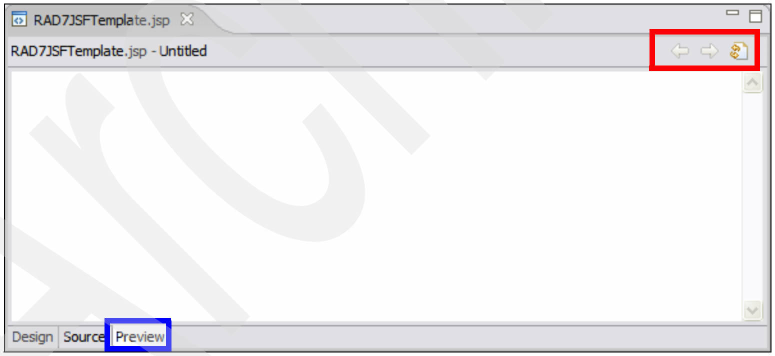


Figure 14-16 Preview tab for Page Template editor

Palette view

This view allows you to select palettes to work with. Examples of palettes in the Palette view can be seen in Figure 14-17. Palettes contain the reusable components that can be dragged on to the page to be added to a file.

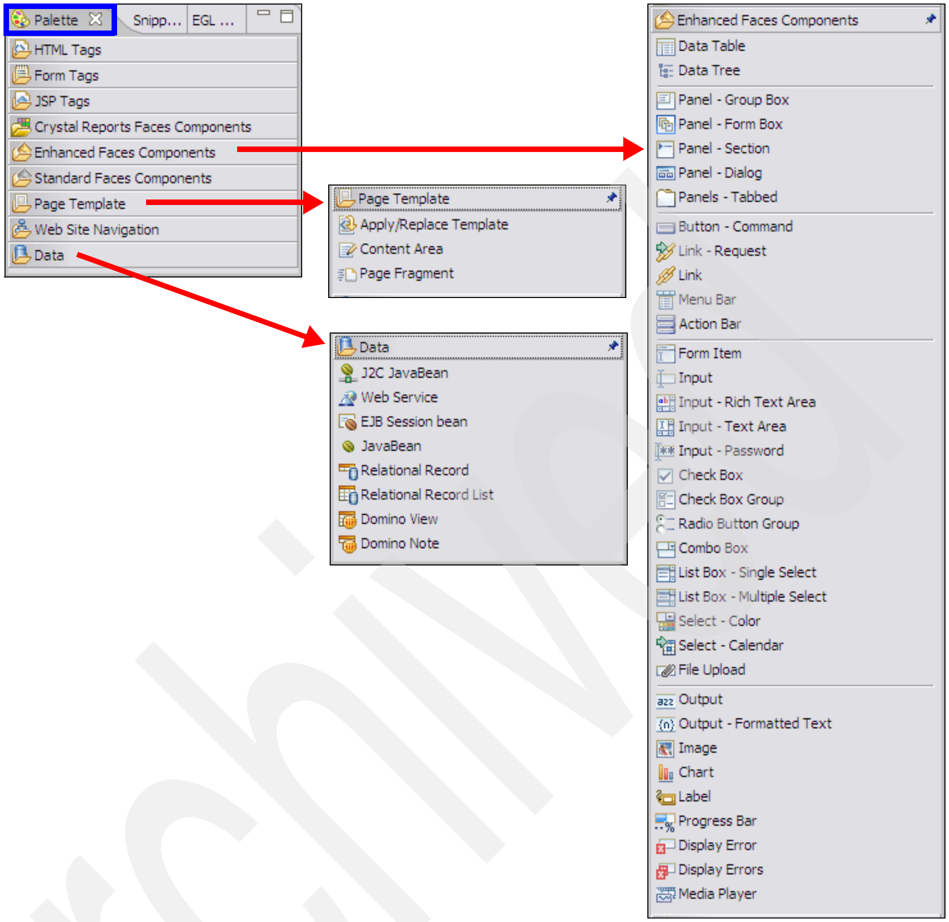


Figure 14-17 Palette view

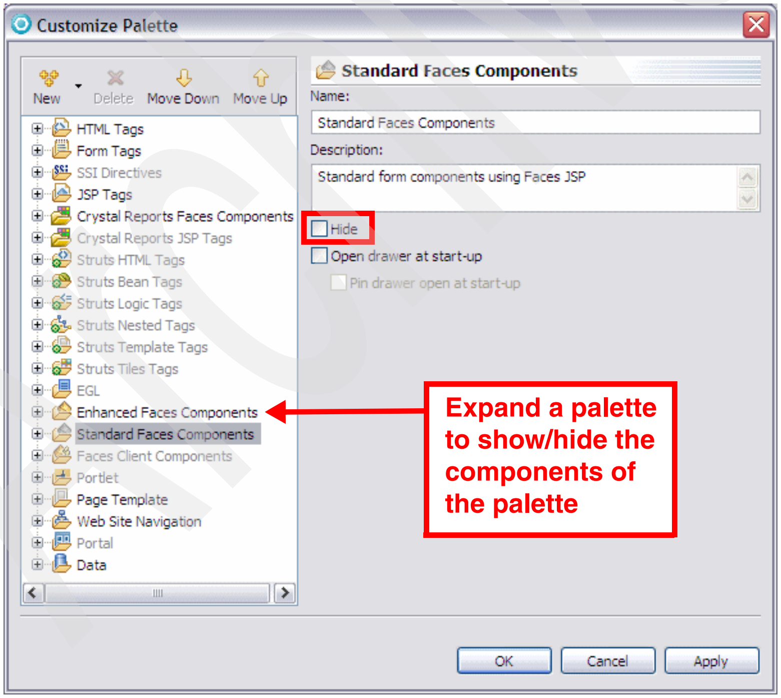
The palettes that are available for page template files include:

- **HTML Tags:** This palette contains common HTML tags such as images, lists, tables, links, and so on.
- **Form Tags:** This palette contains common Form tags such as form, submit button, reset button, radio button, text box, and so on.
- **JSP Tags:** This palette contains common JSP tags such as beans, expressions, scriptlets, declarations, include directives, as well as some logical tags such as if, when, otherwise, and so on.
- **Crystal Report Faces Components:** This palette contains Crystal Report Faces Components. You can drag these reusable Crystal Report Components to be used on the page template.

- **Enhanced Faces Components:** This palette contains the IBM extended Faces components and the standard Faces components.
- **Standard Faces Components:** This palette contains the standard Faces components. (Select **Customize** to show. By default, this Palette is hidden.)
- **Page Template:** This palette contains page Template tags such as content area and page fragment.
- **Web Site Navigation:** This palette contains IBM's site edit tags, which can be used to add site navigation to the page template.
- **Data:** This palette contains SDO and other data components such as JavaBean, domino view, relation record, relational record lists, and Web service.

The Palette view can be customized to hide and display palettes that you want to use. The palettes listed above are the default palettes presented when editing a page template with JSF and SDO components. Other palettes that can be included are Portlet, Struts, and Server Side Include (SSI) directives.

Right-click the Palette view and select **Customize** (Figure 14-18).



- HTML Tags
- Form Tags
- SSI Directives
- JSP Tags
- Crystal Reports Faces Components
- Crystal Reports JSP Tags
- Struts HTML Tags
- Struts Bean Tags
- Struts Logic Tags
- Struts Nested Tags
- Struts Template Tags
- Struts Tiles Tags
- EGL
- Enhanced Faces Components
- Standard Faces Components
- Faces Client Components
- Portlet
- Page Template
- Web Site Navigation
- Portal
- Data

Standard Faces Components

Name: Standard Faces Components

Description: Standard form components using Faces JSP

□ Hide

□ Open drawer at start-up

□ Pin drawer open at start-up

Expand a palette to show/hide the components of the palette

Figure 14-18 Customize Palette dialog

The Customize Palette dialog allows you to hide and show palettes, as well as reorganize the order in which they appear. Clear **Hide** for each palette that you want to display on the palette. Notice that in the dialog, the hidden palettes are a lighter shade of gray.

Some palettes contain more components that can fit on the screen and arrows allow you to scroll up and down. Also, you can pin open frequently used palettes to stop them from being closed when you open other palettes.

Properties view

The Properties view allows you to define the most common set of attributes of a selected component, and you can also see options for adding controls, actions, or other components. The Properties view shown in Figure 14-19 shows the properties of the body tag. Notice the highlighted button on the top right, which allows you to view all the attributes of the tag in tabular format.

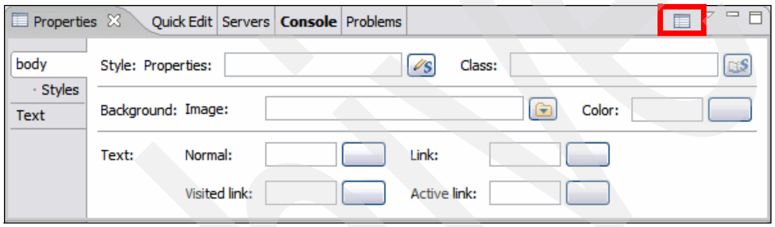


Figure 14-19 Properties view

Page Data view

The Page Data view provides access to data objects and scripting variables (Figure 14-20).

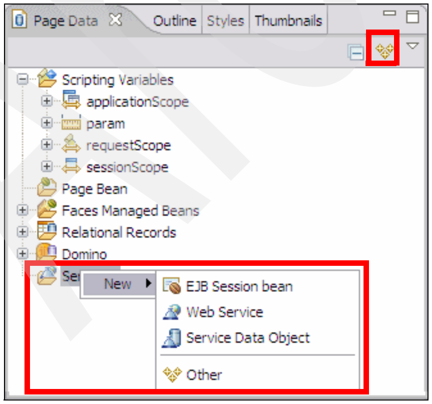


Figure 14-20 Page Data view

In the Page Data view, you can select managed beans to be added to the page or bind to existing page components. You can add, modify, or delete data components such as:

- Standard scripting variables:
  - applicationScope
  - param
  - requestScope
  - responseScope
- Other data objects:
  - JavaBeans
  - EJB session beans
  - Web services
  - Relational records and relational record lists
  - Portlet data objects
  - Struts form beans
  - Domino® Notes and Domino Views
  - J2C (J2EE Connector) JavaBeans

You can add these data objects using the highlighted **New** button on the Page Data toolbar or by right-clicking on the view. An appropriate wizard opens, depending on the type of object selected.

Customizing the page template

Now that you have created a page template, it is likely that you want to customize the page. This section demonstrates how to make the following common customizations to a page template:

- Customize the logo image and title of the page template.
- Customize the style of the page template.
- Add the content area to the page template.

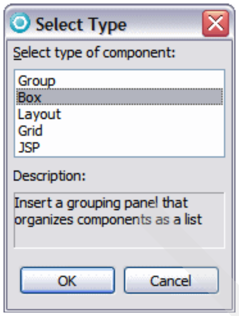
Customize the logo image and title of the page template

To customize the page template to include the ITSO logo image and ITSO RedBank title, do these steps:

- Import the `itso_logo.gif` image into the `theme` folder:
  - Expand **RAD7JSF** → **WebContent** → **theme**, right-click **theme**, and select **Import**.
  - Select **File System** and click **Next**.
  - Browse to the `C:\7501code\webapps\images` folder, select **itso_logo.gif**, and click **Finish**.

> **Tip:** You can also navigate to the `itso_logo.gif` in Windows Explorer and drag it into the `theme` folder in the Project Explorer of Application Developer.

- Expand **RAD7JSF** → **WebContent**.
- Open the **RAD7JSFTemplate.jsp** if you closed the editor.
- Select the **Design** tab.
- From the Palettes view, expand the **Enhanced Faces Components**.
- Select the **Panel - Group Box** Panel - Group Box and click into the page.
- In the Select Type dialog, select **Box** and click **OK** (Figure 14-21). Box is a kind of grouping panel that organizes components as a list.



Insert a grouping panel that organizes components as a list

Figure 14-21 Select Type - Box

- You should now see a box on your page with the text `box1: Drag and Drop Items from the palette to this area to populate this region.`

  From the Enhanced Faces Components palette, select **Image** Image and click into the box. After adding the image, the page is shown in Figure 14-22.

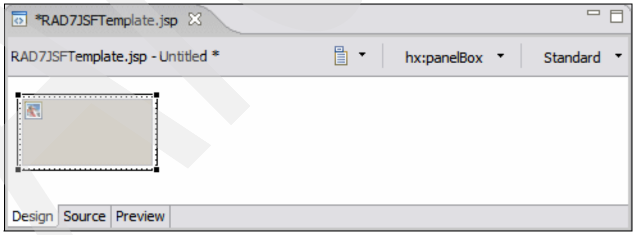
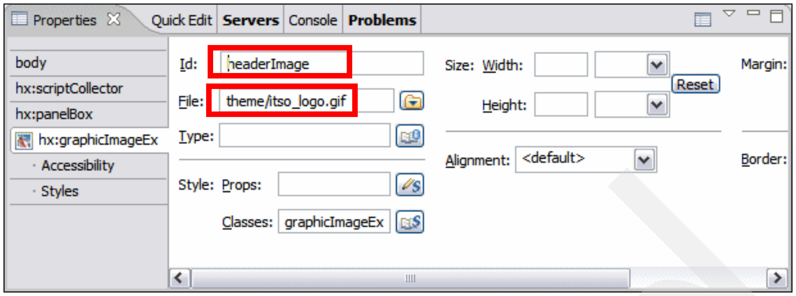


Figure 14-22 Page template after adding an image to Panel - Group box

- Select the image on the page and update the image values in the Properties view. Enter `headerImage` in the Id field and `theme/itso_logo.gif` in the File field (Figure 14-23).



Properties | Quick Edit | Servers | Console | Problems

- body
- hx:scriptCollector
- hx:panelBox
- hx:graphicImageEx
  - Accessibility
  - Styles

Id: headerImage

File: theme/itso_logo.gif

Type:

Style: Props:

Classes: graphicImageEx

Size: Width:

Height:

Reset

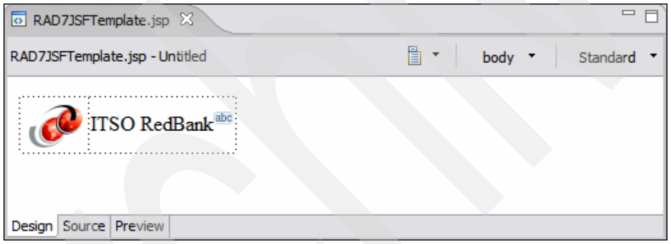
Alignment: <default>

Margin:

Border:

Figure 14-23 Properties for an image component

- From the Enhanced Faces Components palette, select **Output** Output and drag it under the image.
- Select the `outputText` and in the Properties view enter `ITSO RedBank` into the Value field.
- Select the `outputText` (`ITSO RedBank`) and move it to the right or left of the image (Figure 14-24). If the text displays on the left, move the image in front of the text.



RAD7JSFTemplate.jsp

RAD7JSFTemplate.jsp - Untitled | body | Standard

ITSO RedBank

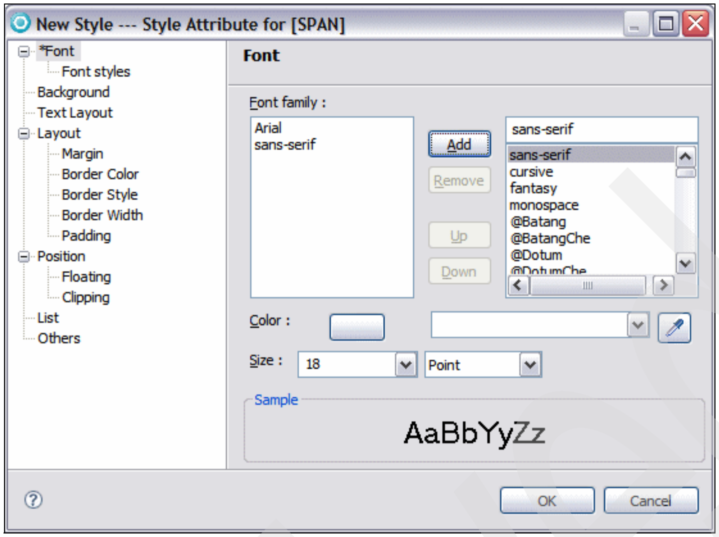
Design | Source | Preview

Figure 14-24 Page design after adding image and title to the template

Customize the style of the page template

To customize the style of the page template, do these steps:

- Select the Output box (`ITSO RedBank`) on the page.
- In the Properties view click the icon next to the **Style: Props:** field. The New style dialog is displayed (Figure 14-25).
- Change **Size** to `18`.
- Select **Arial** for the Font family and click **Add**.
- Select **sans-serif** for the Font family and click **Add**.
- Click **OK**.



New Style --- Style Attribute for [SPAN]

Font

Font family : Arial, sans-serif

Size : 18 Point

Figure 14-25 Style dialog used to configure the style of the selected components

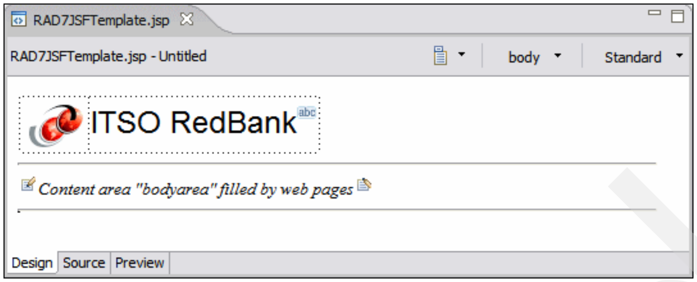
Add the content area to the page template

A *content area* is the section of the page that is filled in later by the pages that use the template. When you create a page from the template, the content area is editable, and the common area is fixed. Note that a page template must contain at least one content area.

To add the content area to the page template, do these steps (Figure 14-26):

- Right-click under the Output box (`ITSO RedBank`) and from the context menu, select **Insert** → **Horizontal Rule**.
- Expand Page Template in the Palette view.
- From the Page Template, select the **Content Area** Content Area and drag it under the horizontal rule. In the Insert Content Area for Page Template dialog accept the default name (`bodyarea`) and click **OK**.

  You should now see the text *Content area “bodyarea” filled by Web pages* on the page.
- Right-click under the content area and from the context menu, select **Insert** → **Horizontal Rule**.
- Save and close the page template file.



Content area "bodyarea" filled by web pages

Figure 14-26 Customized page template: RAD7JSFTemplate.jsp

Create Faces JSP pages using the Web Diagram Editor

This section demonstrates using the Web Diagram Editor to create Faces JSP pages, JSPs, Faces actions, and connections between pages and actions.

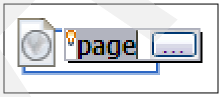
The sample application consists of the following three pages:

- Login page (`RAD7JSFLogin`): Validate the social security number (ssn). If it is valid it then display the customer details for the customer.
- Customer details page (`RAD7JSFCustomerDetails`): Display all the accounts of the customer and allow you to select an account to view the transactions.
- Account details page (`RAD7JSFAccountDetails`): Display the selected account details.

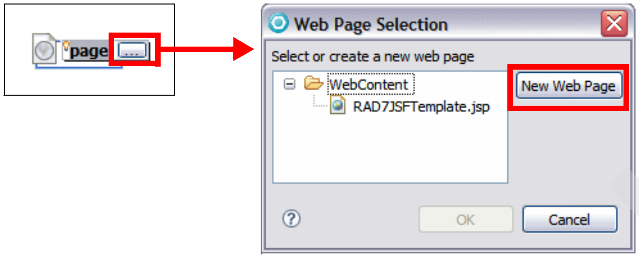
Create a Faces JSP page using the Web Diagram Editor

To create a Faces JSP page using the Web Diagram Editor, do these steps:

- In the Project Explorer, expand **RAD7JSF**.
- Open the **Web Diagram**, then close the Welcome box.
- Select **Web Page** Web Page from the Web Parts palette and drag it onto the page. The page appears as an icon, with the name (page) selected and a ... button.



- Click the ... button immediately to launch the page selection wizard (Figure 14-27). If you react too late, delete the `page.jsp` that is created.
- In the **Web Page Selection** dialog, click **New Web Page** to create a new page.



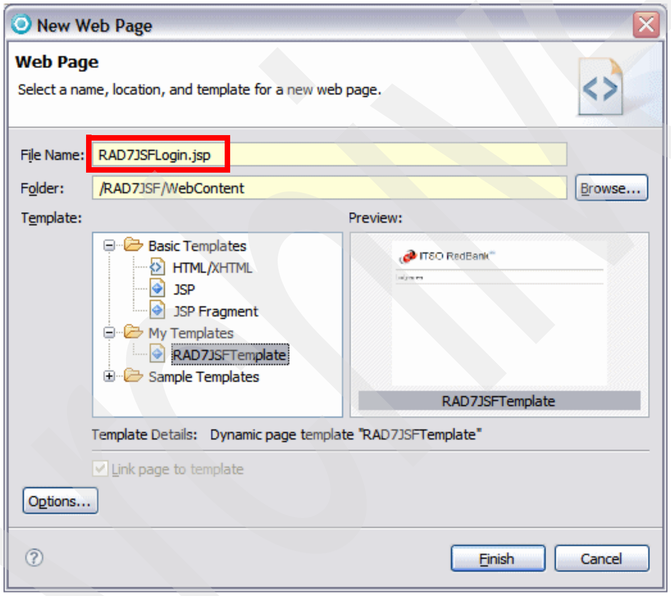
Web Page Selection

Select or create a new web page

WebContent
RAD7JSFTemplate.jsp

Figure 14-27 Create a new Web Page

- ► In the New Web Page dialog, enter `RAD7JSFLogin.jsp` as File Name.
- ► In the Template section, select **My Templates** → **RAD7JSFTemplate** (Figure 14-28). Click **Finish**.



New Web Page

Web Page

Select a name, location, and template for a new web page.

File Name: RAD7JSFLogin.jsp

Folder: /RAD7JSF/WebContent

Template:

Basic Templates
HTML/XHTML
JSP
JSP Fragment
My Templates
RAD7JSFTemplate
Sample Templates

Preview:

RAD7JSFTemplate

Template Details: Dynamic page template "RAD7JSFTemplate"

☑ Link page to template

Figure 14-28 New Web Page dialog

Note that is the template you created in the previous steps. You can see the preview of the selected template in the preview pane.

- ► In the Web Page Selection dialog, select the **RAD7JSFLogin.jsp** and click **Finish**.
- ► A *realized node* is shown in the diagram (Figure 14-29).

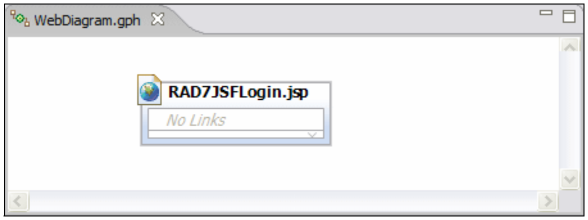


Figure 14-29 Web diagram with RAD7JSFLogin page

A node in a Web Diagram is *realized* when its underlying resource exists. Otherwise the node is *unrealized*. When you add a node to a Web Diagram, its underlying resource (for example, a Web page) is normally created automatically. In other words, you realize a node by default when you create it. Alternatively, you can add a node to a diagram without creating its underlying resource.

In a Web Diagram, realized and unrealized nodes are shown differently. Realized nodes have color and black title text. Unrealized nodes are gray and have gray title text. In our sample, the `RAD7JSFLogin` page is realized.

Tip: If you want to create the associated resource later, press **Shift+Enter** after you drag **Web Page** Web Page onto the page, otherwise the underlying resource is created automatically.

- Repeat the process to create Web pages for the other two JSF pages (Figure 14-30):
  - `RAD7JSFCustomerDetails.jsp`—Customer details and account overview
  - `RAD7JSFAccountDetails.jsp`—Account details with transactions

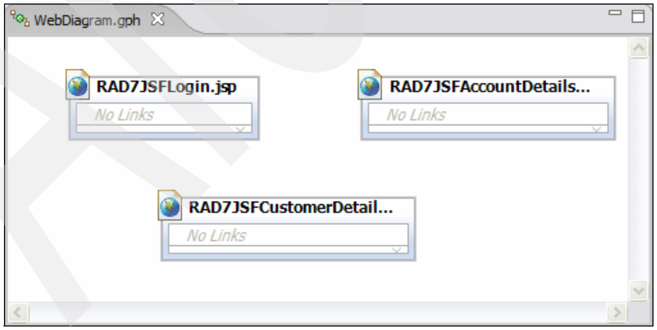


Figure 14-30 Realized Faces JSPs for the sample application

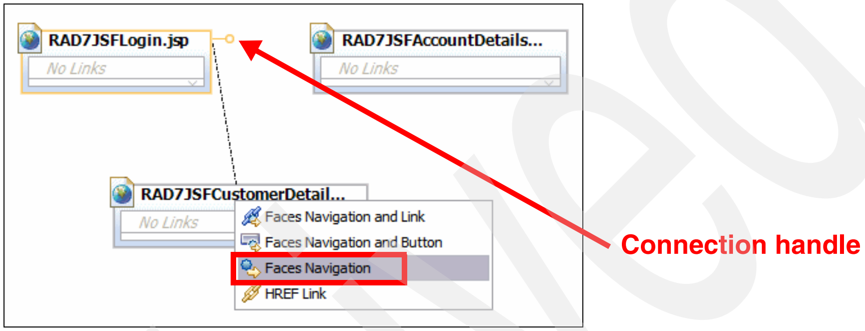
- Notice that all three pages are realized (black title). Save the Web Diagram.

Create connections between Faces JSP pages

Now that the pages have been created, we can create connections between the pages using the Web Diagram Editor.

To add connections between pages, do these steps:

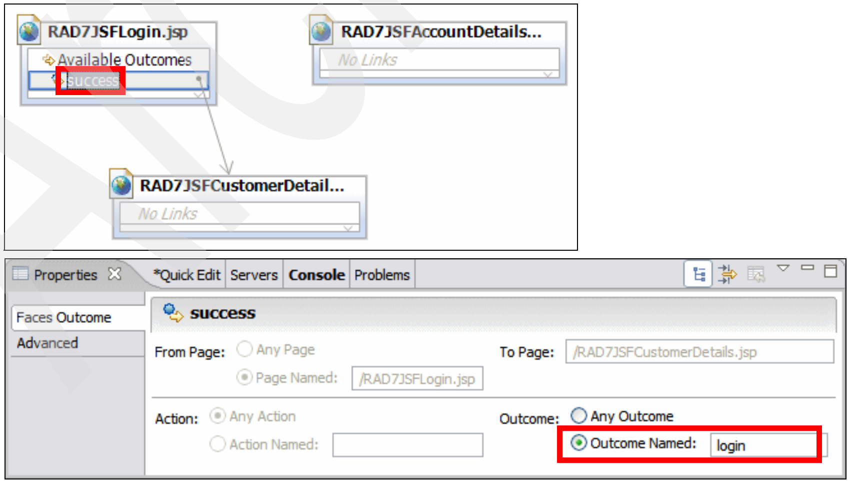
- Place the mouse over the **RAD7JSFLogin.jsp**. A connection handle is displayed near the top right corner of the node. Drag the connection handle to the **RAD7JSFCustomerDetails.jsp**. When prompted, select **Faces Navigation** as connection type (Figure 14-31).



Connection handle

Figure 14-31 Create a connection

- Change the name of the success action under Available Outcomes (Figure 14-32). Select the **success** action to highlight the text. Click on success again and overtype the name with **login**. You can also change the name in the Properties view Outcome Named field.



success

From Page: ○ Any Page ● Page Named: /RAD7JSFLogin.jsp

To Page: /RAD7JSFCustomerDetails.jsp

Action: ● Any Action ○ Action Named:

Outcome: ○ Any Outcome ● Outcome Named: login

Figure 14-32 Change the name of the action

- Save the Web Diagram.
- An arrow is drawn from the `RAD7JSFLogin` to the `RAD7JSFCustomerDetails`. The line is solid because the connection has been *realized* (added to the `faces-config.xml`).
- Review the modified Faces configuration file. Now we have a link from the `RAD7JSFLogin.jsp` to the `RAD7JSFCustomerDetails.jsp`. This link is activated when the outcome of an action on the `RAD7JSFLogin` page is `login`. To review how this link is realized in the JSF configuration, do these steps:
  - Expand **RAD7JSF** → **WebContent** → **WEB-INF** and open the **faces-config.xml** file.
  - Verify the navigation rule was added:

```
<navigation-rule>
      <from-view-id>/RAD7JSFLogin.jsp</from-view-id>
      <navigation-case>
         <from-outcome>login</from-outcome>
         <to-view-id>/RAD7JSFCustomerDetails.jsp</to-view-id>
      </navigation-case>
</navigation-rule>
```

Create Faces action

The Web Diagram Editor provides the means to create faces actions and connect Web pages to these actions.

To create a new Faces action, do these steps:

- From the Web Parts palette select **Faces Action (Managed Bean)** Faces Action (Managed Bean) and drag it onto the page.
- Click the ... button immediately to launch the Faces Action selection dialog. Click **New** to create a new Faces action (Figure 14-33).

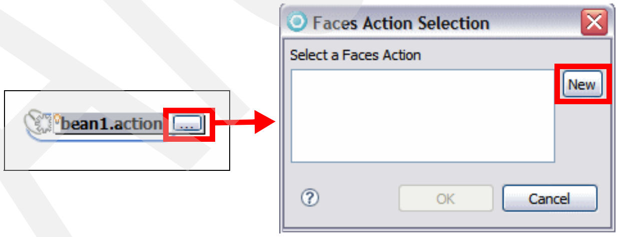
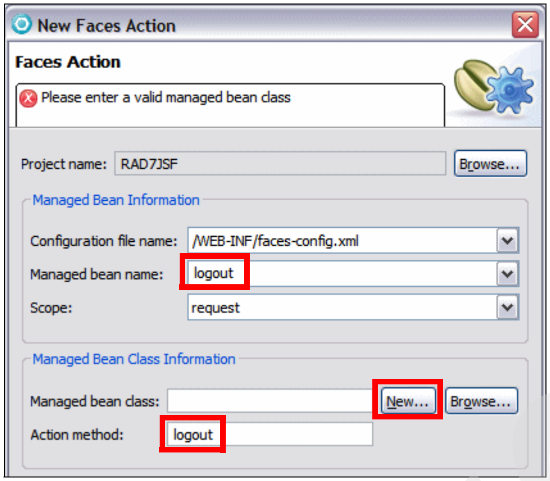


Figure 14-33 Create new Faces action

- In the New Faces Action dialog, enter **logout** in the Managed bean name and **logout** in the Action method field. Click **New** to create a new Managed bean class (Figure 14-34).


New Faces Action

Faces Action

Please enter a valid managed bean class

Project name: RAD7JSF

Managed Bean Information

Configuration file name: /WEB-INF/faces-config.xml

Managed bean name: logout

Scope: request

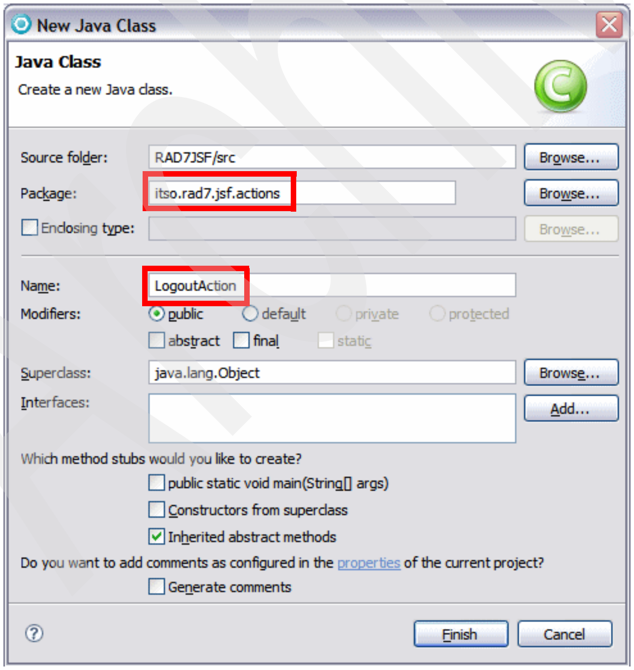
Managed Bean Class Information

Managed bean class:

Action method: logout

Figure 14-34   Create new Faces Action

- In the New Managed bean class dialog, enter **itso.rad7.jsf.actions** in the Package field and **LogoutAction** in the Name field (Figure 14-35). Click **Finish**.


New Java Class

Java Class

Create a new Java class.

Source folder: RAD7JSF/src

Package: itso.rad7.jsf.actions

□ Enclosing type:

Name: LogoutAction

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

□ Generate comments

Figure 14-35   Create new Managed Bean class

- In the New Faces Action dialog, click **Finish**.
- In the Faces Action Selection dialog, select **logout** → **logout** and click **OK** (Figure 14-36).

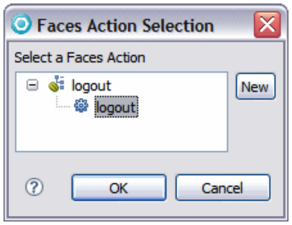


Figure 14-36 Faces Action Selection dialog

- The new action class (`LogoutAction`) is displayed in the editor. Modify the source code to match Example 14-1.

> **Note:** The Java code for the `LogoutAction` class can be copied from the file `c:\7501code\jsfsdo\Logout.jpage`.

Example 14-1 Completed LogoutAction action

```
package itso.rad7.jsf.actions;

import java.util.Map;
import javax.faces.context.FacesContext;

public class LogoutAction {

   protected FacesContext facesContext;
   protected Map sessionScope;
   private static final String CUSTOMERSSN_KEY = "SSN";
   private static final String SESSION_SCOPE = "#{sessionScope}";
   private static final String OUTCOME_LOGOUT = "logout";

   public String logout() {
      facesContext = FacesContext.getCurrentInstance();
      sessionScope = (Map) facesContext.getApplication().createValueBinding(
            SESSION_SCOPE).getValue(facesContext);
      if (sessionScope.containsKey(CUSTOMERSSN_KEY)) {
         sessionScope.remove(CUSTOMERSSN_KEY);
      }
      return OUTCOME_LOGOUT;
   }
}
```

The Web Diagram should now look similar to Figure 14-37.

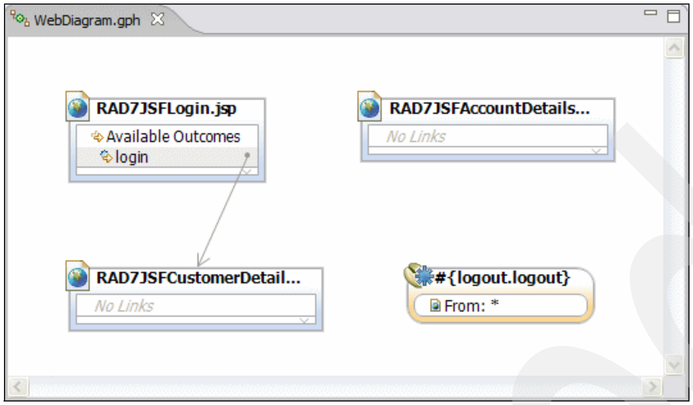


Figure 14-37 Logout bean in the Web Diagram

Add a connection for the action

To add a connection from the Action to a page, do these steps:

- Place the mouse over the **logout** action. A connection handle is displayed near the top right corner of the node. Drag the connection handle to the **RAD7JSFLogin.jsp**.
- Change the name of the success action to **logout** (overtype or use the Properties view).
- The new navigation rule is added to the `faces-config.xml` file:

```
<navigation-rule>
   <from-view-id>*</from-view-id>
   <navigation-case>
      <from-action>#{logout.logout}</from-action>
      <from-outcome>logout</from-outcome>
      <to-view-id>/RAD7JSFLogin.jsp</to-view-id>
   </navigation-case>
</navigation-rule>
```

Add remaining navigation rules

We now have an action bean that performs an action to return the user to the `RAD7JSFLogin` screen. We have to define the remaining navigation rules in the Web diagram:

- Create a connection from `RAD7CustomerDetails` to `RAD7AccountDetails` and name the action **accountDetails**.
- Create a connection from `RAD7AccountDetails` to `RAD7CustomerDetails` and name the action **customerDetails**.

Note that the navigation rules that point to the logout action do not require any outcome information.

- After adding the connections and rearranging the pages and action, the Web Diagram is shown in Figure 14-38.

> **Tip:** To change the shape of a connection, select the connection, point somewhere on the line and drag the mouse away to reshape the line for better visibility.

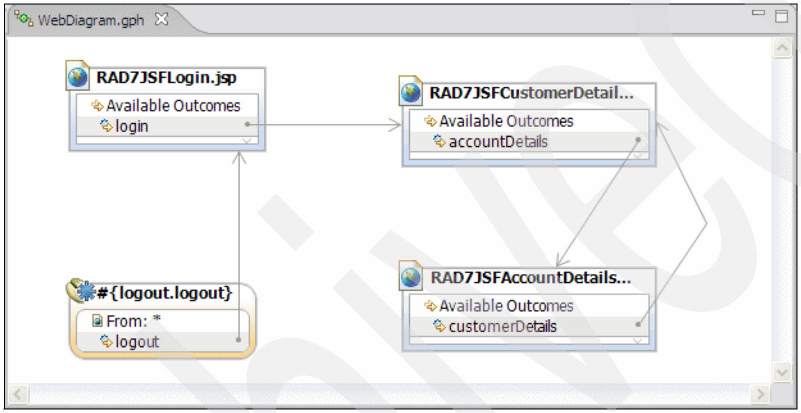
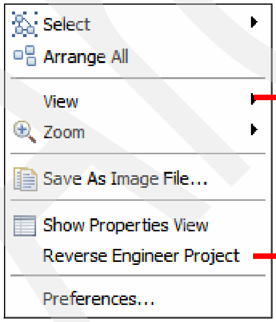


Figure 14-38 Completed Web diagram

Most functions in the Web Diagram Editor are available from the context-menu. To access any of these functions, right-click in the diagram and select an item from the menu (Figure 14-39).



Grid, Rulers

Populate diagram from existing Web pages

Figure 14-39 Web Diagram Editor toolbar

Editing the Faces JSP pages

This section demonstrates the JSF editing features in Application Developer by using the JSF pages created in “Create Faces JSP pages using the Web Diagram Editor” on page 611.

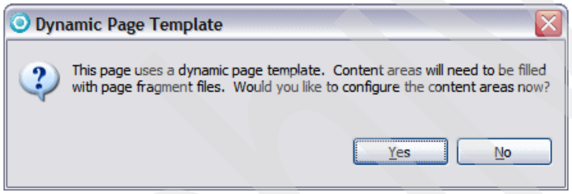
Editing the login page

We complete the login page in a few stages.

Add the UI components

To add the UI components to the `RAD7JSFLogin.jsp` page, do these steps:

- Open **RAD7JSFLogin.jsp** by double-clicking the file in the Web diagram or in the `WebContent` folder.
- In the Dynamic Page Template pop-up, click **Yes** to configure the content area (Figure 14-40).



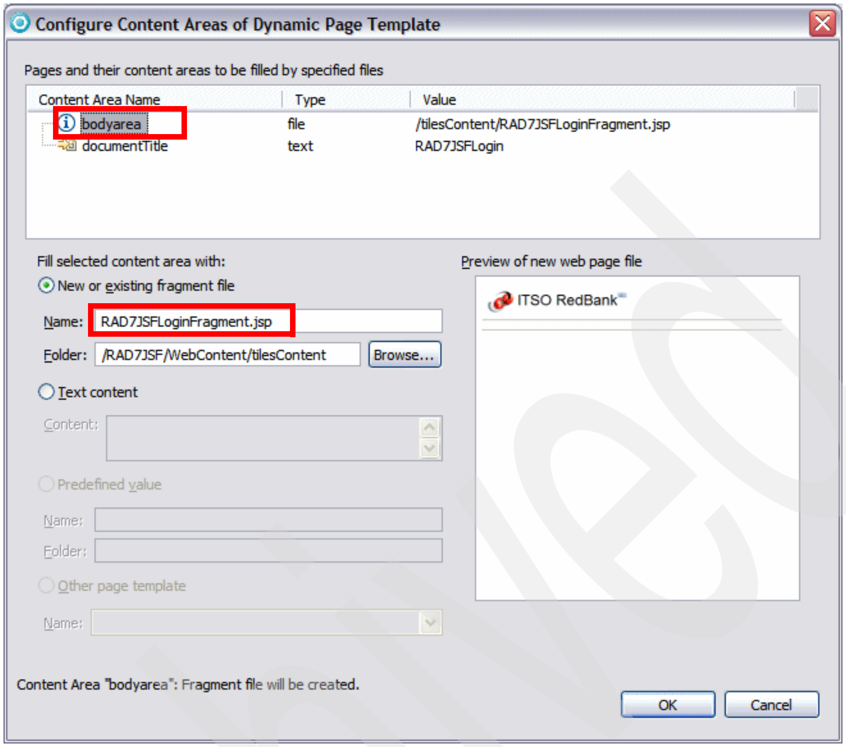
Dynamic Page Template

This page uses a dynamic page template. Content areas will need to be filled with page fragment files. Would you like to configure the content areas now?

Figure 14-40 Dynamic Page Template pop-up

- Dynamic page templates use Struts-Tiles technology to generate pages on the Web server at run-time. (Note that the pages do not actually contain the template content except in the generated files).

  You can insert JSP fragments into the template when you edit the template in Page Designer. That is, each content area is filled with a JSP fragment file that gives page specific content to the page. This enables you to easily keep page content and page layout separate.
- In the Configure Content Areas of Dynamic Page template dialog, do these steps (Figure 14-41) and click **OK**.
  - Highlight **bodyarea** from the content area list.
  - Select New or existing fragment file and enter `RAD7JSFLoginFragment.jsp` in the Name field.



Configure Content Areas of Dynamic Page Template

Pages and their content areas to be filled by specified files

Content Area Name	Type	Value
bodyarea	file	/tilesContent/RAD7JSFLoginFragment.jsp
documentTitle	text	RAD7JSFLogin

Fill selected content area with:

● New or existing fragment file

Name: RAD7JSFLoginFragment.jsp

Folder: /RAD7JSF/WebContent/tilesContent

○ Text content

Content:

○ Predefined value

Name:

Folder:

○ Other page template

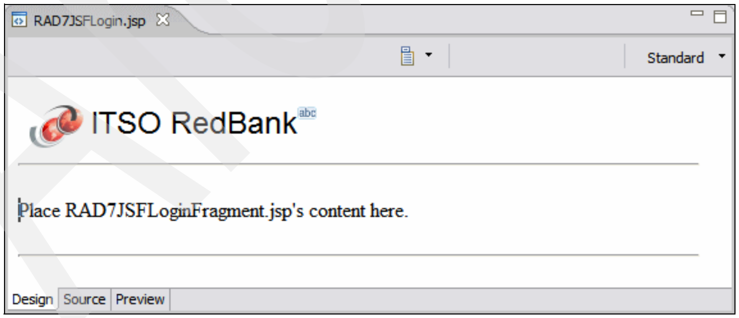
Name:

Preview of new web page file

Content Area "bodyarea": Fragment file will be created.

Figure 14-41 Configure the content area of the dynamic page template

- The fragment file (`RAD7JSFLoginFragment.jsp`) is created under `WebContent/tilesContent` and included in the actual `RAD7JSFLogin.jsp` (Figure 14-42). Now you can design the page.



Place RAD7JSFLoginFragment.jsp's content here.

Figure 14-42 Login page after configuring the content area

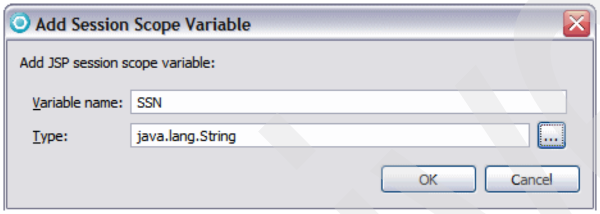
- Select the default text **Place RAD7JSFLoginFragment.jsp's content here** and **Delete**.

Add a variable for the SSN

When a page has a field where text is entered, the input can be stored. This is accomplished by creating a session scope variable to store the entered value and bind it with an input field.

To create a variable, do these steps:

- In the Page Data view, expand **Scripting Variables** (see Figure 14-20 on page 606).
- Right-click **sessionScope** and select **New** → **Session Scope Variable**.
- In the Add Session Scope Variable dialog, enter **SSN** for the Variable name, and `java.lang.String` for Type, and click **OK** (Figure 14-43).



Add JSP session scope variable:

Variable name: SSN

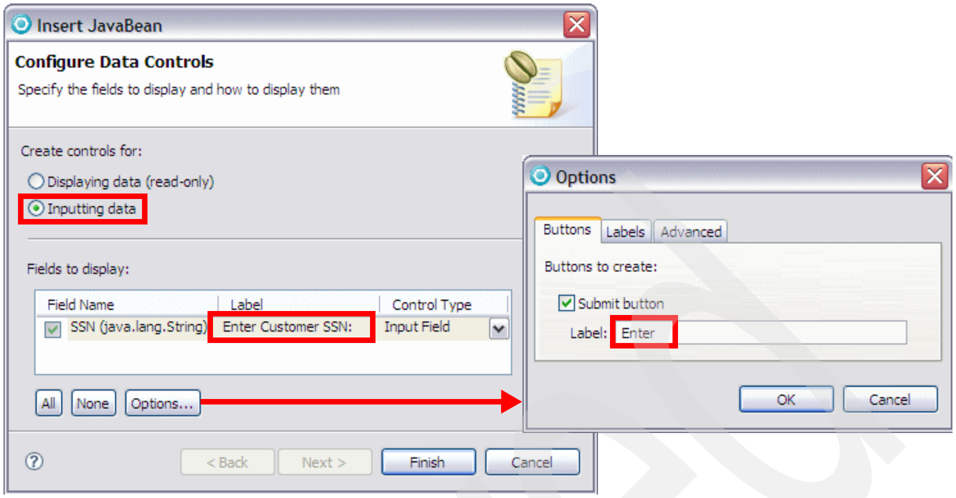
Type: java.lang.String

Figure 14-43 Add session scope variable

Create a form for the SSN

A variable can be dropped into the page to create a form with an input field and a push button:

- In the Page Data view, expand and select **Scripting Variables** → **sessionScope** → **SSN**. Drag SSN into the area under the Login text. A tooltip pop-up says *Drop here to insert new controls for "SSN"*.
- In the Insert JavaBean dialog (Figure 14-44):
  - Select **Inputting data**.
  - Overtype the generated Label with **Enter Customer SSN:**.
  - Leave the control type as **Input Field**.
  - Click **Options**, select **Submit button**, type **Enter** for the Label, and click **OK**.



Insert JavaBean

Configure Data Controls

Specify the fields to display and how to display them

Create controls for:

○ Displaying data (read-only)
● Inputting data

Fields to display:

	Field Name	Label	Control Type
☑	SSN (java.lang.String)	Enter Customer SSN:	Input Field

Options

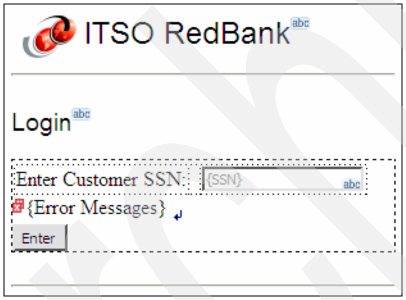
Buttons to create:

☑ Submit button

Label: Enter

Figure 14-44 Adding a variable to the JSP

- The resulting page is shown in Figure 14-45.



ITSO RedBank

Login

Enter Customer SSN: {SSN}

{Error Messages}

Figure 14-45 Design of the login page

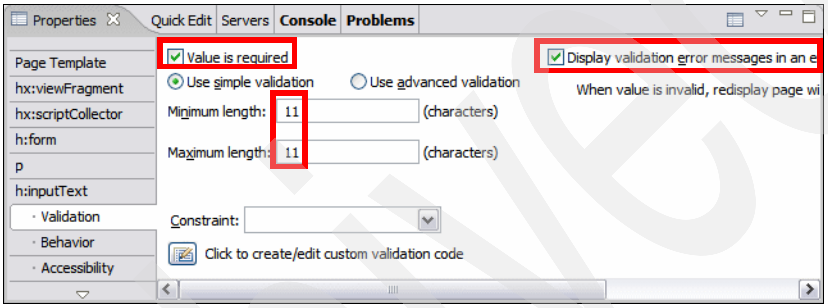
- Save the `RAD7JSFLogin.jsp`.
- Select the SSN input field and verify in the Properties view that the value is set to **#{sessionScope.SSN}**. This value indicates that the input field is bound to the session variable SSN. The session variable value will be displayed in the field and any change to the value is stored in the session variable.

Add simple validation

JSF offers a framework for performing validation on input fields such as required values, validation on the length of the input, and input check for all alphabetic or digits. You can also define your custom validations.

To add simple validation to an input field, do these steps:

- Select the Input component in the Design view.
- In the Properties view for the input component, select the **Validation** tab.
- Enter the following items in the Validation tab (Figure 14-46):
  – Select **Value is required**.
  – Select **Display validation error message in an error message control**. When you select this checkbox an error message component is added next to the Input box.
  – Enter **11** in the Minimum and Maximum length fields.



Properties ☒ | Quick Edit | Servers | Console | Problems

- Page Template
- hx:viewFragment
- hx:scriptCollector
- h:form
- p
- h:inputText
  - Validation
  - Behavior
  - Accessibility

☑ Value is required

● Use simple validation ○ Use advanced validation

Minimum length: 11 (characters)

Maximum length: 11 (characters)

Constraint:

Click to create/edit custom validation code

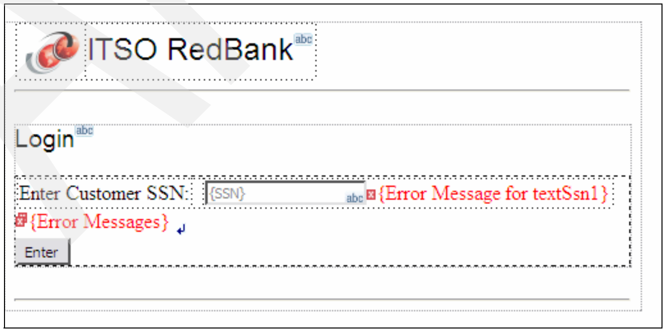
☑ Display validation error messages in an e

When value is invalid, redisplay page wi

Figure 14-46 Input box validation properties

- Select the **Error Message for ssn** component in the Design view.
- In the Properties view, for Style: Props enter **color: red**. (You could also click the Style icon and select **Red** for the Color field.
- Do the same for the **{Error Messages}** field (color: red).

After adding simple validation for the input component, the login page is shown in Figure 14-47.



ITSO RedBank

Login

Enter Customer SSN: {SSN} {Error Message for textSsn1}

{Error Messages}

Enter

Figure 14-47 Design view: Custom validation and error messages

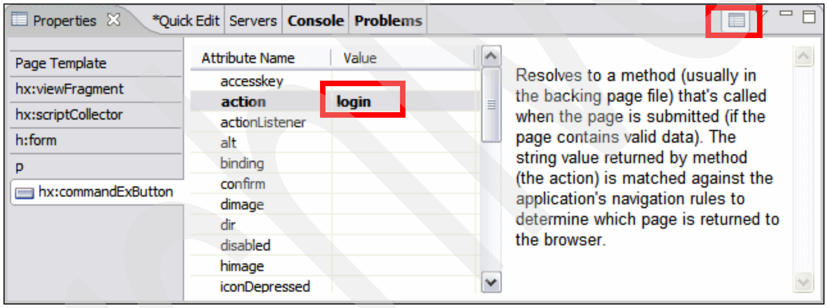
Note: If we do not add an error message field for an input field, messages are displayed automatically in the {Error Messages} field for the whole page.

Add static navigation to a page

Static navigation is called when the page is submitted. The string value returned by the method is matched against the application's navigation rules.

To add static navigation to page, do these steps:

- Select the **Enter** component (Command - Button).
- In the Properties view, click the **All Attributes** icon (▤) found in the top right. The Properties view is changed to show all the attributes of the selected component (Figure 14-48).
- Select the **action** attribute, and enter **login**. This is the outcome of the page when Enter is pressed, and as a result the navigation rule that we created is invoked.



Attribute Name	Value
accesskey	
action	login
actionListener	
alt	
binding	
confirm	
dimage	
dir	
disabled	
himage	
iconDepressed	

Resolves to a method (usually in the backing page file) that's called when the page is submitted (if the page contains valid data). The string value returned by method (the action) is matched against the application's navigation rules to determine which page is returned to the browser.

Figure 14-48 Properties view showing all attributes

- Save and close the `RAD7JSFLogin.jsp`.
- Open the `RAD7JSFLoginFragment.jsp` (in `WebContent\tilesContent`). In the Source tab you can see the generated source code for the input field and the Enter button:

```
<h:inputText id="textSsn1" value="#{sessionScope.SSN}"
   styleClass="inputText" required="true">
   <f:validateLength minimum="11" maximum="11"></f:validateLength>
</h:inputText>
<h:message for="textSsn1" style="color: red"> </h:message>
......
<hx:commandExButton id="button1" styleClass="commandExButton"
      type="submit" value="Enter" action="login">
</hx:commandExButton>
```

We have to compare the customer SSN to the values in the database. We do so by creating an SDO object to retrieve the records from the database.

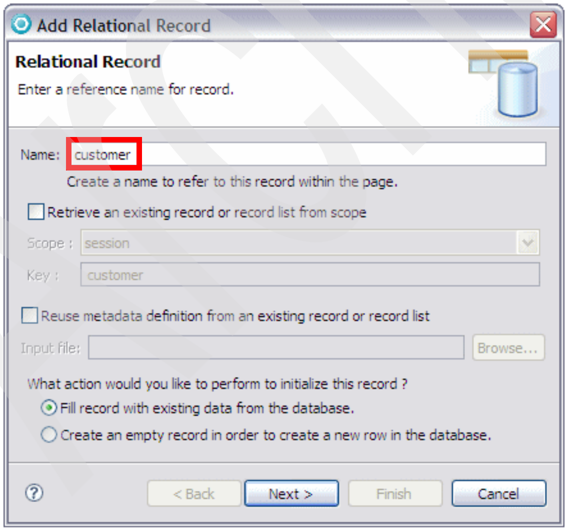
> **Important:** To retrieve records from the relational database we require a connection. We use the **ITSOBANK** connection that was created in Chapter 9, “Develop database applications” in “Creating a database connection” on page 358.

Add SDO to a Faces JSP page

When you are creating a Faces application, you can choose from among several kinds of data sources, such as relational record lists or relational records. Relational records connect to only one record from a database. Relational record lists connect to more than one record from a database. We use relational records to retrieve one customer based on the SSN.

To add an SDO relational record to a JSF page, do these steps:

- Open the **RAD7JSFLoginFragment.jsp** (if you closed it). Make sure that you do not use the `RAD7JSFLogin.jsp` (close that file if it is open).
- In the Page Data view, right-click **Relational Record** and select **New** → **Relational Record**.
- In the Add Relational Record dialog, enter **customer** in the Name field, and click **Next** (Figure 14-49).



Relational Record

Enter a reference name for record.

Name: customer

Create a name to refer to this record within the page.

□ Retrieve an existing record or record list from scope

Scope : session

Key : customer

□ Reuse metadata definition from an existing record or record list

Input file:

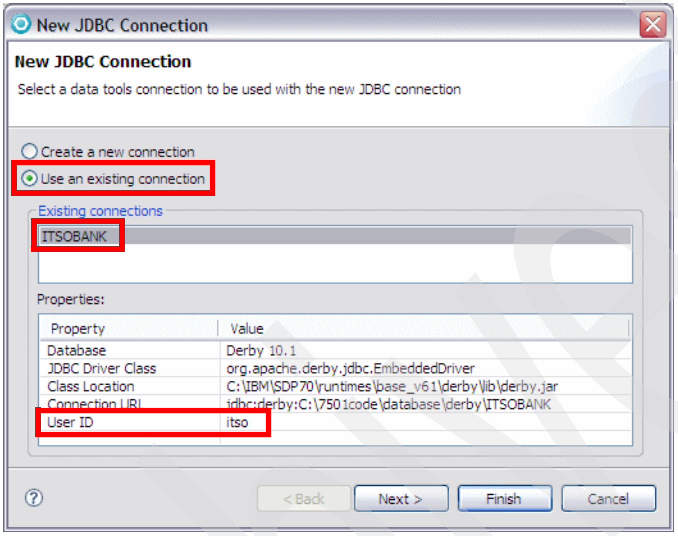
What action would you like to perform to initialize this record ?

● Fill record with existing data from the database.

○ Create an empty record in order to create a new row in the database.

Figure 14-49 Add Relational Record wizard

- In the Record Properties dialog, do these steps:
  - The first time this dialog opens, the Connection name drop-down list is empty even though the actual database connection may exist. Click **New**.
  - In the New JDBC Connection dialog, select **Use an existing connection**, select `ITSOBANK` from the list, and click **Next** (Figure 14-50).



New JDBC Connection

New JDBC Connection

Select a data tools connection to be used with the new JDBC connection

○ Create a new connection
● Use an existing connection

Existing connections

ITSOBANK

Properties:

Property	Value
Database	Derby 10.1
JDBC Driver Class	org.apache.derby.jdbc.EmbeddedDriver
Class Location	C:\IBM\SDP70\runtimes\base_v61\derby\lib\derby.jar
Connection URL	jdbc:derby:C:\7501code\database\derby\ITSOBANK
User ID	itso

Figure 14-50 Add SDO to JSF page: Select a connection

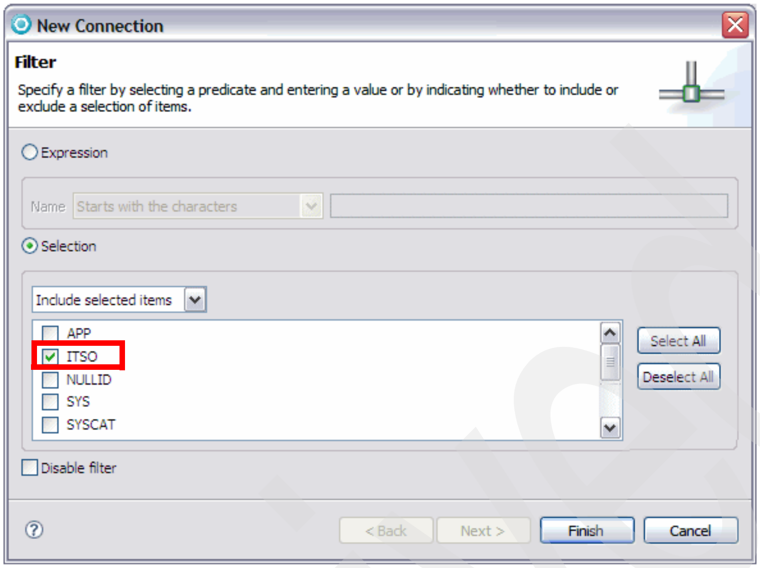
  - Enter the user ID **itso** (we are using the `ITSO` schema), leave the password empty, and click **Next**.

Important: In this step you have to ensure that **itso** is used as the user ID to connect to the Derby database. Derby considers any non-qualified reference to be in the current user's schema. You must specify a Derby user that has a schema in the Derby database you are accessing.

Refer to this technote for more information:

`http://www.ibm.com/support/docview.wss?rs=2042&context=SSRTLW&dc=DB520&uid=swg21257670&loc=en_US&cs=UTF-8&lang=en&rss=ct2042rational`

– Select the **ITSO** schema and click **Finish** (Figure 14-51).



New Connection

Filter

Specify a filter by selecting a predicate and entering a value or by indicating whether to include or exclude a selection of items.

○ Expression

Name: Starts with the characters

● Selection

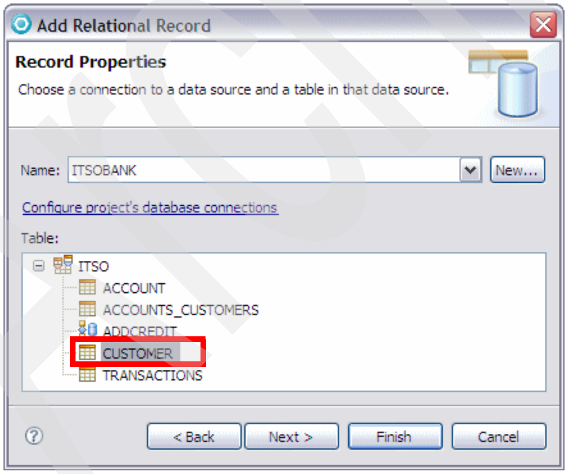
Include selected items

- □ APP
- ☑ ITSO
- □ NULLID
- □ SYS
- □ SYSCAT

□ Disable filter

Figure 14-51 SDO record: Select the database schema

► The Record Properties page is populated with the database tables. Select the **ITSO.CUSTOMER** table, and click **Next** (Figure 14-52).



Add Relational Record

Record Properties

Choose a connection to a data source and a table in that data source.

Name: ITSOBANK

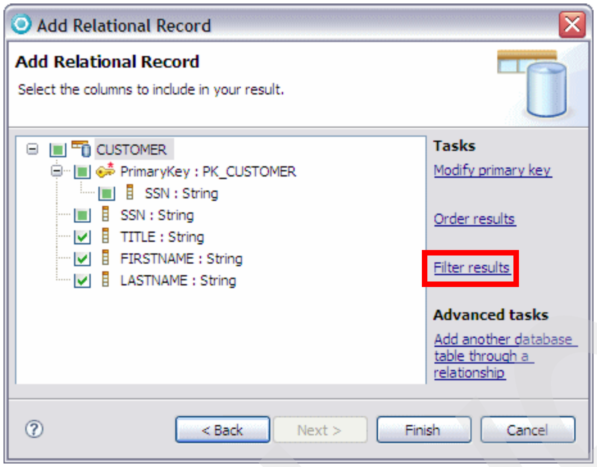
Configure project's database connections

Table:

- ITSO
  - ACCOUNT
  - ACCOUNTS_CUSTOMERS
  - ADDCREDIT
  - CUSTOMER
  - TRANSACTIONS

Figure 14-52 SDO record: Select a connection and a table

- In the Add Relational Record dialog, select all the columns (preselected) and click **Filter results** (Figure 14-53).



Add Relational Record

Add Relational Record

Select the columns to include in your result.

- ☑ CUSTOMER
  - ☑ PrimaryKey : PK_CUSTOMER
    - ☑ SSN : String
  - ☑ SSN : String
  - ☑ TITLE : String
  - ☑ FIRSTNAME : String
  - ☑ LASTNAME : String

Tasks

Modify primary key

Order results

Filter results

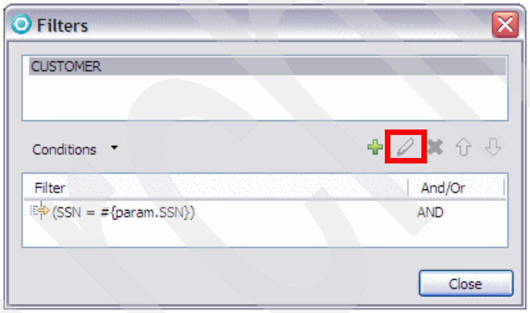
Advanced tasks

Add another database table through a relationship

Figure 14-53 SDO record: Column selection

- In the Filters dialog, one rule is preconfigured (Figure 14-54):

```
(SSN = #{param.SSN})
```



Filters

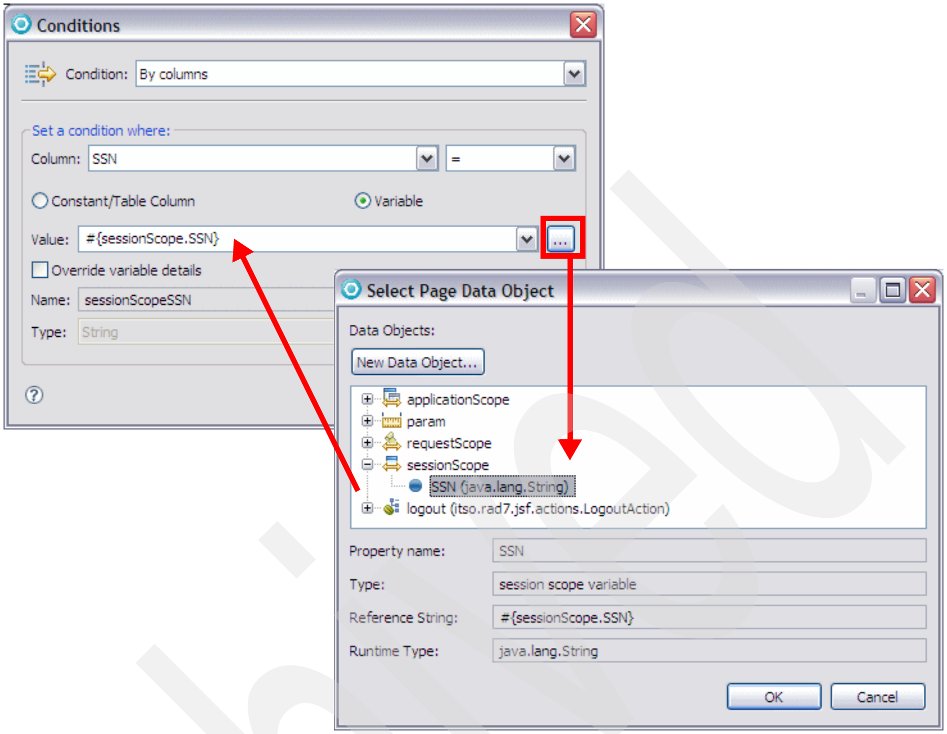
CUSTOMER

Conditions

Filter	And/Or
(SSN = #{param.SSN})	AND

Figure 14-54 SDO record: Add a filter

- This rule is almost perfect, however, we provide the SSN in a session variable. Select the rule and click Edit ✎.
- In the Conditions dialog, click ..., then select the **sessionScope** → **SSN** variable (Figure 14-55).



Condition: By columns

Set a condition where:

Column: SSN =

○ Constant/Table Column ● Variable

Value: #{sessionScope.SSN}

□ Override variable details

Name: sessionScopeSSN

Type: String

Data Objects:

New Data Object...

- applicationScope
- param
- requestScope
- sessionScope
  - SSN (java.lang.String)
- logout (itso.rad7.jsf.actions.LogoutAction)

Property name: SSN

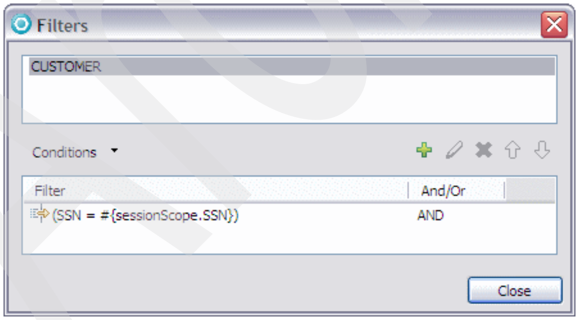
Type: session scope variable

Reference String: #{sessionScope.SSN}

Runtime Type: java.lang.String

Figure 14-55 SDO record: Editing a filter condition

- ► In the Conditions dialog, the selected value is displayed. Click **OK**.
- ► The Filters dialog is complete (Figure 14-56). Click **Close**.



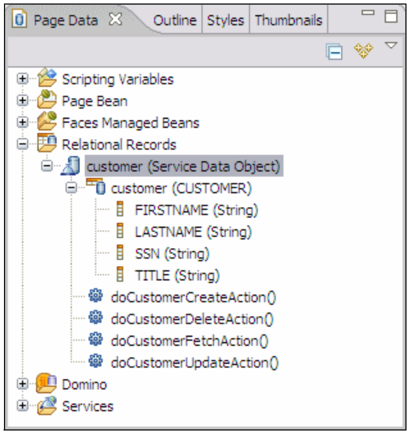
CUSTOMER

Filter	And/Or
(SSN = #{sessionScope.SSN})	AND

Figure 14-56 SDO record: Filter dialog with values

- ► In the Add Relational Record dialog, click **Finish**.
- ► Save the `RAD7JSPLoginFragment.jsp`.

The SDO is listed in the Page Data view under Relational Records (Figure 14-57). We can use the SDO to validate the customer SSN that is entered.



Page Data | Outline | Styles | Thumbnails

- Scripting Variables
- Page Bean
- Faces Managed Beans
- Relational Records
  - customer (Service Data Object)
    - customer (CUSTOMER)
      - FIRSTNAME (String)
      - LASTNAME (String)
      - SSN (String)
      - TITLE (String)
    - doCustomerCreateAction()
    - doCustomerDeleteAction()
    - doCustomerFetchAction()
    - doCustomerUpdateAction()
- Domino
- Services

Figure 14-57   Relational record in the Page Data view

As a result of creating the relational record list, several files were generated. The /WebContent/WEB-INF/wdo/**customer.xml** file is of most interest to us in that it contains the configuration details entered in the wizard.

```
<?xml version="1.0" encoding="UTF-8"?>
<com.ibm.websphere.sdo.mediator.jdbc.metadata:Metadata xmi:version="2.0"
      xmlns:xmi="http://www.omg.org/XMI"
      xmlns:com.ibm.websphere.sdo.mediator.jdbc.metadata=
            "http:///com/ibm/websphere/sdo/mediator/jdbc/metadata.ecore"
      rootTable="//@tables.0">
  <tables schemaName="ITSO" name="CUSTOMER">
    <primaryKey columns="//@tables.0/@columns.0"/>
    <columns name="SSN" type="4"/>
    <columns name="TITLE" type="4" nullable="true"/>
    <columns name="FIRSTNAME" type="4" nullable="true"/>
    <columns name="LASTNAME" type="4" nullable="true"/>
    <filter predicate="( SSN = ? )">
      <filterArguments name="sessionScopeSSN" type="4"/>
    </filter>
  </tables>
</com.ibm.websphere.sdo.mediator.jdbc.metadata:Metadata>
```

Java code is generated into the page code (RAD7JSFLoginFragment.java). To open the page code, right-click in the **RAD7JSFLoginFragment.jsp** and select **Edit Page Code**. You can see these methods in RAD7JSFLoginFragment.java:

- getSDOConnectionWrapper—Get a JDBC connection
- doCustomerUpdateAction—Update a customer record

- doCustomerDeleteAction—Delete a customer record
- getCustomerParameters—Get the parameters for retrieve
- getCustomerMediator—Get the mediator that handles all database access
- doCustomerCreateAction—Insert a customer
- doCustomerFetchAction—Fetch a customer record
- getCustomer—Retrieve a customer (calls doCustomerFetchAction)

The simplest way to use these methods is by calling getCustomer, which invokes the other methods required to perform the action.

Retrieve the customer by SSN and add dynamic navigation

To retrieve the customer and add dynamic navigation, we have to add some Java code to the page code (managed bean, RAD7JSFLoginFragment.java).

- Insert the code in Example 14-2 into the page code. The code is in:

  ```
  c:\7501code\jsfsdo\RAD7JSFLoginFragment.jpage
  ```

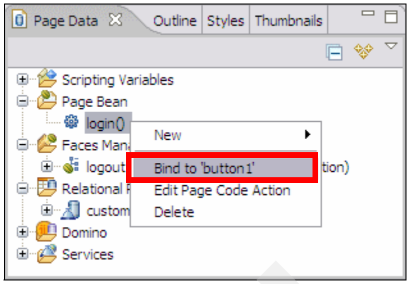
- Add the constants after the class, and the two methods at the end of the code. Add import statements by selecting **Source** → **Organize Imports** (or press Ctrl+Shift+O).
- Save the RAD7JSFLoginFragment.java file.

Example 14-2 Custom code to retrieve a customer

```
public class RAD7JSFLoginFragment extends PageCodeBase {
   public static final String OUTCOME_FAILED = "failed";
   public static final String OUTCOME_LOGIN = "login";
   ....................................................... all other code
   public String login() {
      String outcome = OUTCOME_FAILED;
      DataObject cust = getCustomer();
      // A valid customer SSN should only return one record
      if (cust == null) {
         addErrorMessage("Customer Record not found.");
      } else {
         // successfully entered a valid Customer SSN
         outcome = OUTCOME_LOGIN;
      }
      return outcome;

      protected void addErrorMessage(String error) {
      FacesMessage message = new FacesMessage(error);
      getFacesContext().addMessage(getTextSsn1()
               .getClientId(getFacesContext()), message);
   }
}
```

We have to invoke the `login` method when the **Enter** button is clicked:

- Open the **RADJSFLoginFragment.jsp** (if you closed it).
- In the Design view, select the **Enter** button.
- In the Page Data view, expand **Page Bean**.
- Right-click **login()** and select **Bind to 'button1'**, where `button1` is the ID of the Enter button (Figure 14-58).



Alternative:

Drag the login() action from the Page Data view on top of the Enter button in the Design view.

Figure 14-58 Select action from Page Data view

Binding the Enter button to the `login` method replaces the static navigation that we used before. The action attribute (Properties view or Source tab) now has the value:

```
action="#{pc_RAD7JSFLoginFragment.login}"
```

Tip: Select the **Enter** button and select the **Quick Edit** view (behind the Properties view). You can see the code of the login method. You can also edit the method code in the Quick Edit view if there is not too much code.

Now we have bound the command button (Enter) to run the `login` method, which retrieves the customer from the database using the SDO relational record. The method verifies that one customer record is retrieved, and issues an error message otherwise. The `login` method returns `login` if a customer is found, and `failed` otherwise. The `login` outcome is linked to a navigation rule, that invokes the customer details screen.

Editing the customer details page

We have completed the login page. Close the open editors. We now start work on the customer details page, where the user can update the details (name, title) of a customer.

This section describes how to edit the other pages of the sample application by using JSF and SDO.

Prepare the content area

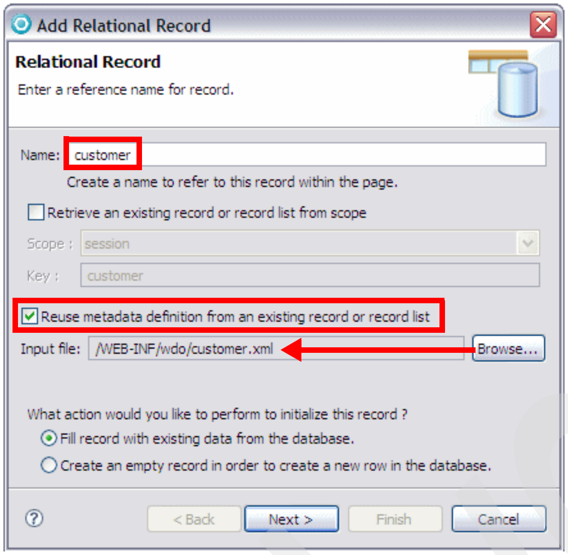
To prepare the content area, do these steps:

- Open the **RAD7JSFCustomerDetails.jsp** in the editor.
- In the Dynamic Page Template pop-up click **Yes** to configure the content area.
- In the Configure Content Areas of Dynamic Page template dialog, do these steps:
  - Highlight **bodyarea** from the content area list.
  - Select **New or existing fragment file** and enter `RAD7JSFCustomerDetailsFragment.jsp` in the Name field.
  - Click **OK**.
- The fragment file (`RAD7JSFCustomerDetailsFragment.jsp`) is created under `WebContent/tilesContent` directory and included in the actual `RAD7JSFCustomerDetails.jsp`. Now you can design the page.
- Close the `RAD7JSFCustomerDetails.jsp` and open the **RAD7JSFCustomerDetailsFragment.jsp**.
- Select the default text `Place RAD7JSFCustomerDetailsFragment.jsp's content here` and **Delete**.

Add SDO relational record

We reuse the customer relational record created for the login page. To add the SDO relational record, do these steps:

- In the Page Data view, right-click **Relational Record** and select **New** → **Relational Record**.
- In the Add Relational Record dialog (Figure 14-59):
  - Enter **customer** in the Name field.
  - Select **Reuse metadata definition from an existing record or record list**.
  - Click **Browse** and select the **customer.xml** file.
  - Click **Next**.



Relational Record

Enter a reference name for record.

Name: customer

Create a name to refer to this record within the page.

□ Retrieve an existing record or record list from scope

Scope : session

Key : customer

☑ Reuse metadata definition from an existing record or record list

Input file: /WEB-INF/wdo/customer.xml

What action would you like to perform to initialize this record ?

● Fill record with existing data from the database.

○ Create an empty record in order to create a new row in the database.

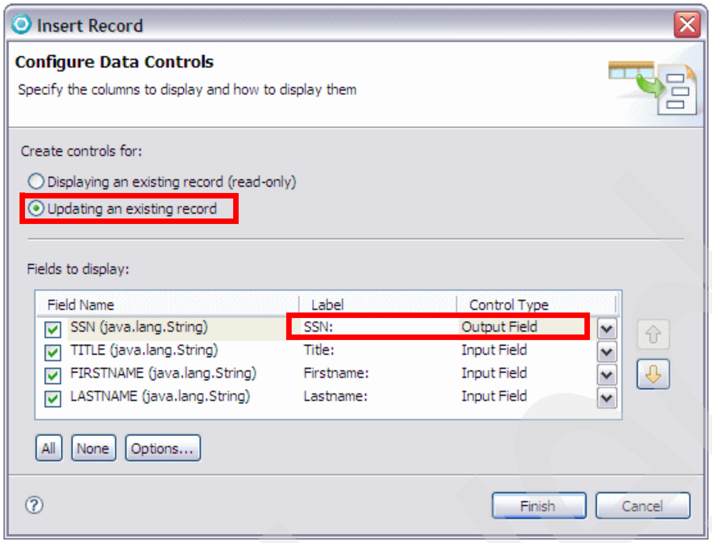
Figure 14-59 Adding an existing relational record

- In Record Properties page, click **Next**.
- In the Filters page click **Finish**.
- Select the **ITSO.CUSTOMER** table and click **Next**.
- Save the page. The relation record appears in the Page Data view (see Figure 14-57 on page 631).

Add the relational record to the page

To add the relational record to the page, do these steps:

- In the Page Data view, expand **Relational Records** → **customer (Service Data Object)**, select the **customer (CUSTOMER)** relational record, and drag it into the content area of the page. The help-tip shows the text *Drop here to insert new controls for "customer"*.
- In the Insert Record - Configure Data Controls dialog, select **Updating an existing record**, change the SSN label to `SSN:`, and change the SSN control type to **Output Field** (Figure 14-60).



Insert Record

Configure Data Controls

Specify the columns to display and how to display them

Create controls for:

○ Displaying an existing record (read-only)

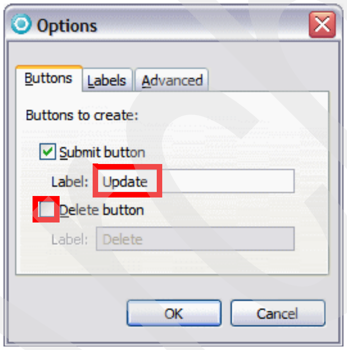
● Updating an existing record

Fields to display:

Field Name	Label	Control Type
☑ SSN (java.lang.String)	SSN:	Output Field
☑ TITLE (java.lang.String)	Title:	Input Field
☑ FIRSTNAME (java.lang.String)	Firstname:	Input Field
☑ LASTNAME (java.lang.String)	Lastname:	Input Field

Figure 14-60 Insert Record wizard

- Click **Options and** In the Options dialog, clear **Delete button**, enter **Update** in the Label field for the Submit button, and click **OK** (Figure 14-61).



Options

Buttons to create:

☑ Submit button

Label: Update

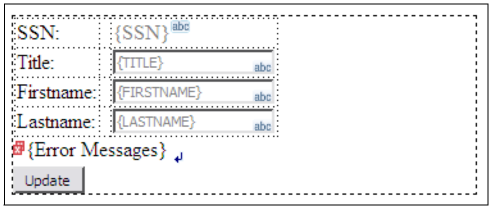
□ Delete button

Label: Delete

Figure 14-61 Insert Record Options dialog

- In the Insert Record - Configure Data Controls dialog, click **Finish**.
- Save the `RAD7JSFCustomerDetailsFragment.jsp`.

The customer details can now be displayed in the Web page, with the input fields available to update the data (Figure 14-62).



SSN: {SSN}

Title: {TITLE}

Firstname: {FIRSTNAME}

Lastname: {LASTNAME}

{Error Messages}

Figure 14-62 Customer details form for updating

Select the **Update** button and study the code in the Quick Edit view. The button is linked automatically to the `doCustomerUpdateAction` method. This is also visible in the Properties view after clicking the **All Attributes** icon.

Create a relational record list to display the accounts

In this section we demonstrate how to add a relational record list for the accounts of the customer to the page.

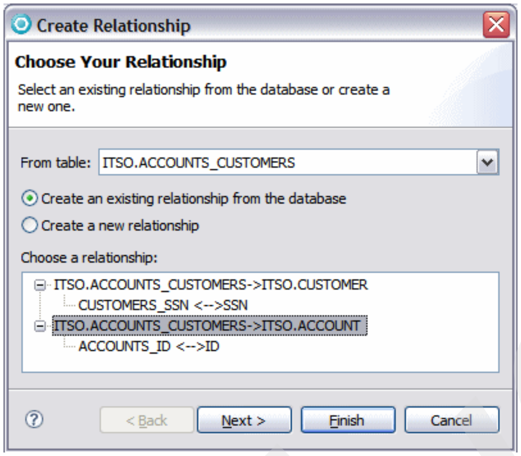
To add the relational record list for the accounts, do these steps:

- In the Page Data view, right-click **Relational Record** and select **New** → **Relational Record List**.

> **Note:** You can also select the **Relational Record List** in the Data drawer of the Palette and drag it into the JSF fragment in Page Designer.

- In the Add Relational Record List dialog, enter **accounts** in the Name field and click **Next**.
- Select the **ITSO.ACCOUNTS_CUSTOMERS** table and click **Next**.
- Click the **Filter results** under the Tasks on the right of the dialog.
- In the Filters dialog, click the + icon and add a condition to retrieve by SSN:
  - Select the **CUSTOMERS_SSN** from the Column drop-down list.
  - Select **Variable** and click the ... icon to the right of Value.
  - In the Select Page Data Object dialog, select **sessionScope** → **SSN** and click **OK**.
  - In the Conditions dialog, click **OK**.
  - In the Filters dialog, click **Close**.
- In the Add Relational Record List dialog, click **Add another database table through a relationship** under Advanced tasks.
  - In the Choose Your Relationship dialog, select **Create an existing relationship from the database**.

- Select the **ITSO.ACCOUNTS_CUSTOMERS → ITSO.ACCOUNT** relationship and click **Next** (Figure 14-63).



Create Relationship

Choose Your Relationship

Select an existing relationship from the database or create a new one.

From table: ITSO.ACCOUNTS_CUSTOMERS

● Create an existing relationship from the database

○ Create a new relationship

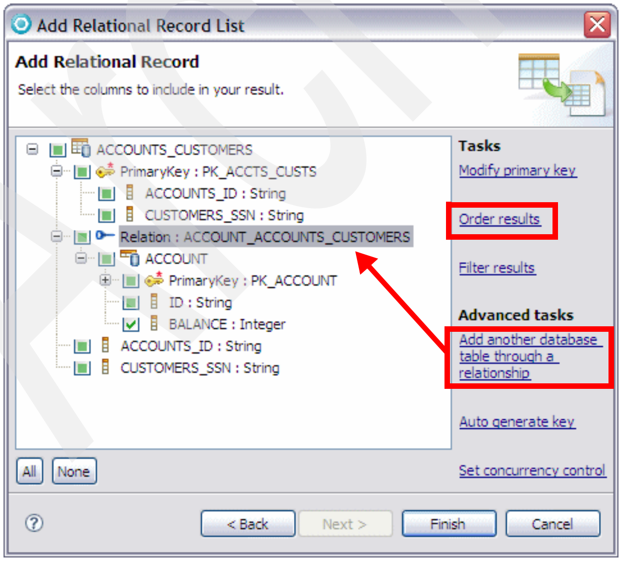
Choose a relationship:

```
⊟ ITSO.ACCOUNTS_CUSTOMERS->ITSO.CUSTOMER
     CUSTOMERS_SSN <-->SSN
⊟ ITSO.ACCOUNTS_CUSTOMERS->ITSO.ACCOUNT
     ACCOUNTS_ID <-->ID
```

Figure 14-63 Create Relationship dialog

- In the Edit Your Relationship dialog, verify the following settings:
  - Multiplicity: ***->1 FOREIGN KEY → PRIMARY KEY**.
  - **ACCOUNTS_ID:String** is mapped to **ID:String**.
  - Click **Finish**.

► The relational record list is now complete (Example 14-64).



Add Relational Record List

Add Relational Record

Select the columns to include in your result.

```
⊟ ■ ACCOUNTS_CUSTOMERS
   ⊟ ■ PrimaryKey : PK_ACCTS_CUSTS
        ■ ACCOUNTS_ID : String
        ■ CUSTOMERS_SSN : String
   ⊟ ■ Relation : ACCOUNT_ACCOUNTS_CUSTOMERS
      ⊟ ■ ACCOUNT
         ⊞ ■ PrimaryKey : PK_ACCOUNT
           ■ ID : String
           ☑ BALANCE : Integer
     ■ ACCOUNTS_ID : String
     ■ CUSTOMERS_SSN : String
```

Figure 14-64 Relational record list with a relationship

- Click **Order results** to sort the accounts by balance. In the Orders dialog, select **ACCOUNT**, then select the **BALANCE** column and click **>**. Select the **BALANCE** column under Order by and select **Descending** (Figure 14-65).

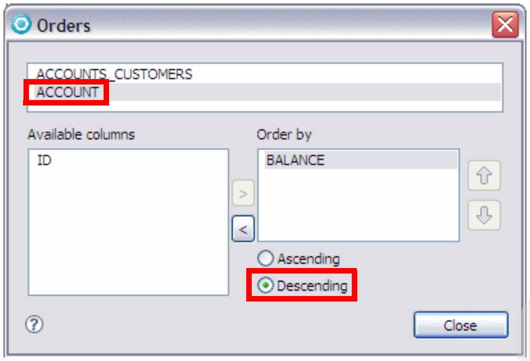


Figure 14-65 Sorting a relational record list

- Click **Finish** to generate the code. Save the `RAD7JSFCustomerDetails.jsp`.

Add the relational record list to the page

To add the relational record list to the page, do these steps:

- In the Page Data view, expand **Relational Records** → **accounts (Service Data Object)**, select the **accounts (ACCOUNTS_CUSTOMER)** relational record and drag it into the area under the Update button (Figure 14-66).

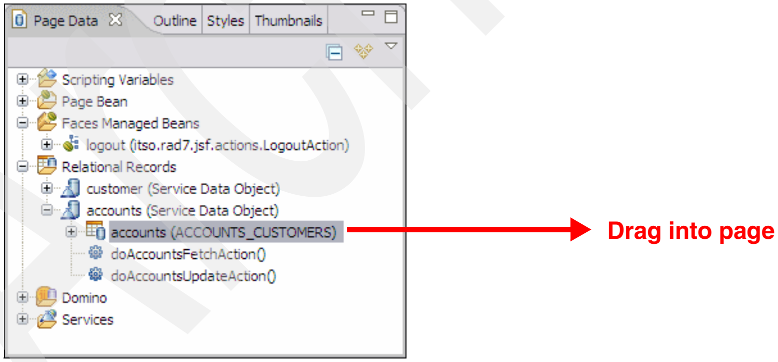
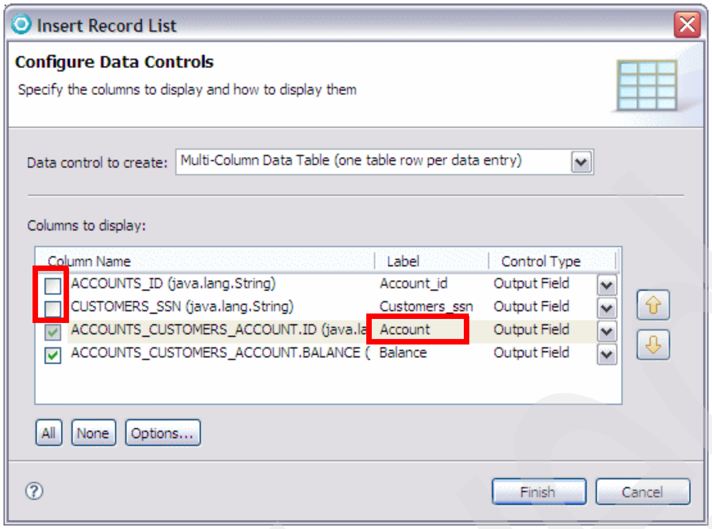


Figure 14-66 Page Data view with customer record and accounts record list

- In the Insert Record List - Configure Data Control dialog (Figure 14-67):
  – Clear **ACCOUNTS_ID** and **CUSTOMERS_SSN**.
  – Change the label for `ACCOUNT.ID` to **Account**.
  – Click **Finish**.



Configure Data Controls

Specify the columns to display and how to display them

Data control to create: Multi-Column Data Table (one table row per data entry)

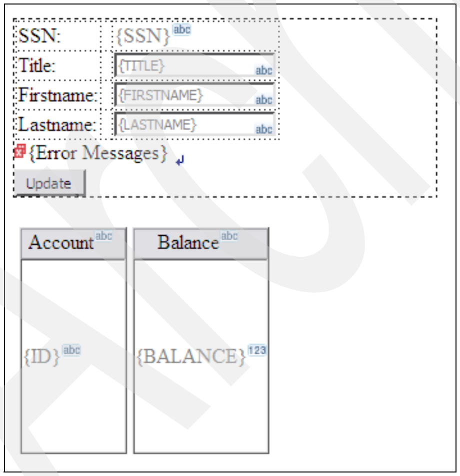
Columns to display:

	Column Name	Label	Control Type
□	ACCOUNTS_ID (java.lang.String)	Account_id	Output Field
□	CUSTOMERS_SSN (java.lang.String)	Customers_ssn	Output Field
☑	ACCOUNTS_CUSTOMERS_ACCOUNT.ID (java.la	Account	Output Field
☑	ACCOUNTS_CUSTOMERS_ACCOUNT.BALANCE (	Balance	Output Field

Figure 14-67 Insert Record List

- Save the RAD7JSFCustomerDetails.jsp.

The RAD7JSFCustomerDetails.jsp is shown in Figure 14-68 in the Design view.



SSN:	{SSN}
Title:	{TITLE}
Firstname:	{FIRSTNAME}
Lastname:	{LASTNAME}

{Error Messages}

Account	Balance
{ID}	{BALANCE}

Figure 14-68 Design view with the table for customer accounts

Change the formatting of the balance field

To change the formatting of the balance field, do these steps:

- Select the **{BALANCE}** output component in the Design view.

- In the Properties view, select **Mask** for Format and enter a Mask value of **########.##** (the balance is stored as an integer in cents).
- Select the **Balance** column (click in the empty space inside the table).
- In the Properties view, **hx:columnEx** tab, select **Right** for Alignment: Horizontal.
- Save the `RAD7JSFCustomerDetails.jsp`.

Change the formatting of the data table

If your data table returns a large number of records, you can add paging options so that the user can page through the records, displaying a specified number of rows at a time. You can choose from a variety of paging formats, which enable the user to move back and forth between the pages of data.

Our sample data has only three accounts per customer, so we arbitrarily set the number of rows to two (not very realistic).

To add paging to your data table do these steps:

- Click inside the data table grid (you can click on any column).
- In the Properties view, select **hx:dataTableEx** → **Display options**. Enter **2** for Rows per page field, and click **Add a deluxe pager** (Figure 14-69).

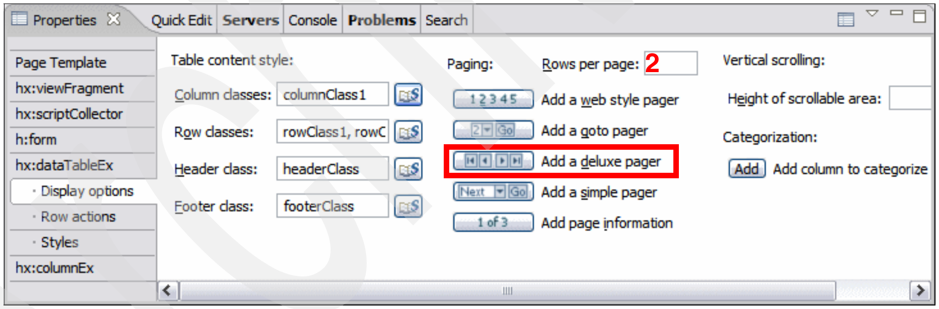


Table content style:

Column classes: columnClass1

Row classes: rowClass1, rowC

Header class: headerClass

Footer class: footerClass

Paging: Rows per page: 2

1 2 3 4 5 Add a web style pager

2 Go Add a goto pager

Add a deluxe pager

Next Go Add a simple pager

1 of 3 Add page information

Vertical scrolling:

Height of scrollable area:

Categorization:

Add Add column to categorize

Figure 14-69 Add a deluxe pager

This action creates a pager with arrow controls for first, back, forward, and last, and a text indicating *Page n of nn*.

By default, the pager is created in the data table footer (Figure 14-70). You can reposition the pager by dragging it into the header or into one of the columns.

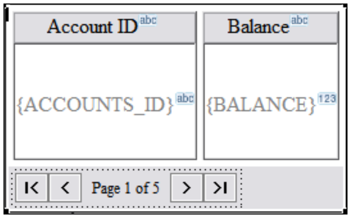
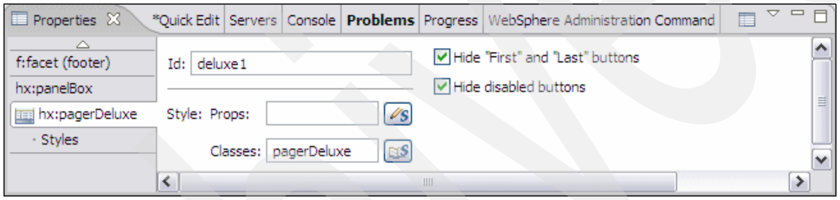


Figure 14-70 Data table with deluxe pager

You can also change the design of deluxe pager by changing the attributes of the deluxe pager component:

- Select the deluxe pager component from the Design view.
- In the Properties view, select **Hide "First" and "Last" buttons** and **Hide disabled button** (Figure 14-71).



Id: deluxe1

☑ Hide "First" and "Last" buttons

☑ Hide disabled buttons

Style: Props:

Classes: pagerDeluxe

Figure 14-71 Deluxe pager attributes

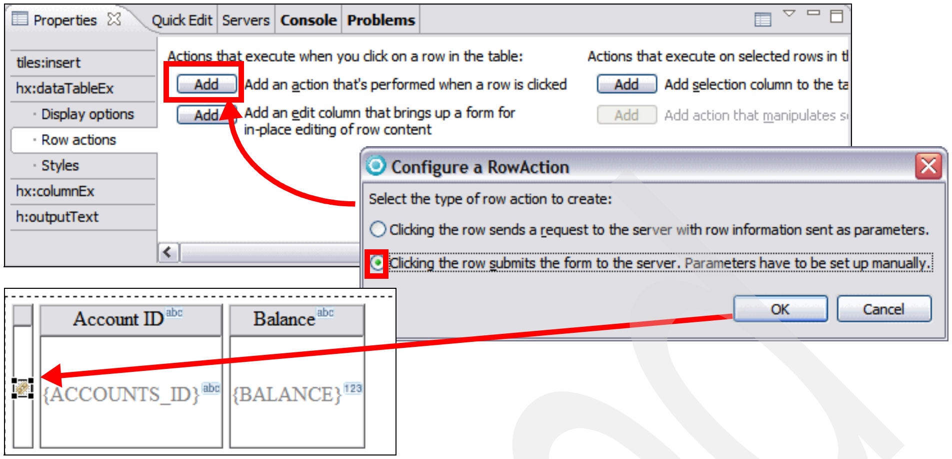
Now first and last buttons are hidden from the deluxe pager in the Design view.

Add a row action

We now add a row action that is activated when the user clicks a row of the accounts table. We want to use this action to display the account details page with the transactions.

To add a row action, do these steps:

- Select the data table component from the Design view.
- In the Properties view, select **hx:dataTableEx** → **Row actions**.
- Click **Add** next to Add an action that's performed when a row is clicked. In the Configure a Row Action dialog, select **Clicking the row submits the form to the server. Parameters have to be set up manually** (Figure 14-72).



Actions that execute when you click on a row in the table:

Add an action that's performed when a row is clicked

Add an edit column that brings up a form for in-place editing of row content

Actions that execute on selected rows in t

Add selection column to the ta

Add action that manipulates s

Configure a RowAction

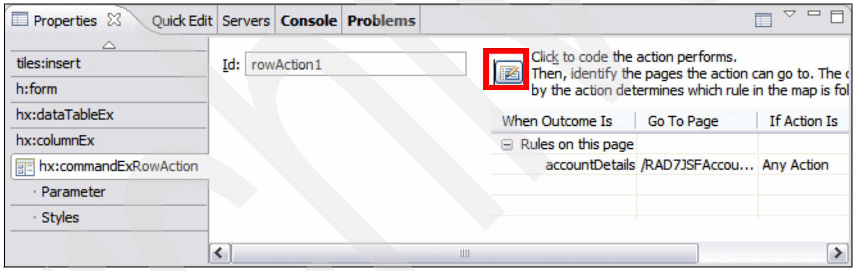
Select the type of row action to create:

○ Clicking the row sends a request to the server with row information sent as parameters.

● Clicking the row submits the form to the server. Parameters have to be set up manually.

Figure 14-72   Add a row action

- Select the row action added in the Design view, and in the Properties view, click **Switch to QuickEdit View** (Figure 14-73).



Id: rowAction1

Click to code the action performs. Then, identify the pages the action can go to. The by the action determines which rule in the map is fol

When Outcome Is	Go To Page	If Action Is
⊟ Rules on this page		
accountDetails	/RAD7JSFAccou...	Any Action

Figure 14-73   Row action: Switch to Quick Edit view

- In the Quick Edit view, read the code in comments, and replace the return statement with the code from Example 14-3, which is available in:

  `c:\7501code\jsfsdo\CustomerDetailRowAction.jpage`

Example 14-3   Code for accounts row action

```
try {
   int row = getRowAction1().getRowIndex();
   Object keyvalue = ((DataObject)getAccounts().get(row)).get("ACCOUNTS_ID");
   getSessionScope().put("accountId", keyvalue);
} catch(Exception e){
   e.printStackTrace();
}
return "accountDetails";
```

- This action code retrieves the index of the selected row, then gets the account ID, and stores the ID in session scope for the next page.
- Create a session scope variable named **accountId**, which is used by the code that is added for row action:
  - In the Page Data view, right-click **Scripting Variables** and select **New** → **Session Scope Variable**.
  - In the Add Session Scope Variable dialog, enter `accountId` as name and `java.lang.String` as type.

Add a reusable JavaBean for logout

We now add the reusable JavaBean getter method to the page to handle the logout process:

- In the Page Data view, expand **Faces Managed Beans**.
- Right-click **logout** and select **Add Getter to Page Code**.
- The `RAD7JSFCustomerDetailsFragment.java` class opens in the Java editor. Verify that the `getLogout` method is added as displayed in Example 14-4.

Example 14-4 getLogout method for reusable JavaBean

```
protected LogoutAction getLogout() {
      if (logout == null) {
         logout = (LogoutAction) getFacesContext().getApplication()
                 .createValueBinding("#{logout}")
                 .getValue(getFacesContext());
      }
      return logout;
   }
```

- Save and close the `RAD7JSFCustomerDetailsFragment.java`.

Add a logout button

To add a command button to the page and bind the `logout` method to the command button, do these steps:

- From the Enhanced Faces Components, select the **Button - Command** Button - Command and drag it onto the page on top of the Update button so that it is placed next to the Update button.
- Select the Command Button (Submit) in the Design view.
- In the Properties view, select the **Display options** tab, and change the Button label to **Logout**.

- In the Page Data view, select **Faces Managed Beans → logout (itso.....) → logout** and drag it on top of the **Logout** button (this binds the button to the action).
- Add a few empty lines (press **Enter**) between the buttons and the accounts table.
- Save and close the `RAD7JSFCustomerDetailsFragment.jsp`.

The `RAD7JSFCustomerDetails.jsp` page is now complete.

Editing the account details page

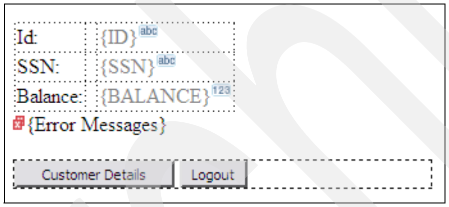
This section describes the steps to complete the `RAD7JSFAccountDetails.jsp`. In many cases, the details for these steps can be found in the procedures used to complete the `RAD7JSFCustomerDetails.jsp`.

To complete the `RAD7JSFAccountsDetails.jsp`, do these steps:

- Open the **RAD7JSFAccountDetails.jsp** in the editor.
- In the Dynamic Page Template pop-up click **Yes** to configure the content area. Highlight **bodyarea** from the content area list, select **New or existing fragment file** and enter `RAD7JSFAccountDetailsFragment.jsp` in the Name field. Click **OK** to create the `RAD7JSFAccountDetailsFragment.jsp` file. Now you can design the page.
- Close the `RAD7JSFAccountsDetails.jsp` and open the `RAD7JSFAccountDetailsFragment.jsp`.
- Delete the default text `Place RAD7JSFAccountDetailsFragment.jsp's content here`.
- In the Page Data view, right-click **Relational Record** and select **New → Relational Record** (see "Add SDO relational record" on page 634):
  - Name the relational record **account**.
  - Select the **ITSO.ACCOUNT** table.
  - Change the default filter to ID equals **sessionScope.accountId**.
  - Click **Finish**.
- From the Page Data view, drag the **account** relational record onto the page (see "Add the relational record to the page" on page 635). Select **Display an existing record (read-only)** is selected. Click **Finish**.
- Change the `{BALANCE}` output component format to **Mask** and a mask value of **$########.##** (Properties view `h:outputText`).
- Under the table, add a command button with the label **Customer Details**.

- Select the button, and in the Quick Edit view, click into the code and change the return statement to `return "customerDetails";` to produce the required outcome for the navigation rule.
- Add the reusable **logout** Java Bean getter method as we did in the previous section (right-click **logout(...)** and select **Add Getter to Page Code**).
- Add a **Logout** command button and bind the logout action to the command button (drag **logout(itso...)** → **logout** from the Page Data view onto the button).
- Add a row to the account information table containing the customer `SSN`:
  - Place the cursor in the `Id:` cell
  - Right-click and select **Table** → **Add Row Below**.
  - Enter `SSN:` in the left-most cell of the new row.
  - Drag an Output component from the Palette into the right cell of the new row.
  - Drag the `SSN` session variable on top of the output component.

The resulting page is shown in Figure 14-74.



Id:	{ID}
SSN:	{SSN}
Balance:	{BALANCE}

{Error Messages}

Figure 14-74 Account details page design

Displaying the account transactions

We also want to display the transactions of the account in a table:

- Create a relational record list named **transactions:**
  - Select the **ITSO.TRANSACTIONS** table.
  - Add a filter as **ACCOUNTS_ID** equals **sessionScope.accountId**.
  - Click **Order results** and select **TRANSTIME**, ascending.
  - Click **Finish**.
- Drag the **transactions(TRANSACTIONS)** into the page:
  - Clear **ID** (generated unique ID) and **ACCOUNTS_ID** (always the same).
  - Change the labels to **Type** and **Time**.
  - Move **TRANSTIME** before **TRANSTYPE**.

  - Click **Finish**.

- Change the **{TRANSTIME}** output component format type to **Date and time** (Properties view, `h:outputText`, the default is Date only)
- Format the **{AMOUNT}** column as **Mask** with a mask of **######.##** and right-adjusted (`hx:columnEx`).
- Change the table border to **1** (Properties view, hx:dataTableEx).
- The completed page is shown in Figure 14-75.

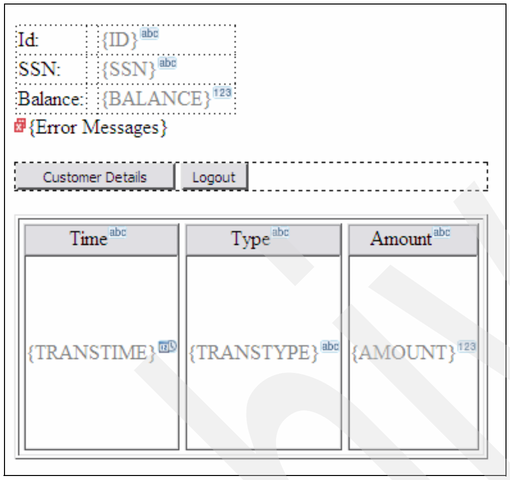


Figure 14-75 Account details page design with transactions

Running the sample Web application

This section demonstrates how to run the sample Web application built using JSF and SDO.

Prerequisites to run sample Web application

To run the Web application you must have completed the sample JSF application:

- Either you completed the JSF pages as described in "Developing a Web application using JSF and SDO" on page 599.
- Or, you can import the interchange file of the application from:

  `c:\7501code\jsfsdo\RAD7JSFapp.zip`

  Refer to "Importing sample code from a project interchange file" on page 1310 for details.

- You also must have set up the ITSOBANK database as described in “Setting up the sample database” on page 598.

Preparation

SDO generated a data source for the ITSOBANK database with the JNDI name jdbc/ITSOBANK. For other chapters we already defined a data source in the server with the JNDI name of jdbc/itsobank. This can lead to conflicts.

To change the JNDI name, do these steps:

- In the Project Explorer, click the down arrow (top right) and select **Filters**. Clear **.* resources** and click **OK**.
- Open the **.wdo-connections** file in the RAD7JSF project. Change the JNDI name in the <runtime> tag to:

```
<runtime xsi:type="connections:datasource-connection" auto-deploy="true"
   class-location="C:\....\SDP70\runtimes\base_v61\derby\lib\derby.jar"
   id="ITSOBANK_runtime"
   classname="org.apache.derby.jdbc.EmbeddedConnectionPoolDataSource"
   database-location="C:\7501code\database\derby\ITSOBANK"
   database-name="ITSOBANK" jndi-name="jdbc/itsobank"
   resource-reference-name="ITSOBANK"
   server-name="derby:C:\7501code\database\derby\ITSOBANK"
   sql-vendor-type="36" userid="itso"/>
```

  Save and close the file.

- Open the deployment descriptor of the **RAD7JSFEAR** project:
  – Select the **Deployment** tag. This is the WebSphere Enhanced EAR.
  – Select the **WDO Derby JDBC Provider**. The data source section shows now an ITSOBANK data source.
  – Select the **ITSOBANK** data source and click **Edit**. Change the JNDI name to **jdbc/itsobank**, and click **Finish**.
  – Expand **Authentication** (further down). There should be an entry **wdo_ITSOBANK** with a user ID of **itso**.
- Close all the files.
- In the Servers view, start the WebSphere Application Server v6.1. Wait until the server is ready.
- Right-click the server and select **Add and Remove Projects**. Select the **RAD7JSFEAR** project and click **Add >**. Click **Finish** and wait until the application is started.

Running the sample Web application

To run the sample Web application in the WebSphere test environment, do these steps:

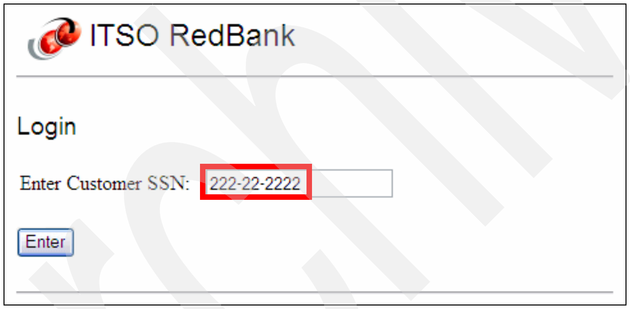
- In the Project Explorer, expand **RAD7JSF** → **WebContent**.
- Right-click **RAD7JSFLogin.jsp**, and select **Run As** → **Run on Server**.
- In the Run on Server dialog, select **Choose an existing server**, select **WebSphere Application Server v6.1**, and click **Finish**.

The Web application Login page is displayed in a Web browser.

Verifying the sample Web application

Once you have launched the application by running it on the test server, there are some basic steps that can be taken to verify that the Web application using JSF and SDO is working properly.

- From the ITSO RedBank login page, enter `222-22-2222` in the customer SSN field, and click **Enter** (Figure 14-76).



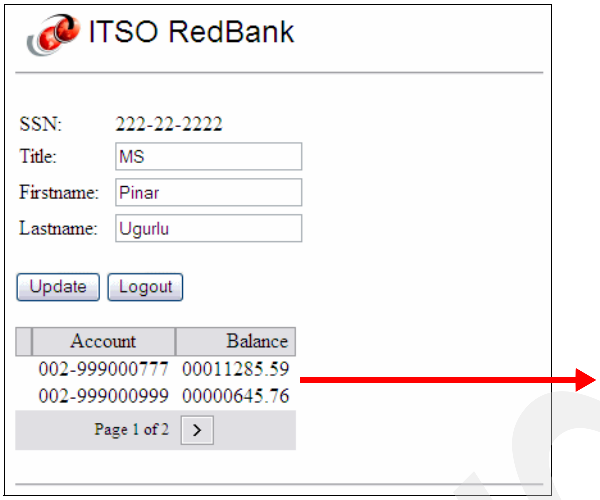
ITSO RedBank

Login

Enter Customer SSN: 222-22-2222

Figure 14-76 ITSO RedBank JSF Login page

- The customer details page is displayed (Figure 14-77).
- The resulting page has the deluxe pager with first and last buttons hidden. Because we specified two rows per page, two accounts are listed and the pager has a forward button. You can test the page by clicking the > and the < buttons.



ITSO RedBank

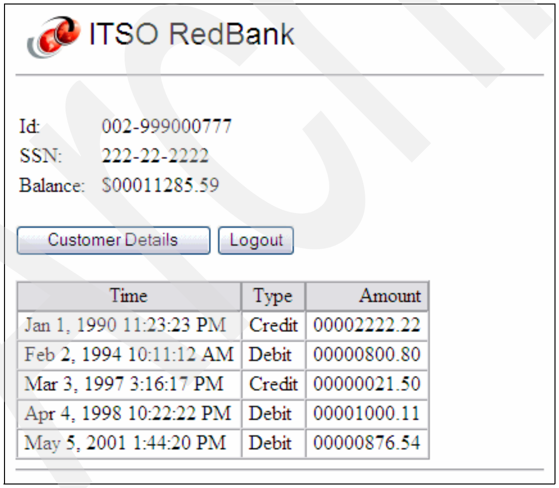
SSN:	222-22-2222
Title:	MS
Firstname:	Pinar
Lastname:	Ugurlu

Account	Balance
002-999000777	00011285.59
002-999000999	00000645.76

Note: The formatting of the balance could be improved. Unfortunately the database was designed with an integer column to store the balance in cents.

Figure 14-77 Display of customer accounts

- Change the fields to update customer information. This verifies write access to the database using SDO. For example, change the Title to **Miss** and click **Update**.
- Click one of the accounts (002-999000777) to display the account information with the transactions, resulting in the page displayed in Figure 14-78.



ITSO RedBank

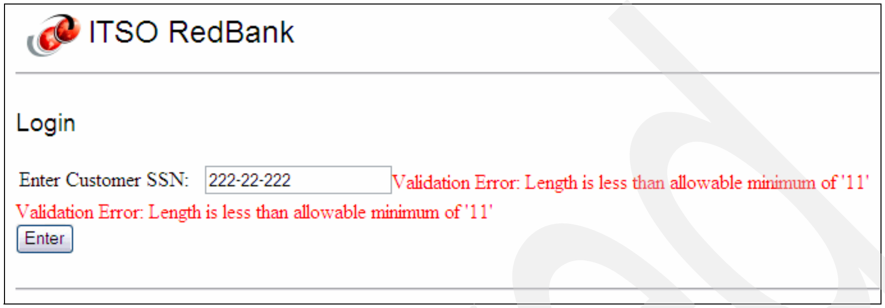
Id:	002-999000777
SSN:	222-22-2222
Balance:	$00011285.59

Time	Type	Amount
Jan 1, 1990 11:23:23 PM	Credit	00002222.22
Feb 2, 1994 10:11:12 AM	Debit	00000800.80
Mar 3, 1997 3:16:17 PM	Credit	00000021.50
Apr 4, 1998 10:22:22 PM	Debit	00001000.11
May 5, 2001 1:44:20 PM	Debit	00000876.54

Figure 14-78 Details for a selected account with transactions

- Click **Customer Details** to return to the previous page. Select another account to display the details.

- Click **Logout** to return to the login page.
- To test the validation in the login page:
  - Enter **222-22-222** and click **Enter**. A validation error is displayed (Figure 14-79).



Enter Customer SSN: 222-22-222 Validation Error: Length is less than allowable minimum of '11'

Validation Error: Length is less than allowable minimum of '11'

Figure 14-79 Testing the minimum length validation

  - Note that the error messages are duplicated in the error message field of the input field and in the error message field of the page. We could certainly eliminate one of the two message fields.
  - Type `222-22-2221` and click **Enter**. You receive an error message saying **Customer record not found**. Note that this action displays error trace information in the Console view:

```
[...] 00000022 SystemOut     O org.eclipse.emf.common.util.
       BasicEList$BasicIndexOutOfBoundsException: index=0, size=0
   at org.eclipse.emf.common.util.BasicEList.get(BasicEList.java:511)
   at pagecode.tilesContent.RAD7JSFLoginFragment.doCustomerFetchAction
       (RAD7JSFLoginFragment.java:233)
   at pagecode.tilesContent.RAD7JSFLoginFragment.getCustomer
       (RAD7JSFLoginFragment.java:260)
   at pagecode.tilesContent.RAD7JSFLoginFragment.login
       (RAD7JSFLoginFragment.java:269)
```

  - Leave the SSN empty and click **Enter**. You get an error that the value is required.
- You can enter other SSN values, however, only 222-22-2222 and 000-00-0000 (and maybe 111-11-1111 from the database chapter) have transactions associated with the accounts.

So far we have a JSF application with SDO access to a relational database. We could certainly enhance the appearance of the pages and the error messages. We could also provide a transactions page where we could allow for deposit and withdraw transactions.

Adding AJAX behavior to the sample Web application

In the section we use the AJAX support of Application Developer to enhance the application.

Stop the server so that we can connect to the Derby database.

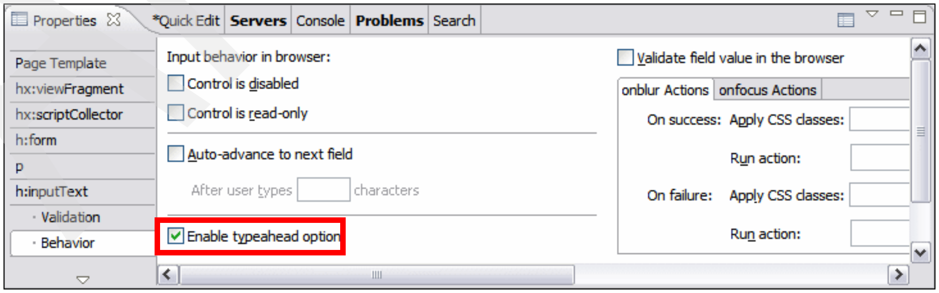
Adding type-ahead control to the login page

When a type-ahead control is attached to an input field, as a user types in the field, a list of suggestions is constructed and displayed in a pop-up list box attached to the field. Alternatively, as a user types, the field can be automatically completed with a suggestion (and the user can keyboard to additional suggestions. Type-ahead uses AJAX requests to communicate with the server when building the list of suggestions. As a result, the page remains on display (it is not submitted/redrawn) while the list is being build. The user interface remains active while the list is being built.

In our sample, type-ahead control is used to generate suggestions for customer SSN numbers. To keep the code simple, we generate static SSN numbers, but in real applications, you would probably require complex generators.

To add type-ahead control to the login page, do these steps:

- In the Project Explorer view, expand **RAD7JSF** → **WebContent** → **tilesContent**.
- Open the **RAD7JSFLoginFragment.jsp** in the editor (Design tab).
- Select the **{SSN}** input text component.
- In the Properties view, select **Behavior** under `h:inputText`.
- Select **Enable typeahead option** (Figure 14-80).



Input behavior in browser:

□ Control is disabled

□ Control is read-only

□ Auto-advance to next field

After user types [ ] characters

☑ Enable typeahead option

□ Validate field value in the browser

On success: Apply CSS classes:

Run action:

On failure: Apply CSS classes:

Run action:

Figure 14-80 Enable typeahead option

The JSP is shown in the Design tab in Figure 14-81. Notice that the input field is now type-ahead support enabled .

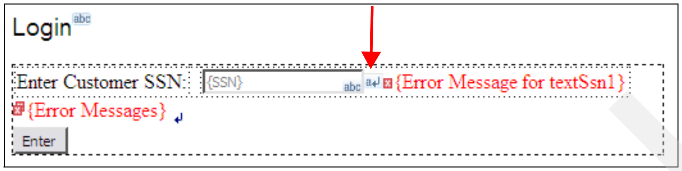


Figure 14-81 Typeahead enabled input component

We have to generate a suggestion list for the type-ahead component. We can do that by simply implementing a Java class that generates suggestions and returns them in an `ArrayList` object. In our sample suggestions for the customer SSN is generated.

To create the suggestions class, do these steps:

- In the Project Explorer view, expand **RAD7JSF**.
- Right-click **Java Resources: src** and select **New** → **Class**.
- In the New Java Class dialog, enter the following items and click **Finish.**
  - Package: `itso.rad7.jsf.suggestions`
  - Name: `CustomerSsnSuggestions`
  - Superclass: `java.util.AbstractMap`
- The `CustomerSsnSuggestions` class opens in the editor. Change the source code to match Example 14-5 (you can copy/paste from `c:\7501code\jsfsdo\CustomerSsnSuggestions.jpage`.

Example 14-5 CustomerSsnSuggestions class

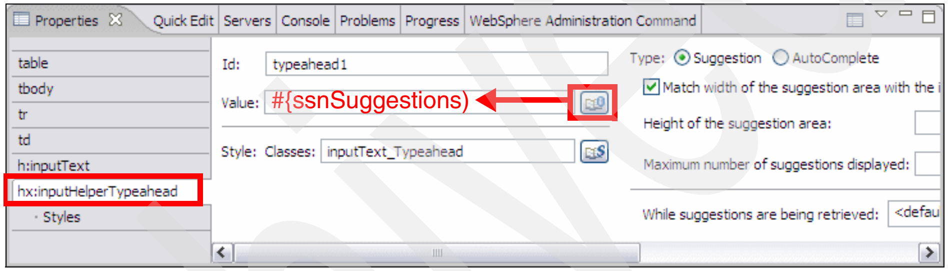
```
package itso.rad7.jsf.suggestions;
import java.util.AbstractMap;
import java.util.ArrayList;
import java.util.Set;

public class CustomerSsnSuggestions extends AbstractMap {

   public Set entrySet() {
      return null;
   }

   public Object get(Object key) {
      int arrayLength = 10;
      ArrayList ssnSuggestions = new ArrayList(arrayLength);
      for (int i = 0; i < arrayLength; i++) {
         ssnSuggestions.add(i+""+i+""+i+"-"+i+""+i+"-"+i+""+i+""+i+""+i);
```

```
        }
        return ssnSuggestions;
    }}
```

To bind the `CustomerSsnSuggestions` class to type-ahead component, do these steps:

- Select the typeahead component in the Design view.
- In the Properties view, click **Browse** for the Value field (Figure 14-82).

  You can examine the other attributes of type-ahead component from the Properties view. Some of these attributes are: maximum number of suggestions displayed, delay time before a suggestion is made, and match width of the suggestion area with the input field.



Id: typeahead1

Value: #{ssnSuggestions}

Style: Classes: inputText_Typeahead

Type: ● Suggestion ○ AutoComplete

☑ Match width of the suggestion area with the i

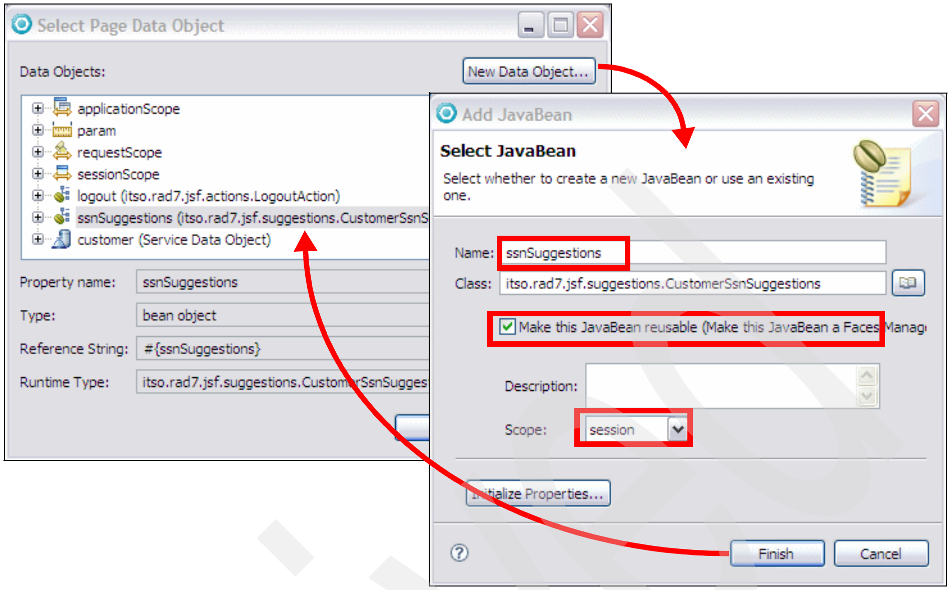
Height of the suggestion area:

Maximum number of suggestions displayed:

While suggestions are being retrieved: <defau

Figure 14-82 Select type-ahead component in the Properties view

- In the Select Page Data Object dialog, click **New Data Object** (Figure 14-83).
  - In the New Data Component dialog, select **Page Bean** → **JavaBean** and click **OK**.
  - In the Add JavaBean dialog:
    - Enter **ssnSuggestions** for the Name.
    - Locate the `CustomerSsnSuggestions` class.
    - Select **Make this JavaBean reusable**.
    - Select **session** scope.
    - Click **Finish**.
  - In the Select Page Data Object dialog, select **ssnSuggestions** and click **OK**.



Data Objects:

- applicationScope
- param
- requestScope
- sessionScope
- logout (itso.rad7.jsf.actions.LogoutAction)
- ssnSuggestions (itso.rad7.jsf.suggestions.CustomerSsnS
- customer (Service Data Object)

Property name: ssnSuggestions

Type: bean object

Reference String: #{ssnSuggestions}

Runtime Type: itso.rad7.jsf.suggestions.CustomerSsnSugges

Select JavaBean

Select whether to create a new JavaBean or use an existing one.

Name: ssnSuggestions

Class: itso.rad7.jsf.suggestions.CustomerSsnSuggestions

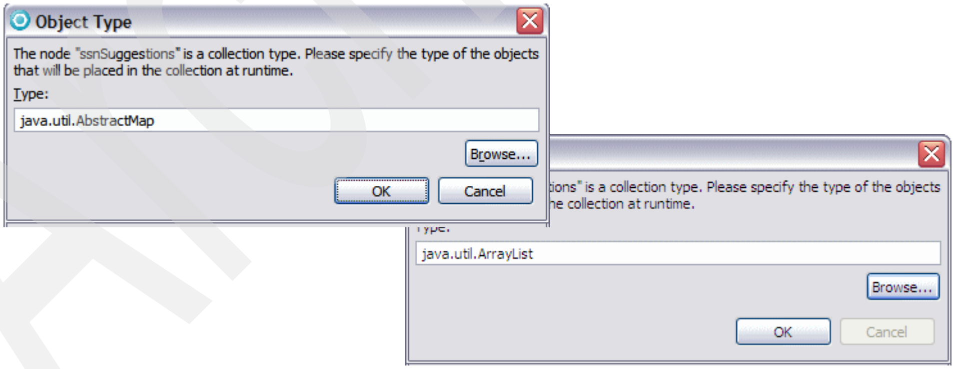
☑ Make this JavaBean reusable (Make this JavaBean a Faces Manag

Description:

Scope: session

Figure 14-83 Add and bind the suggestions JavaBean

- In the Object Type dialog, click **Browse**, locate `java.util.AbstractMap` and click **OK** (Figure 14-84).
- In the second Object Type dialog, locate `java.util.ArrayList` and click **OK**.



The node "ssnSuggestions" is a collection type. Please specify the type of the objects that will be placed in the collection at runtime.

Type: java.util.AbstractMap

tions" is a collection type. Please specify the type of the objects
he collection at runtime.

Type: java.util.ArrayList

Figure 14-84 Select contained object types

- The `CustomerSsnSuggestions` bean is now bound to the type-ahead component with a value of **#{ssnSuggestions}**. The suggestions bean is added to the Page Data view (Figure 14-85).

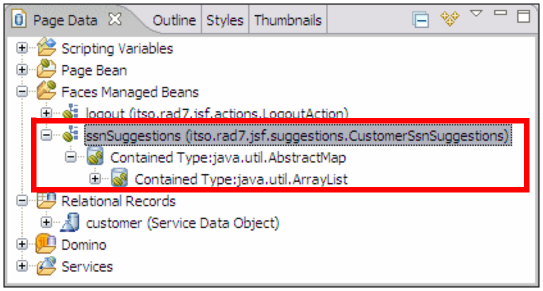

Page Data
Outline
Styles
Thumbnails
Scripting Variables
Page Bean
Faces Managed Beans
logout (itso.rad7.jsf.actions.LogoutAction)
ssnSuggestions (itso.rad7.jsf.suggestions.CustomerSsnSuggestions)
Contained Type:java.util.AbstractMap
Contained Type:java.util.ArrayList
Relational Records
customer (Service Data Object)
Domino
Services

Figure 14-85 page Data view with suggestions bean

Adding AJAX refresh submit behavior

AJAX refresh submit behavior defines alternative content for the panel, which can be asynchronously retrieved after the page has been loaded into the browser without refreshing the whole page.

Whenever an action is triggered (for example, a button click), the client requests the alternative content for the panel and it replaces the existing panel content with new content. The page containing the panel is not replaced by the get, instead, this tag allows part of the page to be replaced. The revised content is retrieved from the same JSP from which the original content came.

Both the server life cycle copy of the page and the client-side page are kept in sync. The new page content is retrieved using a post HTTP request operation. The contents of the form containing the panel are posted as part of the request so that the values in the form are available to the server code calculating the new content to put in the panel.

Create an alternate login page

For our sample, we create an alternative login page (`RAD7JSFLoginAJAX.jsp`) that is AJAX enabled. As long as a customer SSN is selected from the combo-box, the customer name is returned from the server without refreshing the page.

- In the Project Explorer view, right-click **WebContent**, and select **New** → **Web Page**.
- In the New Web Page dialog, do these steps, and click **Finish**:
  - File Name: `RAD7JSFLoginAJAX`
  - Folder: `/RAD7JSF/WebContent` (default)
  - Template: Select **My Templates** → **RAD7JSFTemplate**
- In the Dynamic Page Template pop-up click **Yes** to configure the content area. Select **bodyarea** from the content area list, select **New or existing fragment file**, and enter `RAD7JSFLoginAJAXFragment.jsp` in the Name field.

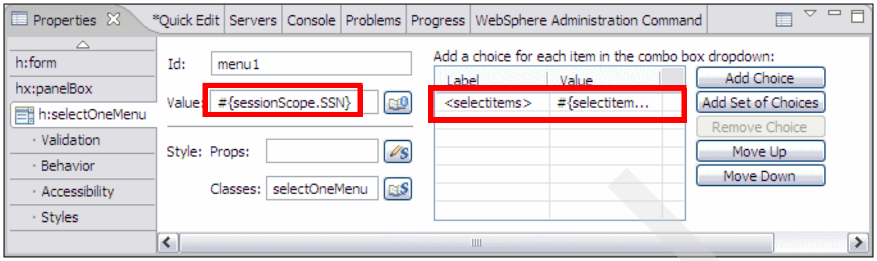
- Save and close the `RAD7JSFLoginAJAX.jsp`, then open the **RAD7JSFLoginAJAXFragment.jsp**.
- Select and delete the default text.
- Select **Insert** → **Paragraph** → **Heading 3**, then type the value **AJAX Enabled Login**. In the Properties view, change the font to **Arial**.
- In the Palette, expand the **Enhanced Faces Components**.
- Select **Panel - Group Box** and drop it under the heading. Select the box1 and in the Properties view, enter **5** for Padding.
- In the Palette, select **Output** and drag it into the panel box.
- In the Palette, select **Combo Box** and drag it into the panel box.
- In the Palette, select **Output** and drag it into the panel box.
- Arrange the three components so that the combo box is in the middle.
- Select the first `outputText` component, and in the Properties view, enter the text `Select Customer SSN:` into the Value field. Change the font to **Arial**.
- In the Palette, select the **Button - Command** and drag on the page into a new line below the table. Leave the label as Submit.

Create a relational record list to retrieve all customers

We want to fill a drop-down list with the customer SSNs:

- In the Page Data view, right-click **Relational Record** and select **New** → **Relational Record List**.
- In the Add Relational Record List dialog, enter **allCustomers** in the Name field and click **Next**.
- Select the **ITSO.CUSTOMER** table and click **Next**.
- Click **Order results** and select **SSN**, ascending.
- Click **Finish**.
- We have to bind the value of the combo-box so that it lists all customer SSN from the relational record list. To bind the value of combo-box do these steps:
  - In the Page Data view, expand **Relational Record** → **allCustomers (Service Data Object)** → **allCustomers (CUSTOMER)**.
  - Select **SSN (String)** and drag it on the combo-box. The help-tip shows the text *Drop here to bind “SSN” to the control “menu1”*.
  - In the Properties view, click **Browse** next to the Value field, then select **sessionScope** → **SSN** and click **OK**.

  The combo-box is bound to the customer SSN list and the selected customer SSN number is stored in `sessionScope` (Figure 14-86).



Id: menu1

Value: #{sessionScope.SSN}

Style: Props:

Classes: selectOneMenu

Add a choice for each item in the combo box dropdown:

Label	Value
<selectitems>	#{selectitem...

Figure 14-86 Properties view of the combo box

Display the customer name when an SSN is selected

When a customer SSN is selected from the combo-box, an AJAX submit request is triggered and the customer name is displayed. To fetch the customer name, we have to add a method to the page code (`RAD7JSFLoginAJAXFragment.java`).

- Right-click in the JSP editor and select **Edit Page Code**. This action opens the `pagecode.tilesContent.RAD7JSFLoginAJAXFragment.java` file.
- Insert the variables and the `getName` method from Example 14-6 into the code (copy/paste from `c:\7501code\jsfsdo\RAD7JSFLoginAJAXFragment.jpage`).

Example 14-6 Update for the AJAX fragment page code

```
import java.util.Iterator;
import java.util.Map;

   protected Map sessionScope;
   private static final String CUSTOMERSSN_KEY = "SSN";
   private static final String SESSION_SCOPE = "#{sessionScope}";

   public String getName() {
      String name = "";
      sessionScope = (Map) FacesContext.getCurrentInstance().getApplication()
            .createValueBinding(SESSION_SCOPE).getValue(
                  FacesContext.getCurrentInstance());

      if (sessionScope.containsKey(CUSTOMERSSN_KEY)) {
         getAllCustomers();
         Iterator iter = allCustomers.iterator();
         while (iter.hasNext()) {
            DataObject element = (DataObject) iter.next();
            if (element.getString("SSN")
                           .equals(sessionScope.get(CUSTOMERSSN_KEY))) {
               name = "Hello " + element.getString("FIRSTNAME") + " "
               + element.getString("LASTNAME") + "! Click Submit to login...";
            }
         }
```

```
        }
        return name;
    }
```

- Press `Ctrl+Shift+O` (Organize Imports) to add the necessary imports.
- Save and close the file.
- We have to configure the output component in the table to display the customer name:
  - In Design tab, select the output component on the right.
  - In the Properties view, click **Browse** next to the Value field, select the **name** object, and click **OK**.

The output component is now bound to the `getName` method to display the name of the selected customer (Figure 14-87).

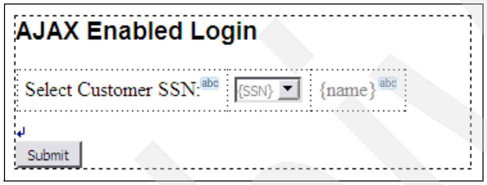
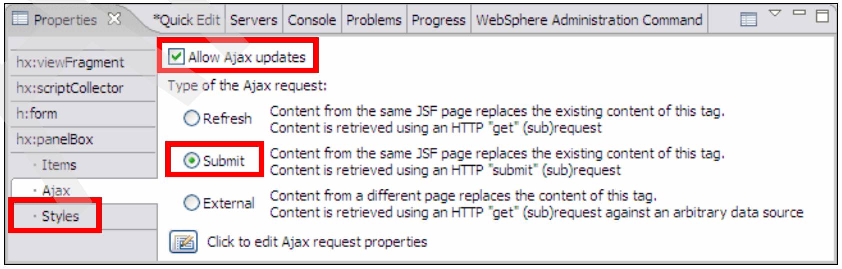


Figure 14-87 Layout of the AJAX enable login page

Add AJAX behavior to the panel

To add AJAX behavior to the panel, do these steps:

- Select the panel box or a component inside the panel box, and locate the **hx:panelBox** in the Properties view. (You can also select **hx:panelBox** from the drop-down menu on the top right of the editor.)
- In the Properties view, select **AJAX** (under `hx:panelGroup`), then select **Allow AJAX Updates**, and select **Submit** (Figure 14-88).



☑ Allow Ajax updates

Type of the Ajax request:

○ Refresh — Content from the same JSF page replaces the existing content of this tag. Content is retrieved using an HTTP "get" (sub)request

● Submit — Content from the same JSF page replaces the existing content of this tag. Content is retrieved using an HTTP "submit" (sub)request

○ External — Content from a different page replaces the content of this tag. Content is retrieved using an HTTP "get" (sub)request against an arbitrary data source

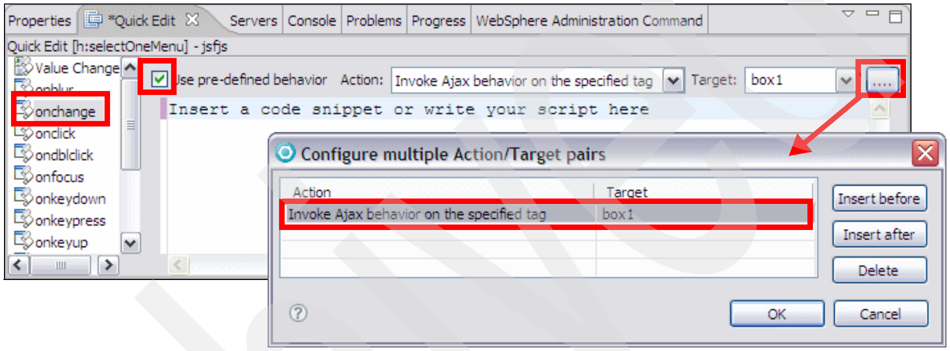
Click to edit Ajax request properties

Figure 14-88 Add AJAX behavior to the panel

Add AJAX behavior to the combo-box

To add AJAX behavior to the combo-box, do these steps:

- Select the combo-box component in the Design view.
- Select the **Quick Edit** view (Figure 14-89:
  - Select **onchange** in the left pane. Select **Use pre-defined behavior** and click the .... button.
  - In the Configure multiple Action/Target pairs dialog, select **Invoke Ajax behavior on the specified tag** for the Action field and **box1** for the Target. Click **OK**.



☑ Use pre-defined behavior Action: Invoke Ajax behavior on the specified tag Target: box1 ....

```
Insert a code snippet or write your script here
```

Action	Target
Invoke Ajax behavior on the specified tag	box1

Figure 14-89 Add AJAX behavior to the combo-box

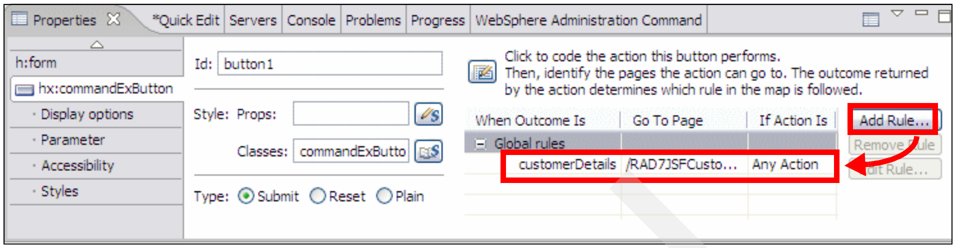
- Save the `RAD7JSFLoginAJAXFragment.jsp`

Complete the navigation

When the Submit button is clicked we want to invoke the customer details page. We do add this navigation logic in the Web Diagram, but we use a different approach for illustration purposes:

- In the Design tab select the **Submit** button.
- In the Properties view (Figure 14-90), click **Add Rule** to add a navigation rule. In the Add Navigation Rule dialog:
  - Select **RAD7JSFCustomerDetails** as the page.
  - Select **The outcome named** and enter **customerDetails**.
  - Select **All pages** and **Any action**.

  A rule for any page is a global rule.



Id: button1

Style: Props:

Classes: commandExButto

Type: ● Submit ○ Reset ○ Plain

Click to code the action this button performs. Then, identify the pages the action can go to. The outcome returned by the action determines which rule in the map is followed.

When Outcome Is	Go To Page	If Action Is
Global rules		
customerDetails	/RAD7JSFCusto...	Any Action

Figure 14-90 Adding a navigation rule from the Properties view

- ► In the Quick Edit view, select **Command** on the left side (preselected), and click into the code section. Replace the return statement with:

```
return "customerDetails";
```

- ► Save and close the `RAD7JSFLoginAJAXFragment.jsp`.

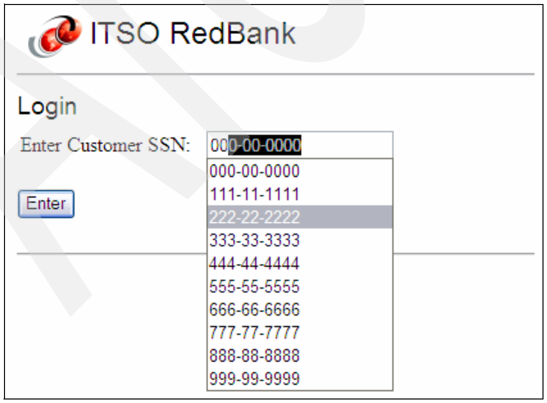
Testing the AJAX enabled JSF pages

To test the AJAX enabled application start the server.

Test the login page with suggestions

To test the login page with suggestions, do these steps:

- ► In the Project Explorer, expand **RAD7JSF** → **WebContent**.
- ► Right-click **RAD7JSFLogin.jsp**, and select **Run As** → **Run on Server**.
- ► In the login page, enter anything into the customer SSN field. As soon as you stop typing, the suggestion list opens with customer SSNs returned from the suggestions bean (Figure 14-91).



Login

Enter Customer SSN: 000-00-0000

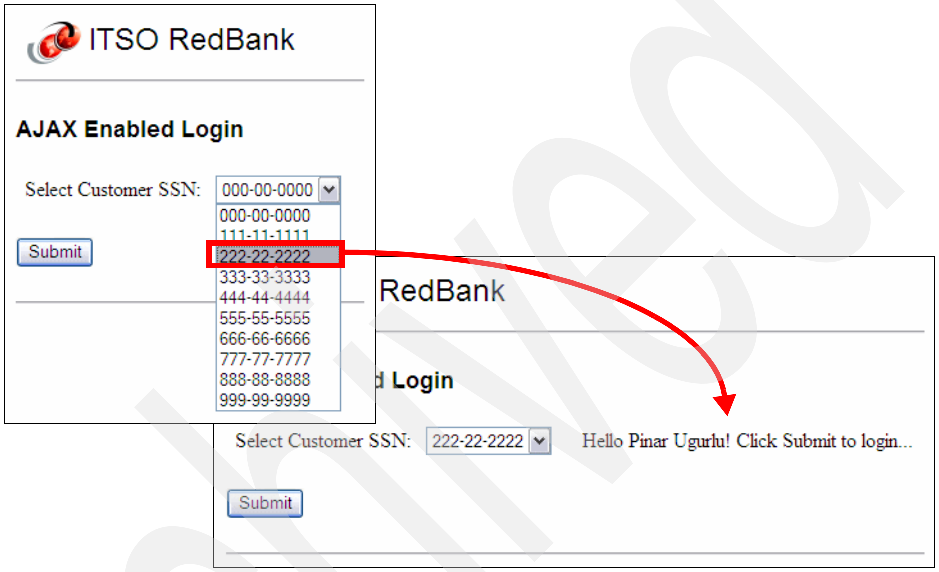
000-00-0000
111-11-1111
222-22-2222
333-33-3333
444-44-4444
555-55-5555
666-66-6666
777-77-7777
888-88-8888
999-99-9999

Figure 14-91 Ajax enabled login page with suggestions

Test the AJAX enabled login page

To test the new login page with AJAX refresh, do these steps:

- Right-click **RAD7JSFLoginAJAX.jsp**, and select **Run As** → **Run on Server**.
- The new AJAX enabled login page is displayed (Figure 14-92).
- Select a customer SSN from the combo-box and the name is returned from server without refreshing the page.



ITSO RedBank

AJAX Enabled Login

Select Customer SSN: 000-00-0000

Select Customer SSN: 222-22-2222

Hello Pinar Ugurlu! Click Submit to login...

Figure 14-92 Login page with Ajax enabled refresh

Possible improvements

The performance could be improved in a number of ways:

- Retrieve only SSNs in the `allCustomers` relational record list.
- Use a relational record to retrieve the name of the customer (reuse the customer relational record for that purpose).
- Store the customer names in session data for faster retrieval.

Developing a JSF page with a Web service call

Refer to "Creating a Web service JSF client" on page 835 for an example.

More information on JSF, SDO, and AJAX

For more information on JSF, SDO, and AJAX, we recommend the following resources.

- *Improve the usability of Web applications with type-ahead input fields using JSF, AJAX, and Web services in Rational Application Developer V7*, article found at:

  http://www.ibm.com/developerworks/rational/library/06/1205_kats_rad1/index.html

- *Advanced usage of the Typeahead control in Rational Application Developer V7.0*, article found at:

  http://www.ibm.com/developerworks/rational/library/07/0206_bermingham/index.html

- *JSF and Ajax: Web 2.0 application made easy with Rational Application Developer V7*, article found at:

  http://www.ibm.com/developerworks/rational/library/06/1205_kats_rad2/index.html

- *Creating a sorted JavaServer Faces Widget Library dataTable using Rational Application Developer V7.0*, article found at:

  http://www.ibm.com/developerworks/rational/library/07/0220_gallagher/index.html

- *New features of the JavaServer Faces Widget Library dataTable component in IBM Rational Application Developer 7.0*, article found at:

  http://www.ibm.com/developerworks/rational/library/07/0213_gallagher/index.html

- *Using the ProgressBar JSF Component in Rational Application Developer*, article found at:

  http://www.ibm.com/developerworks/rational/library/07/0626_kats/

- *Improve the look and feel of your Web pages by using the Dynamic Page Template*, article found at:

  http://www.ibm.com/developerworks/rational/library/07/0508_koinuma/index.html

15

Develop applications using EGL

IBM Enterprise Generation Language (EGL) is a fourth generation business application programming and run time environment.

The primary justification for the use of EGL (the basic value proposition of EGL), is programmer productivity; EGL is easier to learn, use, and maintain than C/C++, Java/J2SE, or Java/J2EE, for example. EGL is built for the business application developer, with many wizards, language constructs, and tools to support rapid application development. While EGL is procedural and declarative in nature, EGL also provides many object-oriented programming features, such as encapsulation, function overloading, user defined data types, events based programming, and abstraction. You can start small, and adopt object-oriented programming type capabilities as you advance.

EGL is an optionally installed component of Rational Application Developer and other products. Because the larger topic of this Redbooks publication is the Rational Application Developer, this chapter devoted to EGL has to be brief, and demonstrates creating and running two examples; an EGL batch program, and a simple two-page Web site. For a more complete discussion related to EGL, refer to *Transitioning: Informix 4GL to Enterprise Generation Language (EGL)*, SG24-6673, as well as other EGL related documents and sites.

Introduction to EGL

With EGL, you have the following capabilities:

- You can create and deliver batch programs, character based programs, graphical two-tier programs, Web sites (inter-connected Web pages).
- You can consume and provide Web services, output character and graphical printed reports.
- You can interact with numerous and concurrent disparate data sources: relational databases, non-relational databases, IBM MQ-Series data sources, and others.
- You can run EGL programs on Linux, UNIX®, Microsoft Windows (LUW), IBM iSeries, and IBM zSeries, usually, although not exclusively, hosted inside a Java/J2EE compliant Application Server or inside a Java Virtual Machine (JVM).
- Using the industry standard design pattern of model-view-controller (MVC), you can get EGL code re-use across all of the above program types, data sources types, and run time platforms, and all from one skill set, that of EGL programmer.
- There are an increasing number of migration toolkits/utilities that can consume program code from other environments and output EGL, allowing you to webify and preserve program code from legacy and other programming languages and platforms.

EGL key terms and ideas

Enterprise Generation Language (EGL) is an optionally installed component to Rational Application Developer, which is in turn based on the Eclipse open source developer's workbench. Terms and ideas that originate in Eclipse, flow up into Application Developer, which then in turn flow into EGL. The key terms and ideas that EGL inherits from Eclipse and Rational Application Developer include:

- **(Developer's) workbench**: In the context of this book, the developer's workbench is Rational Application Developer.
- **Workspace**: In the simplest context, a workspace is the parent directory under which projects and program source code are located. Each project creates a new (sub) directory under the workspace (parent directory). Given a workspace of `C:/MyWorkspaces/Workspace01/`, and a project entitled `ITSOBank`, all source code for this project is located under `C:/MyWorkspaces/Workspace01/ITSOBank/`.

- **Project and project types**: By design, everything you do in Eclipse has to be part of a project. A project is a grouping of source code, including metadata related to the compilation and deployment of that source code.
  - Eclipse has a small number of project types, and Rational Application Developer adds to that list. EGL itself adds two project types; EGL Web projects, and EGL (non-Web) projects.
  - **EGL Web projects** automatically output Java/J2EE code, which is then deployed and executed within a Java/J2EE compliant application server. You do not have to know anything about Java to benefit from EGL. By standard practice, J2EE application servers provide a readily available SQL database connection pool. As a result, *EGL Web (programs) do not have to open a database connection*, as one is already provided by the J2EE application server.
  - **EGL (non-Web) projects** automatically output Java/J2SE or COBOL code. The COBOL code runs on the IBM iSeries or zSeries platform. The Java/J2SE code runs inside a Java virtual machine (JVM). *EGL (non-Web programs), must open an explicit database connection* when accessing a database, because none is automatically provided by the JVM.
- **Views and perspectives**: An Eclipse view is a specific bounded region of the workbench display. Views are similar to HTML frames, or what other graphical displays call panels. There are so many views in Eclipse, Rational Application Developer, and EGL, that Eclipse offers groupings of views called perspectives. You can open any perspective or view through the menu bar and selecting **Window** → **Open Perspective** or **Window** → **Show View**.

  There are times that you want to have a specific set of two, three, or more views open, so that you may, for example, drag and drop artifacts from one view to another. In other cases, one view acts as a (canvas), another a palette, and a third to examine or set properties related to what you have painted.

  While there is an EGL perspective, most persons work in the Web perspective when creating EGL Web programs, and then maybe open one or two more views to support their EGL non-Web activities.

Key terms and ideas that are specific to EGL include these:

- **Data items**: EGL has constants and variables like any programming language. Additionally EGL has data items, which in the simplest case are variables with additional properties. For example, a data item can extend an integer to include a default value, the range or list of acceptable values, and more. There are 30 or more properties that data items can contain, each with a distinct purpose.

  The advantage towards using data items is code re-use. You can program a given variable's behavior one time, and then re-use it wherever necessary.

- **Records, record types**: Most programming languages have the concept of records, generally that being a collection of variables. The advantage towards using records is that you can pass or operate on a collection of variables while creating and maintaining less program source code.

  Just as you can set properties on EGL data items, you can set properties on EGL records. You can set properties on the record as a whole, and you can set properties on the individual elements of a record.

  EGL has eight or more record types. Its default record type is BasicRecord, which offers little more than a means to group constants, variables, and/or data items. In contrast, an EGL SQLRecord allows you to set properties that specify the source SQL table or tables, including any SQL join criteria or filter criteria, primary key columns, and more. Once an EGL SQLRecord has these properties, getting a single or set of data rows is as easy as saying, *get Customer*, where `Customer` is the EGL SQLRecord name.

  EGL offers a set of abstract data access command verbs, so *get* works for SQL data sources, non-SQL data sources, IBM MQ-Series, and more. Once the EGL record has its associated data source properties set, accessing the given data source is automatic.

- **Generatable (primary) parts**; An EGL program may be contained inside one or more ASCII text source code files with a **.egl** file name suffix. These EGL source code files are called parts; for now, we say that one file equals one part, in all cases.

  Generatable parts have an EGL language declarative statement that indicates the file's purpose within an application. For example, an EGL *library part* generally contains non-visual code, either database access routines, or some collection of business rules. EGL library parts are meant to contain re-usable code, and are meant to be consumed by some other larger entity. An EGL *JSFHandler* part is the end user initiated event handler for a given Web page.

  The root word *generate*, in the term generatable part, means that the given file generates Java/J2SE, Java/J2EE, or COBOL code, whatever is required. EGL generatable parts are also called primary parts, and these two terms are synonyms in all cases.

  A given EGL source file is only one type of generatable (primary) part; meaning that a single EGL source file cannot be both a library and a JSFHandler. To have both a library and a JSFHandler, you must use two files.

  Here we describe the EGL generatable parts used in this chapter:

  – **Library part**: Can be contained in either EGL Web projects, or EGL (non-Web) projects. Libraries are expected to contain non-visual code, either database access routines, or some business rule. Libraries are meant to contain re-usable code, and are meant to be consumed by some other larger entity.

   - **Program part**: Are expected to be contained in EGL (non-Web) projects, and are used by batch jobs, character user interface programs, graphical user interface programs, and printed reports. Programs contain a main function which indicates the first function to be executed and begin a program.
   - **JSFHandler part**: Are expected to be contained in EGL Web projects. A JSFHandler is the end user initiated event handler for a given Web page. Every Web page has one and only one JSFHandler, and by convention, both of these entities share the same name.
   - Other part types, not listed here: There are many other EGL generatable part types, including those for Web services, which are beyond the scope of the larger topic of this chapter.

- **Non-generatable (non-primary) parts**; these include EGL constants, variables, data items, and records.

  It is stated above that *generatable parts* are used to generate (output) Java/J2SE, Java/J2EE, or COBOL as needed. Generatable parts are found one to a file. *Non-generatable parts* only output Java or COBOL when used inside a generatable part. This can be explained, because a variable definition without the (EGL) program code to assign values and evaluate its content, does not have to output anything; it is non-functioning.

  Non-generatable parts can be found one, dozens, or hundreds to a file. You can have an EGL source code file with just non-generatable parts, sort of a catalog of variables with meta data and business rules (data items). And you can have non-generatable parts embedded inside a generatable part (for example, data items), or defined inside a library part.

- **Packages**: Are essentially sub-directories. Under a given EGL project exist two special package sub-directories:
   - `EGLSource`–Contains every EGL source file for a given project.
   - `WebContent`–Contains every piece of content for a given EGL Web project. Examples of content include files of type HTML, XML, XSD, XSLT, cascading style sheets (CSS), GIF files, JPG files, and JavaScript.

Overview and dependencies of examples in this chapter

In the remainder of this chapter, we create an EGL batch program, and a simple two-page Web site. To demonstrate code re-use, the Web site uses EGL program code that was automatically generated for the EGL batch program. Further, it is stated:

- EGL Web projects automatically output Java/J2EE code, which is then deployed and executed within a Java/J2EE compliant application server. We require this application server to host (serve) our two page Web site.

  We create one EGL Web project, and configure this project to use a SQL database connection that is maintained by application server.

- Batch programs should probably operate outside of an application server, and that is the type of program we create. To operate (programs) outside of an application server, we create an EGL (non-Web) project. This project should be configured with its own independent SQL database connection.

- Because both the two page Web site and batch program access a database (and in our actual case, the same `ITSOBANK` database), we choose to share EGL source code between these two EGL projects.

  We generate the shared code using a wizard. While this and most code could be located in either project, we place this code in the EGL (non-Web) project.

  Best practice might actually suggest having three projects; one for just shared code, one for just (non-shared) batch program code, and one for the two page Web site code.

- While our two examples only *read* data so that we can keep this chapter brief, you are fully able to extend these examples and perform SQL *writes* and other, more advanced operations, should you desire to. We provide short instructions for a Web page with updates.

- The examples in this chapter connect to the `ITSOBANK` sample database, used throughout this Redbooks publication. At times in this book, the `ITSOBANK` database is used in either the IBM DB2 UDB database server, and/or the open source embedded Derby database server. While EGL can use many different and concurrent database servers, this chapter uses Derby.

  The embedded version of the Derby database server is a single user database server. At any time, only one user, one program, can access this database server. (The network version of Derby is multi-user.)

  In the examples that follow in this chapter, the `ITSOBANK` database is located in the directory `C:\7501code\database\derby\ITSOBANK\`.

EGL installation

EGL ships as a package called **IBM Rational Business Developer Extension** and is installed using the IBM Installation Manager into the same Eclipse shell as Rational Application Developer.

Refer to “Installing Enterprise Generation Language” on page 1295.

EGL example 1: Simple batch program

In this section, we are primarily creating two EGL source code files: One file generated, the other largely done by hand to save instructional steps. This work is done in a new EGL (non-Web) project, which is configured to access the `ITSOBANK` database, hosted in the open source database server, Derby. As the net result of our activities, we produce a batch program that reads and outputs data from a SQL data source. As a batch program, our program executes with no expected user interface, no keyboard input, and only minimal diagnostic output to the console.

Rational Application Developer workspace

- Start Rational Application Developer.

> **Tip:** To follow this chapter, we suggest that you use a new workspace. However, this is not required, and you can certainly use the same workspace used for the other chapters.
>
> You can select a new workspace when you are prompted after starting Application Developer, or you can switch to a new workspace by selecting **File** → **Switch Workspace** from the menu bar.
>
> Close the Welcome panel that is displayed for a new workspace.

- Enable the EGL development capability by selecting **Window** → **Preferences**:
  - In the Preferences dialog expand **General** → **Capabilities**.
  - Select **EGL Developer** and click **Advanced**. make sure that **EGL Core Language** and **EGL JSF** are selected.
  - Click **OK**.

Creating an EGL (non-Web) project

We start by creating a new EGL (non-Web) project.

- We work in the EGL perspective for this sample. Select **Window** → **Open Perspective** → **Other**. In the Open Perspective dialog, select **EGL** and click **OK**.
- From the menu bar, select **File** → **New** → **Project**.
- In the New Project dialog, select **EGL** → **EGL Project** and click **Next**.

- In the New EGL Project dialog, enter **`RAD7EGLnonWeb`** for the Project name. Select **Java** for Target Runtime Platform (default). and click **Finish**.

  `RAD7EGLnonWeb` is the first of two EGL projects created in this chapter. (We use a naming convention of `RAD7Xxxxx` for all projects created.)

- The EGL (non-Web) project is now created.

> **Note:** Most EGL developers work in the Web perspective. The Project Explorer view offers a hierarchical display of entities inside each project, and is the view most commonly used to manage EGL projects.

Creating a database connection resource

Generally, every entity that is created within Eclipse belongs to a project, however, there are a small number of entities that reside in the workbench proper. These resources are shared and available across projects. One such entity is a database connection resource. Database connection resources are used by projects and various code generation wizards as a means to promote definition and code re-use.

- Select **Window** → **Show View** → **Other** → **Data** → **Database Explorer** and click **OK**.

  This action opens the Database Explorer view, typically in the bottom center of the workbench.

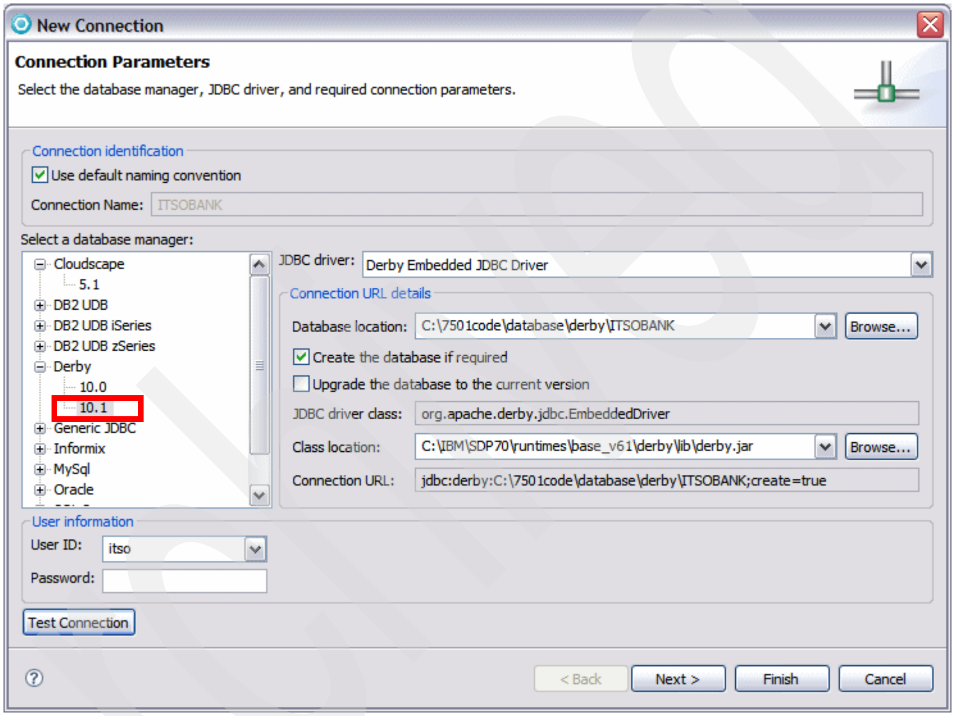
  The Database Explorer view operates much like a tree, or hierarchical representation of the objects it contains. Each entry at the *root level* of the hierarchy represents a database connection definition. If the connection is open, the entry also represents a database session.

- In the Database Explorer view, right-click **Connections** and select **New Connection**.
- In the New Connection dialog (Figure 15-1) perform the following steps:
  - For Select a database manager, select **Derby** → **10.1**.
  - For database location, click **Browse** and navigate to:

    `C:/7501code/database/derby/ITSOBANK`

    This directory path name is the standard value used in this book. This value may differ to equal your actual directory path name.
  - For Class location, click **Browse** to locate the single or set of JAR files which form the JDBC driver for the given database server vendor. For the Derby database server embedded in Rational Application Developer, navigate to `C:\<RAD_HOME>\runtimes\base_v61\derby\lib\derby.jar`.

- The embedded version of Derby does not require a user ID or password because it runs inside your current process environment. However, this dialog expects a user ID. Enter any name, for example, `itso`.
- Note that the connection name is filled as `ITSOBANK`, as a value derived from the database name.
- Click **Test Connection** and correct any errors.
- Click **Finish**.


New Connection

Connection Parameters

Select the database manager, JDBC driver, and required connection parameters.

Connection identification

☑ Use default naming convention

Connection Name: ITSOBANK

Select a database manager:

JDBC driver: Derby Embedded JDBC Driver

Connection URL details

Database location: C:\7501code\database\derby\ITSOBANK

☑ Create the database if required

☐ Upgrade the database to the current version

JDBC driver class: org.apache.derby.jdbc.EmbeddedDriver

Class location: C:\IBM\SDP70\runtimes\base_v61\derby\lib\derby.jar

Connection URL: jdbc:derby:C:\7501code\database\derby\ITSOBANK;create=true

User information

User ID: itso

Password:

Figure 15-1 New Connection dialog

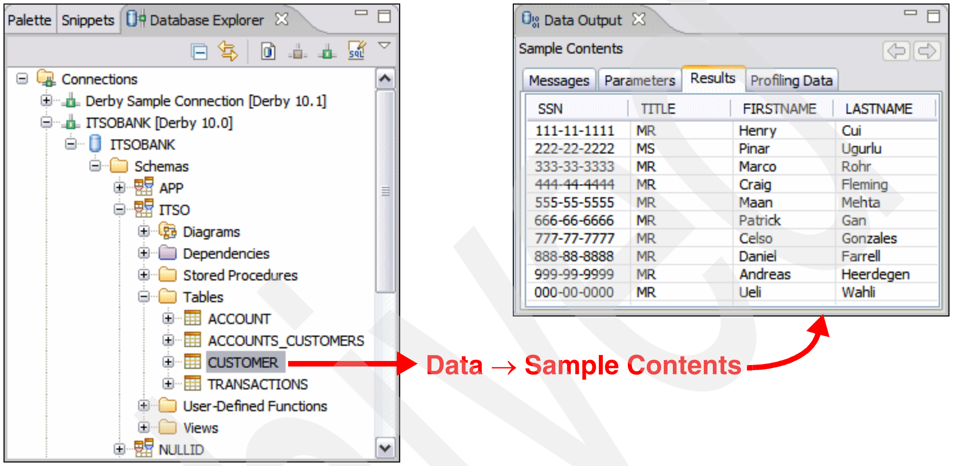
Tip: You might already have this connection from Chapter 9, “Develop database applications” on page 355. You can reuse that connection or create a new connection: Clear **Use default naming convention** and enter a connection name.

Testing a database connection

As a means to further test and explore the database connection resource, use the Database Explorer view:

- In the Database Explorer view expand **Connections** → **ITSOBANK [Derby 10.1]** → **ITSOBANK** → **Schemas** → **ITSO** → **Tables**.

  If you cannot expand `ITSOBANK`, most likely the connection is not open (connected). Right-click **ITSOBANK** and select **Reconnect**.
- Right-click **Customer** and select **Data** → **Sample Contents**.
- This action produces the Data Output view (Figure 15-2).



SSN	TITLE	FIRSTNAME	LASTNAME
111-11-1111	MR	Henry	Cui
222-22-2222	MS	Pinar	Ugurlu
333-33-3333	MR	Marco	Rohr
444-44-4444	MR	Craig	Fleming
555-55-5555	MR	Maan	Mehta
666-66-6666	MR	Patrick	Gan
777-77-7777	MR	Celso	Gonzales
888-88-8888	MR	Daniel	Farrell
999-99-9999	MR	Andreas	Heerdegen
000-00-0000	MR	Ueli	Wahli

Data → Sample Contents

Figure 15-2 Testing the Database Connection Resource, Database Explorer view.

- Before leaving the Database Explorer view, release this (single user) connection to Derby by right-clicking **ITSOBANK [Derby 10.1]** and selecting **Disconnect**.

  At times, you may want to disconnect and reconnect from this database connection resource by using this same context menu.

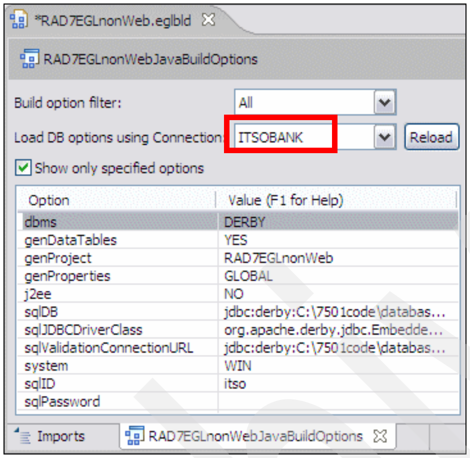
Adding SQL database support to the EGL (non-Web) project

At this point in the example, we have created a new EGL (non-Web) project, and created a new database connection resource. Now we want to make this database connection resource available to the EGL (non-Web) project. All of this is accessible through the Project Explorer, with modification to the EGL build file, and modification to the project properties:

- In the Project Explorer, locate the `RAD7EGLnonWeb` project.
- Expand **RAD7EGLnonWeb** → **EGLSource**, and locate the build file `RAD7EGLnonWeb.elgbld`.

This is the default EGL build file for this project, and contains settings related as to how this project is compiled and deployed. There can be multiple EGL build files allowing you to compile and/or deploy for numerous platforms or targets.

- Open (double-click) `RAD7EGLnonWeb.eglbld` in the EGL Build File editor.
- In the **Load DB options using Connection** drop-down list, select **ITSOBANK** (Figure 15-3).



Build option filter: All

Load DB options using Connection: ITSOBANK

☑ Show only specified options

Option	Value (F1 for Help)
dbms	DERBY
genDataTables	YES
genProject	RAD7EGLnonWeb
genProperties	GLOBAL
j2ee	NO
sqlDB	jdbc:derby:C:\7501code\databas...
sqlJDBCDriverClass	org.apache.derby.jdbc.Embedde...
sqlValidationConnectionURL	jdbc:derby:C:\7501code\databas...
system	WIN
sqlID	itso
sqlPassword	

Figure 15-3 EGL Build File editor view with modifications.

This definition comes from the database connection resource we created in the Database Explorer view. By selecting this database connection resource, it becomes the default for this project

- Save and close the EGL Build File editor.
- Right-click `RAD7EGLnonWeb` and select **Properties**. This action opens the Properties dialog.
- The left side of the Properties dialog displays many subject areas of project properties that can be modified. Specifically, we have to modify two of these areas:
  - On the left side of the dialog, select **Java Build Path**. This action displays four tabs on the right side of the dialog.
  - Select the **Libraries** tab (Figure 15-4).

    This where we can add external Java JAR files as compiled Java code to the project. These JAR files can be anything we picked up from a vendor, or any third party. At this time we are adding the JDBC drivers for Derby.

– Click **Add External JARs**. In the JAR Selection dialog locate the directory that contains the embedded Derby JDBC JAR files:

```
C:\<RAD_HOME>\runtimes\base_v61\derby\lib\derby.jar
```

– Click **Open** and the JAR file is added to the Libraries tab.

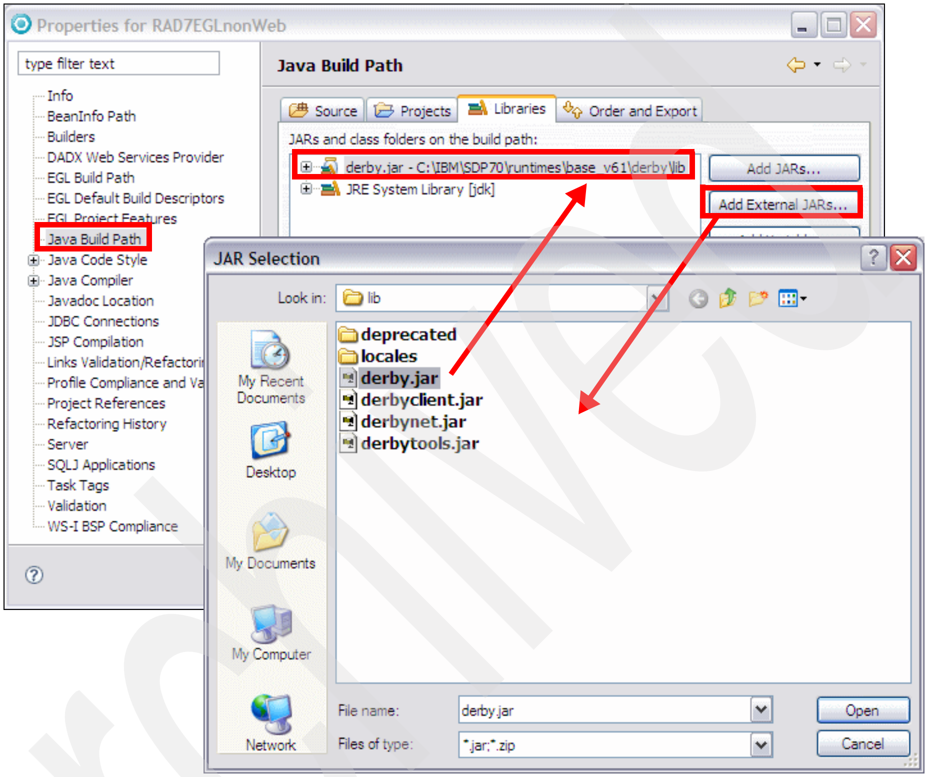


Figure 15-4 Project Properties: Extending the Java Build Path with Derby

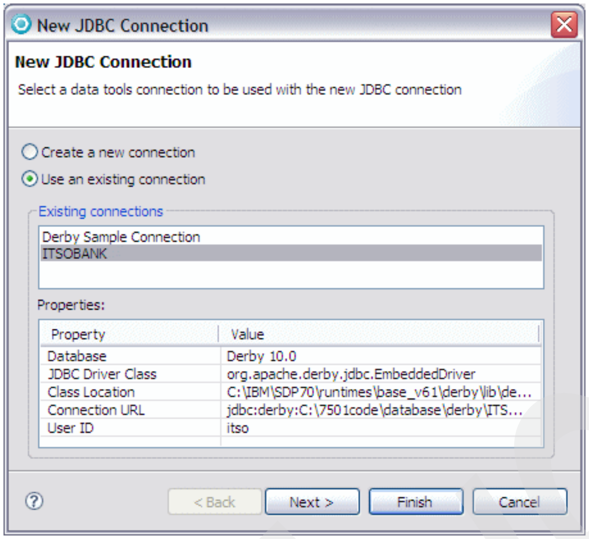
– On the left side of the Properties dialog, select **JDBC Connections**. When prompted to save the changes, click **Apply**.

– Click **New** and the New JDBC Connection dialog opens (Figure 15-5):

– Select **Use an existing connection** and select **ITSOBANK** from the list.

– Click **Finish** and the connection is added to the Properties dialog.

– Click **OK** to close the Properties dialog.



New JDBC Connection

New JDBC Connection

Select a data tools connection to be used with the new JDBC connection

○ Create a new connection

● Use an existing connection

Existing connections

Derby Sample Connection
ITSOBANK

Properties:

Property	Value
Database	Derby 10.0
JDBC Driver Class	org.apache.derby.jdbc.EmbeddedDriver
Class Location	C:\IBM\SDP70\runtimes\base_v61\derby\lib\de...
Connection URL	jdbc:derby:C:\7501code\database\derby\ITS...
User ID	itso

Figure 15-5 Project properties: JDBC connection

In the last few steps, we accomplished the following setup:

- Set `ITSOBANK` as our default JDBC connection. This default is associated with any new EGL program code.

  (Different EGL parts can use different JDBC connections, in effect, different databases.)

- Added the embedded Derby JDBC JAR files to the project.
- Set `ITSOBANK` as the default JDBC connection for any project wizards or code generators for this project.

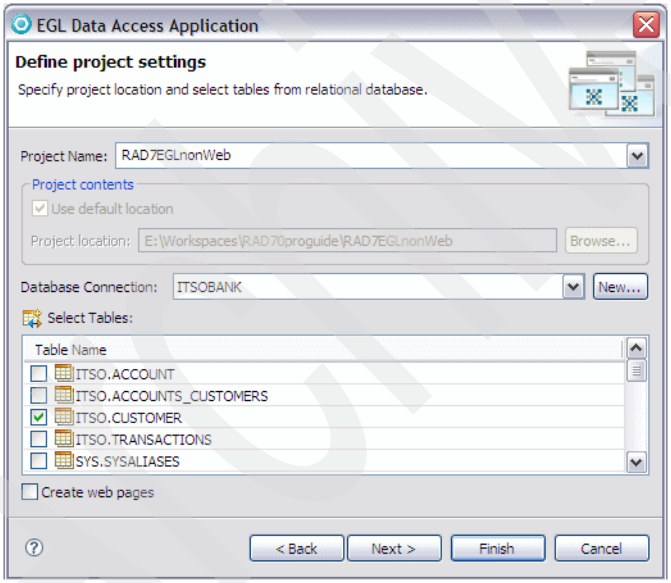
All of this prior activity was preparatory in nature towards defining our *run time environment.* Now we are ready to create EGL program code.

Generating EGL data items, records, SQL access libraries

EGL has numerous wizards to generate parts of a business application, and we use one of these wizards now. The EGL Data Access Application wizard generates EGL data items and SQL records, which we use in this application. The activity we are performing is referred to as *creating an EGL data access application.*

> **Note:** EGL data items and SQL records can be generated from an existing SQL database server, and many other logical and physical modeling sources. What is generated can be re-generated later while still preserving changes we make on the EGL side. This would allow, for example, a data model to change over time, while along the way we make enhancements to any EGL code that relies upon that model.

- Select **File** → **New** → **Other** → **EGL** → **EGL Data Access Application**, and click **Next**.
- The EGL Data Access Application dialog opens (Figure 15-6):
  - Select **RAD7EGLnonWeb** for the Project Name.
  - Select **ITSOBANK** for the Database Connection.
  - Select **ITSO.CUSTOMER** for Select Tables.


EGL Data Access Application
Define project settings
Specify project location and select tables from relational database.
Project contents
Browse...
New...
< Back
Next >
Finish
Cancel

Project Name: RAD7EGLnonWeb

☑ Use default location

Project location: E:\Workspaces\RAD70proguide\RAD7EGLnonWeb

Database Connection: ITSOBANK

Select Tables:

	Table Name
□	ITSO.ACCOUNT
□	ITSO.ACCOUNTS_CUSTOMERS
☑	ITSO.CUSTOMER
□	ITSO.TRANSACTIONS
□	SYS.SYSALIASES

□ Create web pages

Figure 15-6 New EGL Data Access Application dialog

- Click **Next**.
- In the EGL Data Access Application - Define the Fields dialog, leave all the defaults and click **Next**.
- In the EGL Data Access Application - Define Project Creation Options dialog:
  - For Default package name enter the value **`generated`**.

This is the EGL package name where the generated code is output. Any package name value would work here. The value `generated` is a style preference of using a name that indicates this package contains generated code.

– Select **Qualify table name with schema**.

> **Note:** In general, the hierarchy of objects in a SQL database contains; the **database server instance** → **database**(s) → **schemas** → **tables** → **rows and columns**. A SQL schema is a logical grouping of other, lower, SQL objects including SQL tables, SQL views, and more.
>
> *Some SQL database servers require presence of the schema identifier when accessing SQL tables, while other database servers do not*; in other words, is the SQL table name `ITSO.CUSTOMER` or just `CUSTOMER`. To make matters more complicated, SQL databases have the concept of a current schema, versus distinct (absolute path name to) schemas.
>
> The net result of this topic is that we are using fully (schema) qualified SQL table names.

- Click **Finish**.

This action produces several progress dialog boxes and eventually the New EGL Data Access Application dialog closes.

This action also produces a several EGL packages and EGL files under `EGLSource\generated`.

You can use the Project Explorer to open and examine these new entities. Close all of these files without change when you are done.

Creating the EGL simple batch program

Thus far we have configured the run time environment, and even generated EGL data items and SQL records. If we had just wanted to create a batch program that counted to ten and output a string of numbers, none of the prior work would have been required. All of the prior work was related to establishing SQL database connectivity, which allows us to create a batch program of some greater purpose.

EGL has numerous wizards, snippets (galleries of useful sample code fragments), content assist, and other tools to increase your productivity when creating EGL business applications. Because the EGL program we are about to create is very small (20 lines or less), we are just going to type most of it.

Note: If you want to experiment with content assist, do so. Content assist gives you the ability to create or modify lines of EGL program code by typing **Ctrl-Spacebar**. Content assist is context sensitive, and only prompts you with variable references, command verbs, and so on, that are appropriate for the location of the file (the context) you are in.

- In the Project Explorer, right-click **EGLSource** and select **New** → **EGL Package**. Enter the value **`MyPrograms`** and click **Finish**, and the new package appears under `EGLSource`.
- Right-click **MyPrograms** and select **New** → **Other** → **EGL** → **Program**. Click **Next**.
- In the New EGL Program Part dialog, enter **`SimpleBatch`** for EGL source file name. Click **Finish**.
- This action closes the New EGL Program Part dialog, and opens the EGL (source code) editor.

  EGL automatically populates this new EGL source file with source code that is appropriate for an EGL program. The EGL source code that is placed inside any newly created EGL part is something you can manage, and can add further productivity in the use of EGL.

Completing the simple batch program

While the EGL New Program Part dialog gave us some useful EGL code as a guide, we edit the program as shown in Example 15-1 (you can copy/paste from `C:\7501code\egl\SimpleBatch.egl`):

- Edit the source code contained in `SimpleBatch.egl` to what is displayed in Example 15-1. The line numbers displayed are for reference only; **do not enter the line numbers**.
- Use **Ctrl-S** to save the file.
- **Ctrl-G** (or right-click and select **Generate**) generates Java and compile. At times throughout this step, feel free to save your work and attempt to compile.
- You may also experiment with content assist, which is invoked with **Ctrl-Spacebar**.

Example 15-1 SimpleBatch.egl program listing (line numbers are for reference only)

```
package MyPrograms;

import generated.data.*;

program SimpleBatch type BasicProgram {}

```

```
008    function main()
009       customer   Customer;
010       str1       string  ;
011       str2       string  ;
012
013       str1 = "C:/7501code/database/derby/ITSOBANK";
014       str2 = "SELECT * FROM itso.customer ORDER BY 1";
015
016       try
017          SQLLib.connect("jdbc:derby:" + str1, "", "");
018          prepare s1 from str2;
019          open c1 with s1 for customer;
020          forEach (from c1)
021             SysLib.writeStdout("  " + customer.ssn +
022                "  " + customer.lastname );
023          end                         //  end to forEach
024       onException (e AnyException)
025          SysLib.writeStdout("Error: " + e.messageid);
026          SysLib.writeStdout("Error: " + e.message  );
027       end                            //  end to try
028    end                               //  end to Function
029
030 end                                  //  end to Program
```

As a language, EGL is not case sensitive. There are, however, identifiers which are exported to and from EGL that are case sensitive. For example, any variable names passed as part of an HTTP request header are case sensitive.

A line-by-line code review of `SimpleBatch.egl` follows:

- Line-002 includes an EGL package statement.

  The result of Line-002 states that this source code file must be located in `EGLSource/`**`MyPrograms`**`/`.

  EGL rigidly enforces filename and package references. As a result, you are more likely to be able to locate this source code at some later time.

- Line-004 is an EGL import statement.

  The result of this line is to allow reference to EGL source code and objects that are not located (defined) inside this file. Line-004 allows reference to the `Customer` object on Line-009. Highlight the word `Customer` on Line-009, right-click and select **Open on Selection** (or **F3**). With this action you open the EGL source code of `Customer` in the `generated.data` package.

- Line-006 declares that this file is an EGL primary part of type program.

  An EGL program is used for batch jobs, character user interface programs, graphical user interface programs, reports, and more.

Line-006 is paired with (terminated by) Line-030.

- Line-008 begins the definition of an EGL function entitled, `main`.

  Like many other procedural programming languages, EGL has a block (atomic unit) of code that may be invoked by name. The word `main`, means that this is the first function run by this EGL program.

  Line-008 is paired with (terminated by) Line-028.

- Line-009 to Line-011 define three variables that are local in scope to the function entitled `main`.

  `str1`, and `str2` are primitive variables of type string. String variables are of variable length and may contain any printable character.

  On Line-009, the variable name `customer` is of type `Customer`. Not recognizing `Customer` as a keyword inside EGL, we then know that `Customer` is a data item or EGL record.

  As a data item or EGL record, we can resolve the definition of `Customer` by highlighting the entire word and pressing **F3**, or selecting **Open on Selection**.

- Lines-013 and 014 assign values to the primitive variables.

  While JDBC is strong standard as a database connectivity API, the connection URL used to connect to a given database is highly variable by each database vendor. The connection URL for embedded Derby is essentially `jdbc:derby:`, and then the full path name to the database directory.

  Line-013 sets `str1` equal to the connection URL for Derby; essentially the full directory pathname to the database. This value may differ.

  Line-014 sets **str2** equal to an SQL SELECT statement.

- Line-016 begins a *try block* and is paired with (terminated by) Line-027.

  The basic mechanism for run time error recovery in EGL is the try block. In effect, if any run time errors are observed between Line-017 and 023, EGL automatically forwards program control to the first line following the `onException` clause (of the try block).

  On Line-024, we are catching any recoverable type of run time error, (`AnyException`). By convention, there can be several `onException` clauses with specific categories of run time errors.

  In short, EGL can catch run time errors locally, or *throw* them up to any calling function.

- Line-017 invokes a function from a built in EGL library.

  There are many built in libraries which are part of EGL. These built in libraries contain many useful function and variables. The built in EGL library being referenced here is SQLLib. Built in EGL libraries are always available, and do not require an EGL import declaration.

To further investigate SQLLib, create a new source code line and enter **SQLLib.**, then press **Ctrl-Spacebar** to invoke content assist.

Specifically this line, Line-017, is invoking the connect function of SQLLib, which is used to open an explicit connection to an SQL database through JDBC. If this statement fails, program control is forwarded to Line-025.

- Line-018 and 019 are EGL abstracted SQL (cursor) statements.

  The reference to the word *abstracted* is meant to communicate that EGL generates the correct Java JDBC code for the run time target (DB2 UDB, Derby, SQL Server, Oracle, Informix IDS).

  The two SQL statements give us the ability to loop through a set of SQL table resident data, as witnessed below. While these two SQL statements are quite common, many other SQL commands supported by EGL are not; for example, invoking and managing SQL stored procedures are not always standard.

- Line-020 begins a basic SQL fetch loop and is paired with Line-023.

  This loop terminates automatically when reaching the end of the data set (when reaching the last row).

- Line-021 and 022 write output to the Java console, which in Application Developer is displayed in the Console view.

- Line-024 to 027 are the onException clause of the try block, and simply output diagnostic information which may inform us as to the source of run time error.

  Line-024 receives an EGL record of type exception, more specifically, Exception:AnyException, which automatically contains diagnostic codes and text.

  You can extend these exception records and add your own informative values. You can also programmatically throw and (catch) these run time error events.

Compiling and running the EGL simple batch program

Thus far we have edited SimpleBatch.egl to equal what is displayed in Example 15-1, added the Derby JDBC jar files to the project, and completed other smaller tasks. Now we are ready to compile and run our program.

Compiling (or generating) EGL

EGL compilation generates Java (our example) or COBOL code. To compile the SimpleBatch program, follow these steps:

- One way to compile an EGL file is to right-click a file or package in project Explorer and select **Generate**. If you right-click EGLSource and select Generate, every EGL file in the given project is compiled.
- Right-click **EGLSource** and select **Generate**. An EGL Generation Results view opens and displays three compiles: `SimpleBatch`, `ConditionHandlingLib`, and `CustomerLib`.

The Java code generated from the compilation is visible in the Project Explorer under `JavaSource`. If you understand Java, open `myprograms/SimpleBatch.java`. Notice that the original EGL statements are shown as comments, with the generated Java code after each statement.

Running the simple batch program

There are several ways to run an EGL non-Web (Program), perhaps the easiest being to select the associated Java file from the Project Explorer:

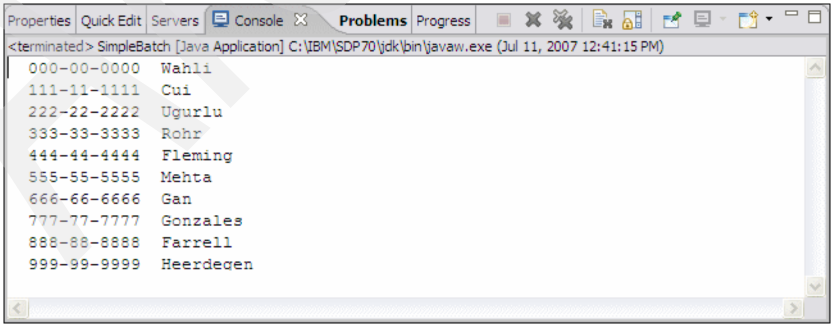
- Locate the matching Java package and program for the EGL program:

```
EGL:  EGLSource/MyPrograms/SimpleBatch.egl
Java: JavaSource/myprograms/SimpleBatch.java
```

- Right-click `SimpleBatch.java` and select **Run As** → **Java Application**.
- Output is displayed in the Console view. Most probably you get an error:

```
Error: EGL0505E Cannot connect to jdbc:derby:C:/7501code/database/derby
/ITSOBANK: Failed to start database 'C:/7501code/database/derby
/ITSOBANK', see the next exception for details.
```

- Derby is single user system, and we are still connected to `ITSOBANK` from the development activity. In the Database Explorer view right-click **ITSOBANK** and select **Disconnect**.
- Rn the program again: Right-click `SimpleBatch.java` and select **Run As** → **Java Application**. The resulting Console view is shown in Figure 15-7.



```
<terminated> SimpleBatch [Java Application] C:\IBM\SDP70\jdk\bin\javaw.exe (Jul 11, 2007 12:41:15 PM)
  000-00-0000  Wahli
  111-11-1111  Cui
  222-22-2222  Ugurlu
  333-33-3333  Rohr
  444-44-4444  Fleming
  555-55-5555  Mehta
  666-66-6666  Gan
  777-77-7777  Gonzales
  888-88-8888  Farrell
  999-99-9999  Heerdegen
```

Figure 15-7 SimpleBatch output in Console view

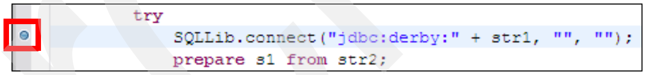
Debugging the EGL simple batch program

Chapter 22, “Debug local and remote applications” on page 1041 covers the Rational Application Developer debugger. While that chapter discusses using the debugger for Java source code and other topics, this same debugger also supports EGL. If you have never used a graphical debugger, there is a small learning curve to understand breakPoints, variable tracing, and related. Some of the things you can do with the debugger include:

- Animate the execution of the EGL (program); watch the source code lines as they are executed in order.
- Stop program execution at any time, and step into or over EGL functions.
- Have the program stop execution on a condition. For example, stop the program only `if (i > 7)`.
- Evaluate and set variable values on the fly.
- View an automatically generated and maintained outline of the variables, function, and other elements of your source code.

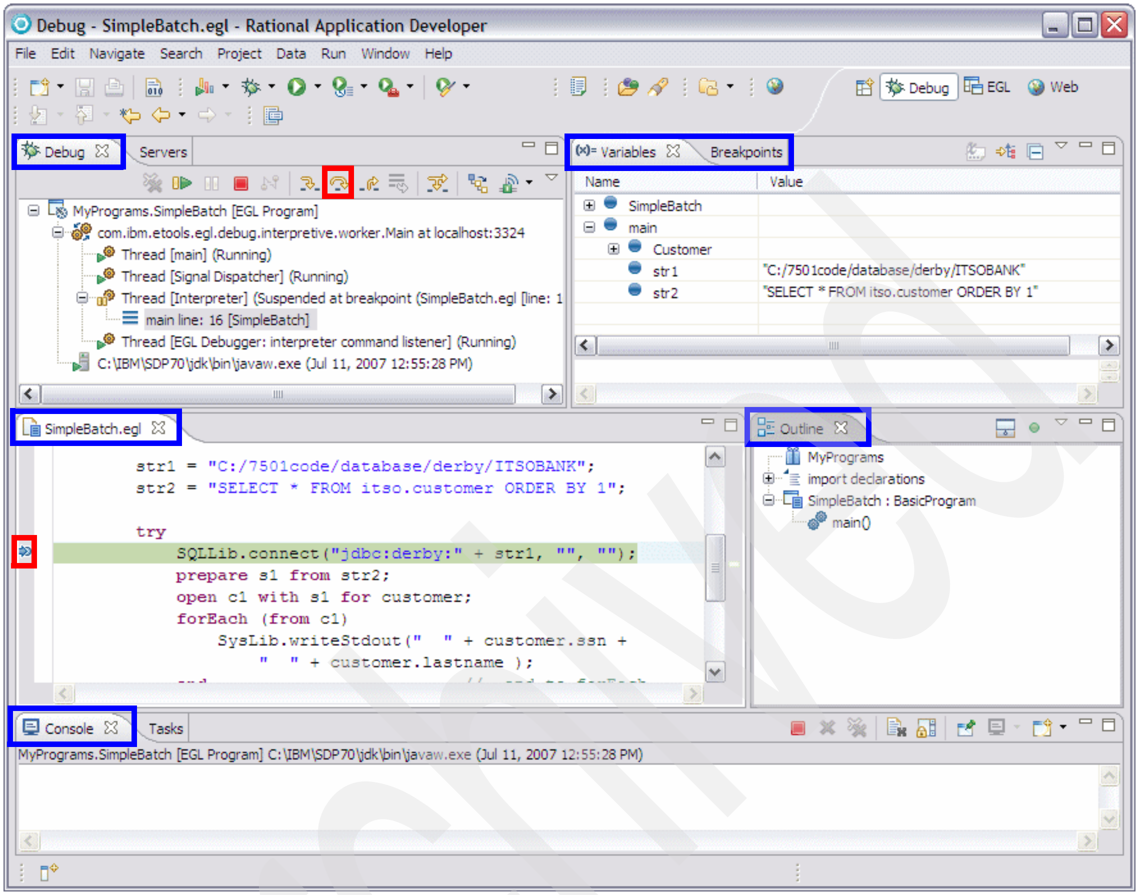
Repeating the contents of Chapter 22, “Debug local and remote applications” on page 1041 is beyond the scope of this section. Review the chapter to get the basic concepts and operation of the Application Developer Debugger, then apply this skills towards debugging `SimpleBatch.egl`:

- Open `SimpleBatch.egl` and set a breakpoint: Double-click into the gray side bar on the left on the line `SQLLib.connect("jdbc:derby...)`. A small blue dot appears in the side bar.



```
try
    SQLLib.connect("jdbc:derby:" + str1, "", "");
    prepare s1 from str2;
```

- Right-click **SimpleBatch.egl** and select **Debug EGL Program**.
- When prompted, click **Yes** to switch to the Debug perspective.
- The Debug perspective with the program stopped at the breakpoint is shown in Figure 15-8:
  - Notice the source code, Debug view, and Variables view.
  - The source code shows the current line.
  - In the Debug view, you have the icons to step through the code.


Debug - SimpleBatch.egl - Rational Application Developer
File Edit Navigate Search Project Data Run Window Help
Debug
EGL
Web
Debug
Servers
Variables
Breakpoints
Outline
MyPrograms
import declarations
SimpleBatch : BasicProgram
main()
Console
Tasks

```
MyPrograms.SimpleBatch [EGL Program]
  com.ibm.etools.egl.debug.interpretive.worker.Main at localhost:3324
    Thread [main] (Running)
    Thread [Signal Dispatcher] (Running)
    Thread [Interpreter] (Suspended at breakpoint (SimpleBatch.egl [line: 1
      main line: 16 [SimpleBatch]
    Thread [EGL Debugger: interpreter command listener] (Running)
  C:\IBM\SDP70\jdk\bin\javaw.exe (Jul 11, 2007 12:55:28 PM)
```

Name	Value
SimpleBatch	
main	
Customer	
str1	"C:/7501code/database/derby/ITSOBANK"
str2	"SELECT * FROM itso.customer ORDER BY 1"

SimpleBatch.egl

```
        str1 = "C:/7501code/database/derby/ITSOBANK";
        str2 = "SELECT * FROM itso.customer ORDER BY 1";

        try
            SQLLib.connect("jdbc:derby:" + str1, "", "");
            prepare s1 from str2;
            open c1 with s1 for customer;
            forEach (from c1)
                SysLib.writeStdout("   " + customer.ssn +
                    "   " + customer.lastname );
```

```
MyPrograms.SimpleBatch [EGL Program] C:\IBM\SDP70\jdk\bin\javaw.exe (Jul 11, 2007 12:55:28 PM)
```

Figure 15-8 EGL Debugger of simple batch program

- Click the Step Over icon to step through the EGL code.
- As you go though the source code, the results are displayed in the Console view.
- Click the icon to run through the program.
- Debug the program once more. When stopped at the breakpoint, overtype the value of **var1** in the Variables view with:

  `"SELECT` **`lastname`** `FROM itso.customer ORDER BY 1"`

- Run through the program and only last names are displayed.

Close the Debug perspective when finished.

EGL example 2: Web site with two pages using EGL

In this section we create a simple two-page Web site using EGL. Following the industry standard design pattern of model-view-controller (MVC), there are three logical functional areas of an end user interface program. These are as follows:

- **Model** (also known as the data model)—This is the data persistence layer, the part of our program that interacts with a database server.
- **View** (user interface)—Presentation layer to the program, be it Web, graphical, or other.
- **Controller**—The controller ties end user initiated program events from the view (button clicks), to functions calls related to the model (the database). The controller sits between the view and the model.

Regarding model-view-controller and EGL, we offer the following explanation:

- The model is written in the EGL programming language.

  There are wizards to write much of this code for you, and there is a set of EGL abstract data access command verbs that allow you to interface with disparate types of data sources in a consistent fashion. This single set of data access commands work the same with SQL data sources, non-SQL data sources, IBM MQ-Series servers, and so forth.
- The controller is written in the EGL programming language, and there are wizards, content assist, (code) snippets, and other tools to reduce that coding burden.
- The view is slightly different.

  When working on EGL Web applications, the user interface (the presentation layer) is entirely painted. EGL Web programmers use the very same Web Page Designer that Rational Application Developer Java/J2EE programmers and others use.

  EGL Web pages use HTML, cascading style sheets (CSS), XML, JavaScript, AJAX; basically any new and open source Web site technology that is available. And again, this is all painted, drag and drop, using the same industry leading tool that comes in Rational Application Developer.

  As stated above, the view, the presentation layer, is painted. *Behind* every EGL Web page is an event handler, and that event handler is written in EGL. (In the Java/J2EE world the event handler to any Web page is written in Java.)

Dependencies for EGL example 2

The simple batch program was a stand-alone example, with no external source code or other dependencies. To demonstrate how to share code across EGL projects, this example uses the EGL data items and SQL record from example 1. If you have not previously completed example 1, then you have to perform these sub-tasks from that example:

- "Creating a database connection resource" on page 672.

  Perform this activity now.

  You must have the `ITSOBANK` database created for use with the embedded Derby database server. While many other database servers work, this example uses Derby.

- "Generating EGL data items, records, SQL access libraries" on page 677.

  Perform this activity after you create the EGL Web Project, below.

If we just run a Web page from an EGL Web project, we would receive a series of prompts that would start the application server (if not running), deploy the EGL Web project (if not previously deployed), and start the application contained within the EGL Web project. This one call to action would invoke a number of sub-actions, some of which can be resource intensive when run for the first time. To minimize any chance for error, we perform each of these steps manually in the sections that follow.

Starting the application server

EGL Web projects automatically generate Java/J2EE code and deploy to a supported Java/J2EE compliant application server. The application server that ships with Rational Application Developer V7 is WebSphere Application Server V6.1. To deploy our code (project) to the application server, the application server must be started:

- Select **Window** → **Open Perspective** → **Web**. The Web perspective is more suited for example 2 because we require Web tools and access to the application server.
- Locate the Servers view.

  The Server view is opened with the Web perspective, and is generally located in the lower center portion. The Servers view is used to start, stop, and create new application servers, as well as deploy Java/J2EE EAR files to defined application servers, and more.

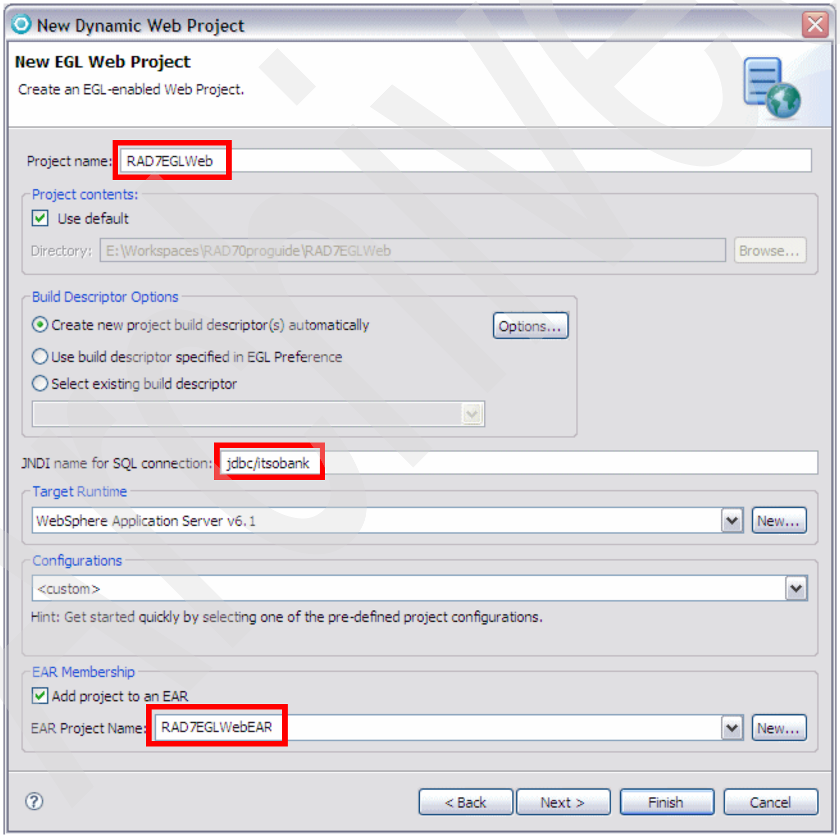
- In the Servers view, right-click the default application server **WebSphere Application Server v6.1** and select **Start**.

  If this is a new server, or a server that has never been started before, this action takes a while to complete; several administrative applications are installed inside a new server, and these installations take time.

Creating an EGL Web project

For example 2, we create an EGL Web project:

- Select **File** → **New** → **Project** → **EGL** → **EGL Web Project** and click **Next**.
- In the New Dynamic Web Project dialog (Figure 15-9):
  - Enter **`RAD7EGLWeb`** as the Project name.



New Dynamic Web Project

New EGL Web Project

Create an EGL-enabled Web Project.

Project name: RAD7EGLWeb

Project contents:

☑ Use default

Directory: E:\Workspaces\RAD70proguide\RAD7EGLWeb

Build Descriptor Options

● Create new project build descriptor(s) automatically

○ Use build descriptor specified in EGL Preference

○ Select existing build descriptor

JNDI name for SQL connection: jdbc/itsobank

Target Runtime

WebSphere Application Server v6.1

Configurations

<custom>

Hint: Get started quickly by selecting one of the pre-defined project configurations.

EAR Membership

☑ Add project to an EAR

EAR Project Name: RAD7EGLWebEAR

Figure 15-9 New EGL Web Project dialog

– For JNDI name for SQL connection, enter the value `jdbc/itsobank`.

  As mentioned previously, the EGL source code we create and run is hosted inside a Java/J2EE compliant application server. These application servers offer resource pools, including but not limited to persistent (always open) SQL database connections, LDAP handles, and many more. In this context, a JNDI name is the identifier to a specific one of these pools.

  The JNDI name we entered above could have had any value. As a convention, JNDI names representing SQL data sources start with the prefix `jdbc` and then a reference to the database. *This name resides in the Java/J2EE domain and is case sensitive.*

  In this entry field, we are saying that the project makes reference to a JNDI name with the given value. Later we define the specific resource that this JNDI value represents.

– Select **Add project to an EAR** and enter **`RAD7EGLWebEAR`** as name.

- Click **Finish** (or go through the remaining pages with **Next**).

  This action closes the New Dynamic Web Project dialog, and creates two new project entries in the Project Explorer view; one entry follows the name of our project, `RAD7EGLWeb`, and contains our source code. The other is named `RAD7EGLWebEAR`.

  Java—and more specifically, Java/J2EE—have a number of file types that bundle up the application that is to be deployed. These include JAR (Java archive) files, WAR (Web archive) files, EAR (enterprise archive) files, and more. Each of these file types are basically zip files with a rigidly defined internal file structure and table of contents (a manifest file of sorts).

  We do nearly all of the work in the Project Explorer view entry for our project, `RAD7EGLWeb`. A small amount of work, related to deployment settings, is done in the entry of the associated EAR project, `RAD7EGLWebEAR`.

Note: The Web Diagram Editor may optionally open at this point. If this editor opens, close it. Using the Web Diagram is outside our scope.

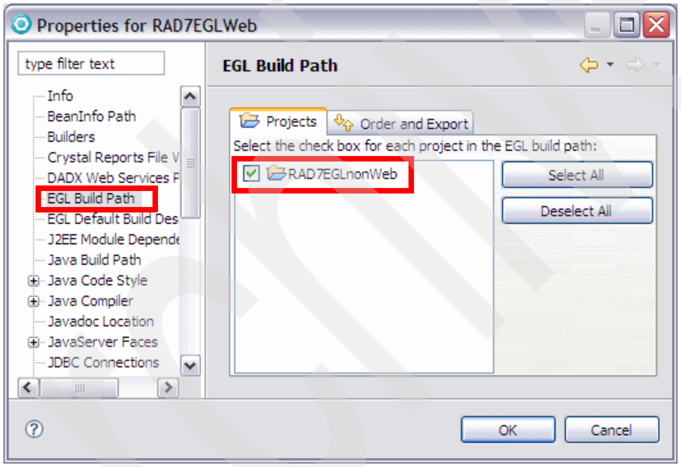
Enabling import of EGL source code from other EGL projects

Inside any given EGL project, you can make reference to EGL source code from other EGL projects. This capability allows for code re-use across EGL projects. Here we *enable* this capability, later, in our EGL source code we import the specific EGL source code we want to re-use.

Tip: You only have to perform this section of steps if you are sharing source code between the two EGL projects as used in this chapter. If you made or are now making the EGL data access application inside this EGL Web project, this section of steps is not required.

- In the Project Explorer right-click **RAD7EGLWeb** and select **Properties**. In the Properties dialog (Figure 15-10):
  – Select **EGL Build Path** in the left pane.
  – Select the **Projects** tab, then select the **RAD7EJBnonWeb** project.

    This action enables us to make reference to EGL source code contained in the selected EGL project. You can allow for reference to EGL source code from numerous EGL projects.
  – Click **OK** to close the properties dialog.



Select the check box for each project in the EGL build path:

☑ RAD7EGLnonWeb

Figure 15-10 Project Properties: EGL Build Path

Defining a JNDI resource for an EGL Web project

When we created the EGL Web project above, we said this project would make reference to a JNDI resource with the identifier `jdbc/itsobank`. Now we define this JNDI resource to point to the `ITSOBANK` database.

Note: JNDI resources can be defined on the application server side, or in the Web project EAR file itself, and there are benefits to either choice.

If you define a JNDI resource on the server side, then all Web project EAR files hosted on that application server would have access to that single JDNI definition. If you define the JNDI resource inside the EAR file, then that (application) has one less external dependency, one less thing that can break.

We are defining our JNDI resource inside the application, inside the EAR file.

Note that we configure the data source with JNDI name `jdbc/itsobank` in the server in "Configuring the data source in WebSphere Application Server" on page 1313. There is no harm to have the same JNDI name configured also in the EAR file, as long as they point to the same database.

- In the Project Explorer, locate the EAR file entry for the EGL Web project, `RAD7EGLWebEAR`.

  If you need a memory aid to distinguish why we are currently referencing the EAR file, and not the EGL Web project, we are currently setting deployment options, which is work generally done in the EAR file.

- Expand `RAD7EGLWebEAR` and open the Deployment Descriptor in the Application Deployment Descriptor editor.

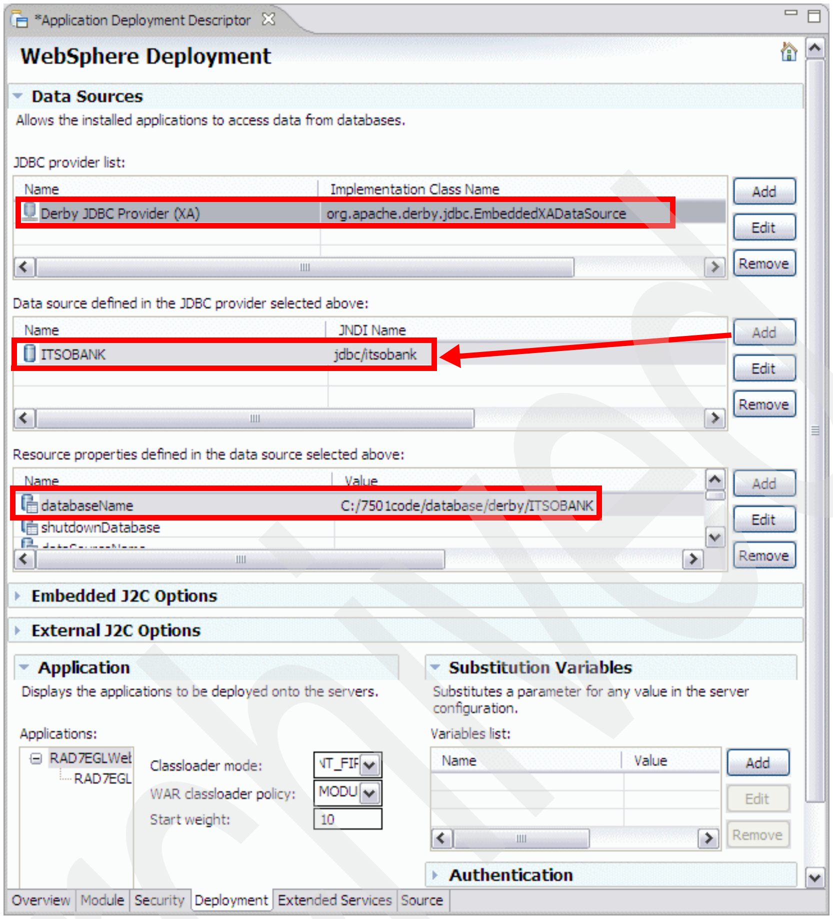
  This actually opens the file `/META-INF/`**`application.xml`**.

Note: The `application.xml` is one of several configuration files for a (EGL) Web project and its EAR project. Because EGL Web projects generate J2EE Java and resources, EGL Web projects use the common Java/J2EE run time environment and standards. The `application.xml` is a standard Java/J2EE configuration file. Generally these files are XML files, and Application Developer offers a graphical editor for editing these files more easily.

- Select the **Deployment** tab (Figure 15-11 shows the completed editor).
- The first area of interest is the JDBC provider list.

  In effect, this is where you register a *category of database vendor* as determined by their JDBC driver packaging. If your application used the open source Derby database server, DB2 UDB, and IBM Informix IDS, you would have three entries in this section, because these three servers use different JDBC driver packaging.

  We are using Derby in this example and the entry for Derby is already present in this list. If our example were only using DB2 UDB, we would add an entry for DB2 UDB. To be extra clean, we would likely delete the entry for Derby.



*Application Deployment Descriptor

WebSphere Deployment

Data Sources

Allows the installed applications to access data from databases.

JDBC provider list:

Name	Implementation Class Name
Derby JDBC Provider (XA)	org.apache.derby.jdbc.EmbeddedXADataSource

Data source defined in the JDBC provider selected above:

Name	JNDI Name
ITSOBANK	jdbc/itsobank

Resource properties defined in the data source selected above:

Name	Value
databaseName	C:/7501code/database/derby/ITSOBANK
shutdownDatabase	

Embedded J2C Options

External J2C Options

Application

Displays the applications to be deployed onto the servers.

Applications:

RAD7EGLWel
RAD7EGL

Classloader mode: NT_FIF

WAR classloader policy: MODU

Start weight: 10

Substitution Variables

Substitutes a parameter for any value in the server configuration.

Variables list:

Name	Value

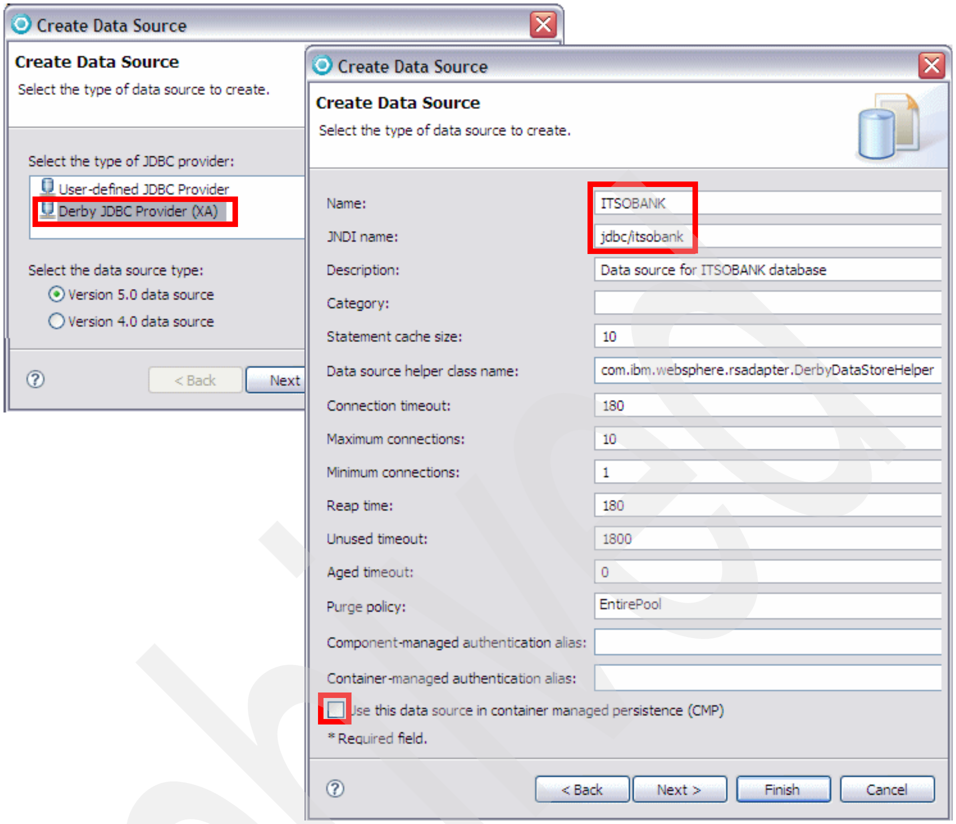
Authentication

Figure 15-11 Application Deployment Descriptor editor (application.xml)

- The second area of interest in the section entitled Data source defined in the JDBC provider section selected above.

  This is the area where we define a specific data source (a database) for the selected JDBC driver (Derby JDBC Provider (XA) is currently selected).

- Click **Add** next to the data source section to define a data source for Derby.
- In the Create Data Source dialog, select the type of JDBC provider as **Derby JDBC Provider (XA)** and click **Next** (Figure 15-12).



Create Data Source

Select the type of data source to create.

Select the type of JDBC provider:

- User-defined JDBC Provider
- Derby JDBC Provider (XA)

Select the data source type:

● Version 5.0 data source

○ Version 4.0 data source

Create Data Source

Select the type of data source to create.

Name:	ITSOBANK
JNDI name:	jdbc/itsobank
Description:	Data source for ITSOBANK database
Category:	
Statement cache size:	10
Data source helper class name:	com.ibm.websphere.rsadapter.DerbyDataStoreHelper
Connection timeout:	180
Maximum connections:	10
Minimum connections:	1
Reap time:	180
Unused timeout:	1800
Aged timeout:	0
Purge policy:	EntirePool
Component-managed authentication alias:	
Container-managed authentication alias:	

□ Use this data source in container managed persistence (CMP)

* Required field.

Figure 15-12 Create data source (1) and (2)

- In the second page (Figure 15-12, right-hand side):
  - Enter **ITSOBANK** as Name (any name will do).
  - Enter **jdbc/itsobank** as JNDI name (this has to match exactly the specification in the EGL Web project).
  - Optionally enter a description.
  - Clear **Use this data source in container managed persistence (CMP)** (we are not using EJBs).
  - Click **Next**.

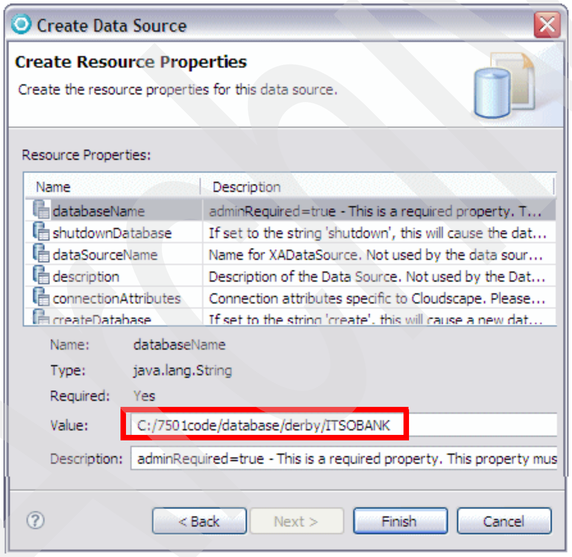
> **Note:** The embedded Derby database server is unique, because it has the ability to run inside the same process space as the parent (process). In this case, the parent is the application server. Hence, Derby is accessed by whatever user name the application server is operating with.
>
> With other database providers, you would have to add an entry for Container managed authentication alias, and this alias must be defined with a user ID and password in the Authentication section.

- In the third and final step (Figure 15-13), select the **databaseName** property and enter the directory name of the Derby database as value, in our case:

  ```
  C:/7501code/database/derby/ITSOBANK
  ```

  This is the only required property for embedded Derby.

  Different vendors display different variables to be set in this dialog. For example, a network accessible database has a port number and host name or IP address.



Create Data Source

Create Resource Properties

Create the resource properties for this data source.

Resource Properties:

Name	Description
databaseName	adminRequired=true - This is a required property. T...
shutdownDatabase	If set to the string 'shutdown', this will cause the dat...
dataSourceName	Name for XADataSource. Not used by the data sour...
description	Description of the Data Source. Not used by the Dat...
connectionAttributes	Connection attributes specific to Cloudscape. Please...
createDatabase	If set to the string 'create', this will cause a new dat...

Name: databaseName

Type: java.lang.String

Required: Yes

Value: C:/7501code/database/derby/ITSOBANK

Description: adminRequired=true - This is a required property. This property mus

Figure 15-13 Create data source dialog (3)

- Click **Finish** and the dialog closes. The properties are added to the Resource properties section in the deployment descriptor.
- You can select an entry in any section and click **Edit** to modify a value.

For example, select the Derby JDBC provider (XA) and click **Edit**. Notice that the Class path entry points to `${DERBY_JDBC_DRIVER_PATH}/derby.jar`. The substitution variable `DERBY_JDBC_DRIVER_PATH` is defined in the server configuration.

- Save and close the Application Deployment Descriptor editor.

Each section of work we have completed thus far for this example were preparatory in nature; meaning that these sections are performed one time only. Once you have created and configured a given EGL Web project as detailed above, the only tasks you are left with are painting and linking the Web pages.

Creating the Web list page

There are several keywords and identifiers representing architectures, design patterns, frameworks, and more, within the Web and business application programming space. A *user interface design consideration* that we introduce here is the concept of list pages and detail pages.

If you go to your favorite on-line bookseller and query an author or subject area, you are likely to receive a *list page,* that is, a Web page offering multiple listings (multiple hits), each with its associated minimal identifying or key data. In the case of books, you might receive the book title, and a very brief descriptive entry regarding each given book. Then a link or menu command allows you to access a more detailed entry for a any given (book); the link sends you to a *detail page*, which has the full (single) book listing.

The example we are about to create uses two Web pages: A list page that shows numerous customer records with only a subset of the customer columns, and then a detail page with the full width of customer columns. A link on the list page allows access to the detail page, and a button on the detail page allows to return to the list page.

In online transaction processing systems, often the list page is read only, while the detail page allows writes. To offer a simple example, both of our pages just read.

Create the JSF Web list page

In this section we are creating a new Faces JSP Web page for use with EGL:

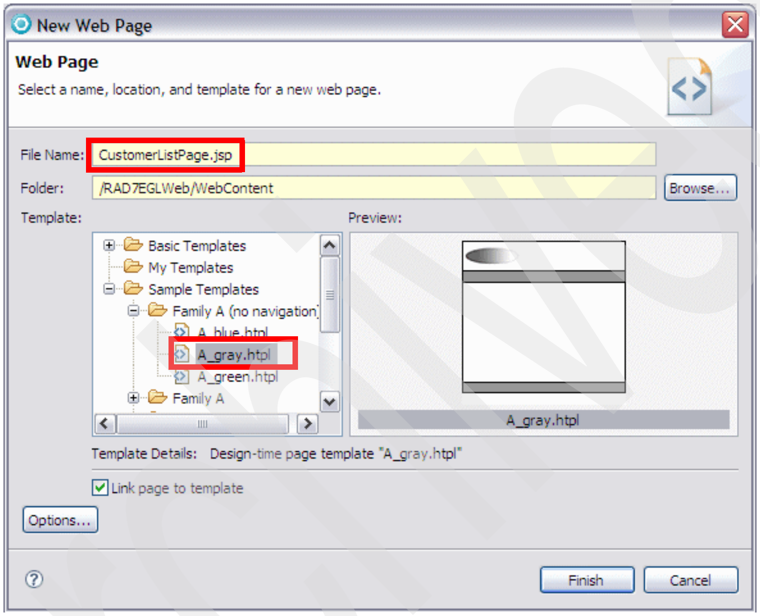
- Expand `RAD7EGLWeb`, right-click **WebContent** and select **New** → **Web Page**.

Note: Whereas all EGL source code goes under `EGLSource`, all HTML, XML, JavaScript, CSS, and related files go under `WebContent`.

- In the New Web Page dialog (Figure 15-14) enter **`CustomerListPage.jsp`** as the File Name (be certain to add a suffix of `.jsp`, or this file defaults to HTML).

  Apply a Page Template to this page. Select **Sample Templates** → **Family A (no navigation)** → **A_gray.htpl**.

> **Note:** Page templates are one method that allow you to include re-usable visual components on single or sets of Web pages. Page templates also promote a consistent look and feel across numerous pages.



New Web Page

Web Page

Select a name, location, and template for a new web page.

File Name: CustomerListPage.jsp

Folder: /RAD7EGLWeb/WebContent

Template: Sample Templates → Family A (no navigation) → A_gray.htpl

Template Details: Design-time page template "A_gray.htpl"

☑ Link page to template

Figure 15-14 New Web Page dialog

- Click **Finish** and the `CustomerListPage.jsp` opens in the Page Designer. The JSP file itself is added to the `WebContent` folder.

When designing Web pages with Page Designer, you work with these views:

- **Page Designer** (editor) with Design, Source, and Preview tabs
- **Palette** view with HTML, JSP, EGL, Faces, Data, and other parts
- **Properties** view for details of elements selected in Page Designer
- **Page Data** view for session and request block data and JavaBeans

The basic or preferred layout of our layout is displayed in Figure 15-15.

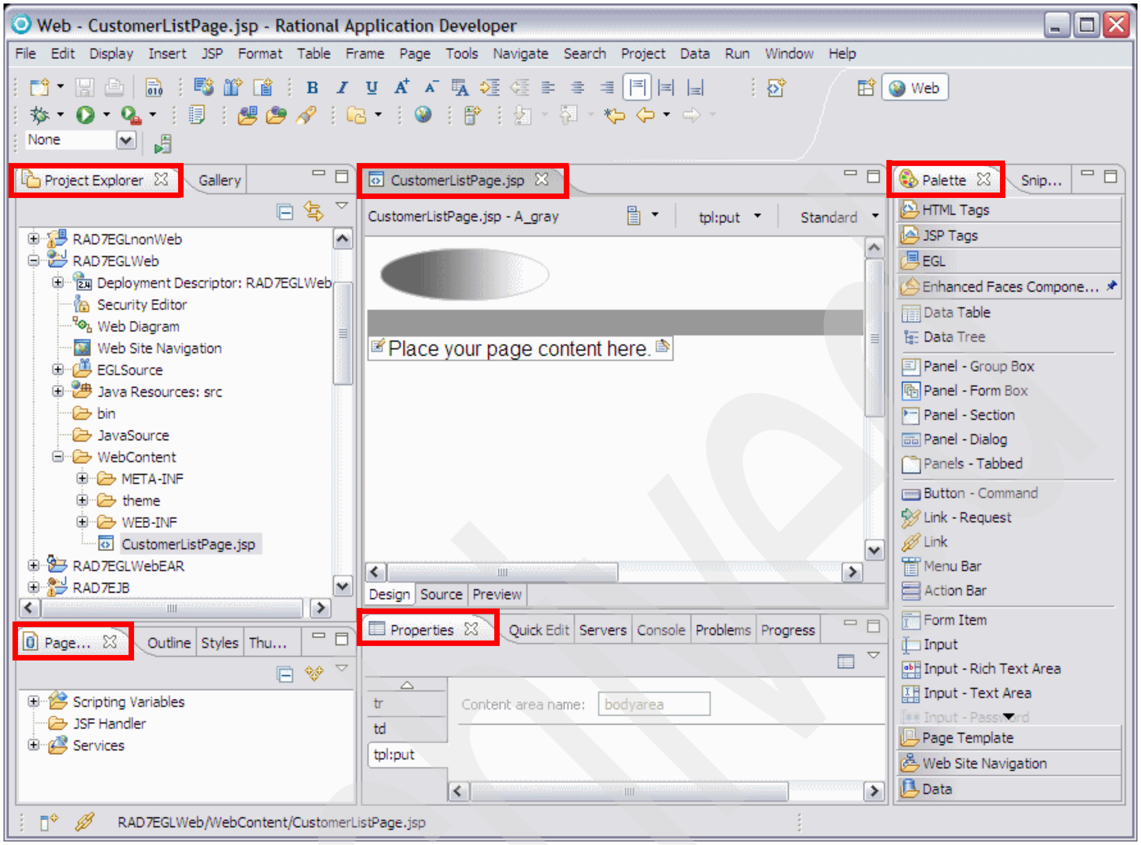


Figure 15-15 Basic Layout of e Web perspective

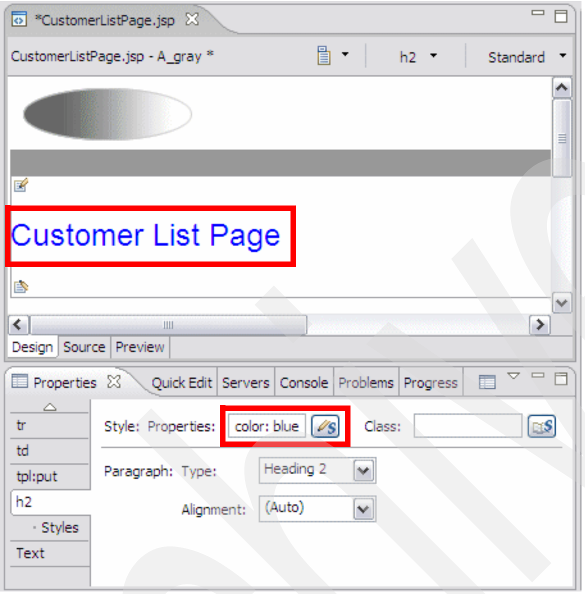
- The Page Designer is in the top middle position. This editor is like our canvas, where we drop visual controls from the Palette view. Most of our work is done in the Design tab, although you can edit the HTML and JSP source directly in the Source tab if you want.
- The Palette view has drawers (as in desk drawers), that open (expand) and present visual components that may be dragged and dropped into Page Designer (both Design and Source tab).
- The remaining views we present by example.

Setting the page title of the list page

To set the page title, perform these steps:

- In Page Designer place the cursor just before the letter P of Place your page, then select **Insert** → **Paragraph** → **Heading 2**.
- A new line opens. Type the text `Customer List Page`.

- Highlight the entire string, `Place your page content here.` (place the cursor before the letter P and drag to the end after the period), then press **Delete**.
- Place the cursor into the string `Customer List Page`.
- In the Properties view, **h2** tab (Figure 15-16), click the Style icon . This opens the New Style dialog, where you can click the Color icon to set the text to **blue**.



Style: Properties: color: blue

Class:

Paragraph: Type: Heading 2

Alignment: (Auto)

Figure 15-16 Customer List Page: Title

- In the Source tab of Page Designer locate the line `<title>A_gray</title>` (close to the top). Change the title text to `Customer List Page`.
- Save the JSP. When prompted with a Conflict in Encoding, click **Yes**.

Creating and modifying the EGL JSF page handler

For every EGL Web page, there exists one and only one EGL JSF page handler. The Web page represents the view from model-view-controller, and the handler represents the controller. These two entities share the same identifier; an EGL Web page called `CustomerListPage.jsp` has an associated handler called `CustomerListPage.egl`. Handlers are written in EGL.

- In the Design tab right-click anywhere on the canvas and select **Edit Page Code**. This action opens the EGL Source editor for the EGL JSF page handler associated with this EGL Web page.

- In the EGL Source editor edit the source to match Example 15-2. The lines in bold are new.

  The line numbers displayed in Example 15-2 are for reference only; **do not enter the line numbers**.

Example 15-2 CustomerListPage.egl (lines numbers are for reference only)

```
002 package jsfhandlers;
003
004 import generated.data.Customer;
005
006 handler CustomerListPage type JSFHandler
007    {onConstructionFunction = onConstruction,
008     view = "CustomerListPage.jsp"}
009
010    customers Customer[0];
011
012    // Function Declarations
013    function onConstruction()
014        get customers;
015    end
016
017 end
```

A line-by-line code review of `CustomerListPage.egl` follows:

- Line-002 is the EGL package statement, specifying that the source is in `EGLSource/`**`jsfhandlers`**.
- Line-004 is an EGL import statement. It allows this EGL program to reference EGL source code and objects that are not located inside this file. Line-004 imports the definition of the `Customer` object from the `RAD7EGLnonWeb` project. This cross referencing was specified in “Enabling import of EGL source code from other EGL projects” on page 690.
- Lines-006 to 008 are one logical source code line. For presentation’s sake alone and to accommodate width, this line is displayed on three physical lines. This line declares this file to be an EGL primary part of type handler.

  There are several EGL handlers; this handler is the event listener to an EGL Web page. (Other handlers support events for Web services and printed reports.)
- Line010 defines a variable called `customers` to be a dynamic sized array of a variable which is of type `Customer`.

  `customers` must be a variable name and `Customer` must be a variable type based on the syntax rules (word order) for EGL. EGL always defines variables as *VariableName VariableType*. Because `Customer` is not an EGL keyword, this means `Customer` is a user defined data type.

If we use the features of the EGL editor to **Open On Selection**, we can retrieve the definition for `Customer`. In this example, `Customer` is an EGL SQL record with four elements (columns): `Ssn`, `Tile`, `Firstname`, and `Lastname`.

- Lines-013 to 015 define an EGL function named `onConstruction`.

  This function referenced in the definition of the EGL handler (Line-007), and is executed one time only when this page is *constructed* (served for the first time for this end user session).

- Line-014 executes the EGL abstract data access command verb `get` on a target of the dynamic array `customers`.

  In effect, the dynamic sized array `customers` is filled with all rows from the `ITSO.CUSTOMER` SQL table. If we wanted fewer data rows, we could do so.

  Properties for the SQL record definition make this array of records read from the `ITSO.CUSTOMER` table. There are ways we could read from other tables or a dynamic single or set of SQL table names.

Save and close the EGL JSF handler.

Compiling the EGL JSF page handler

Thus far we have edited `CustomerListPage.egl` to equal what is displayed in Example 15-2. Unlike EGL programs, you never run an EGL JSF handler. When you run the Web page that the handler supports, the program code for that handler is executed. If we run the `CustomerListPage.jsp` Web page now, the code in our handler is executed, but we would never know it, because we are not displaying any result from the handler.

First we compile our EGL JSF handler, then we change its associated Web page to display some of its results.

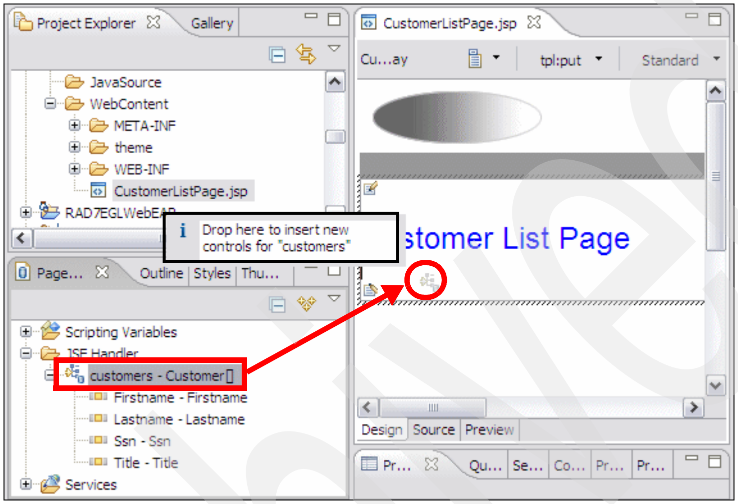
To compile the EGL JSF handler, right-click **EGLSource** (in `RAD7EGLWeb`) and select **Generate**.

Extending the Web list page with customer result data

Currently we have a base EGL Web page that displays a title, and we have the EGL program code that sits behind that page in the manner of an EGL JSF handler. Now we have to paint visual controls on the EGL Web page that can make use of this handler.

For this next set of tasks, we need the Page Designer (Design tab), Page Data view, and Palette view visible:

- In the Page Data view expand JSF Handler and locate the new EGL dynamic array of SQL records named `customers`.
- Drag and drop `customers` to the Page Designer under the title but before the icon (Figure 15-17).



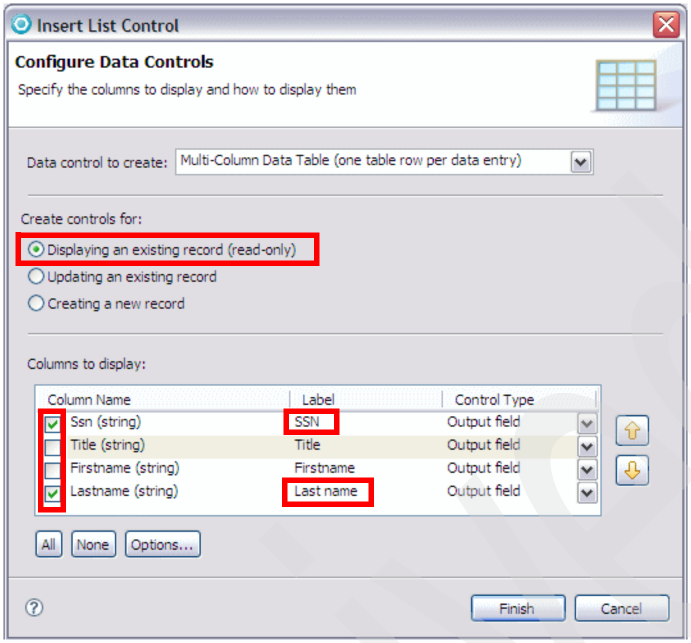
Drop here to insert new controls for "customers"

Figure 15-17 Drag and drop customers from Page Data to Page Designer

- Releasing `customers` in the JSP opens the Insert List Control dialog.
- Change to Insert List Control dialog (Figure 15-18):
  - Select **Displaying an existing record (read-only)**.
  - Clear **Title** and **Firstname**, we only retrieve Ssn and Lastname.
  - Change the labels of Snn to **SSN** and Lastname to **Last name**.
  - Click **Finish**.
- This action closes the Insert List Control dialog and adds a new visual control to the Web page editor. The official name for this type of visual control is, `hx:dataTableEx(tended)`—visible in the Properties view when selected—in effect, it is an HTML table that is bound to our dynamic array of SQL records.

  The binding is visible in the Properties view in the Value field and specifies that the data comes from the `CustomerListPage` JSF handler and the `customers` variable:

  ```
  #{CustomerListPage.customers}
  ```



Insert List Control

Configure Data Controls

Specify the columns to display and how to display them

Data control to create: Multi-Column Data Table (one table row per data entry)

Create controls for:

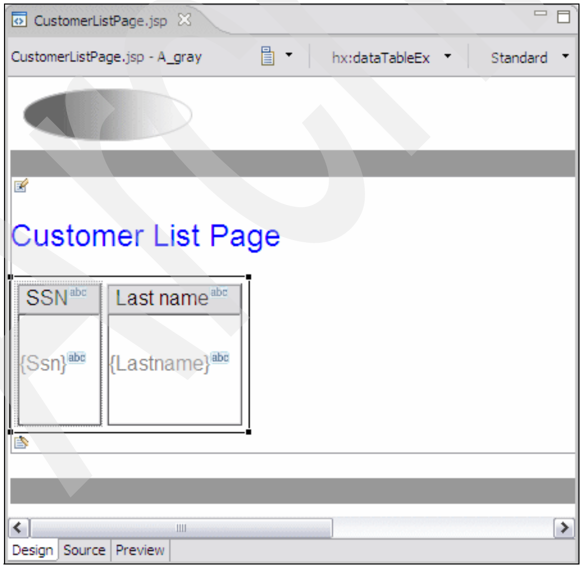
- ● Displaying an existing record (read-only)
- ○ Updating an existing record
- ○ Creating a new record

Columns to display:

	Column Name	Label	Control Type
☑	Ssn (string)	SSN	Output field
□	Title (string)	Title	Output field
□	Firstname (string)	Firstname	Output field
☑	Lastname (string)	Last name	Output field

Figure 15-18 Insert List Control dialog

- ► Save the JSP. The layout of the JSP (so far) is shown in Figure 15-19.



Customer List Page

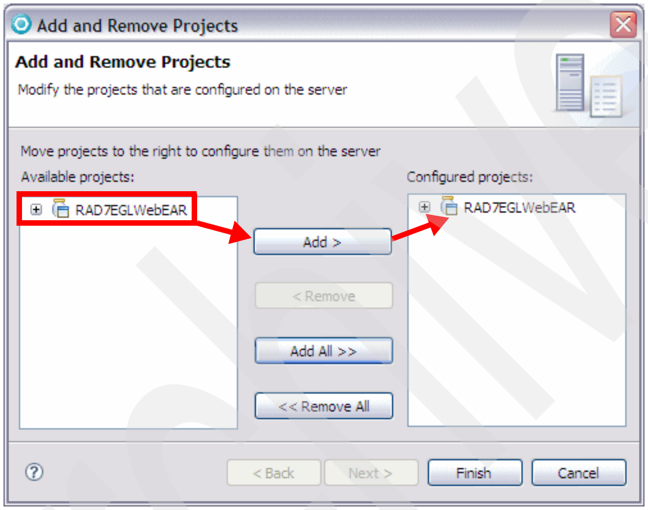
SSN	Last name
{Ssn}	{Lastname}

Figure 15-19 Customer List Page with data table

Deploying the EGL Web project to the application server

As mentioned above, every EGL Web project has an associated *EAR file*. Given an EGL Web project named `MyApp`, there is an associated entry in the Project Explorer view entitled `MyAppEAR`. Generally this entry is referred to as the *EAR file*, although this file is actually a collection (a Zip file) of multiple files. It is this entity, the EAR file, which we deploy to the application server.

- In the Servers view, right-click the default application server (which we started earlier) and select **Add and Remove Projects**.
- In the Add and Remove Projects dialog, select the **RAD7EGLWebEAR** project and click **Add** (Figure 15-20).



Add and Remove Projects

Modify the projects that are configured on the server

Move projects to the right to configure them on the server

Available projects:

RAD7EGLWebEAR

Configured projects:

RAD7EGLWebEAR

You may have more projects in the list if you performed other chapters.

Remove other projects from the server and only deploy the EGL Web project.

Figure 15-20 Servers view: Add and Remove Projects

The Add and Remove Projects dialog has two sides to its display; the left side lists projects which are not deployed but could be (Available projects), and the right side lists projects which are deployed and can be un-deployed (Configured projects).

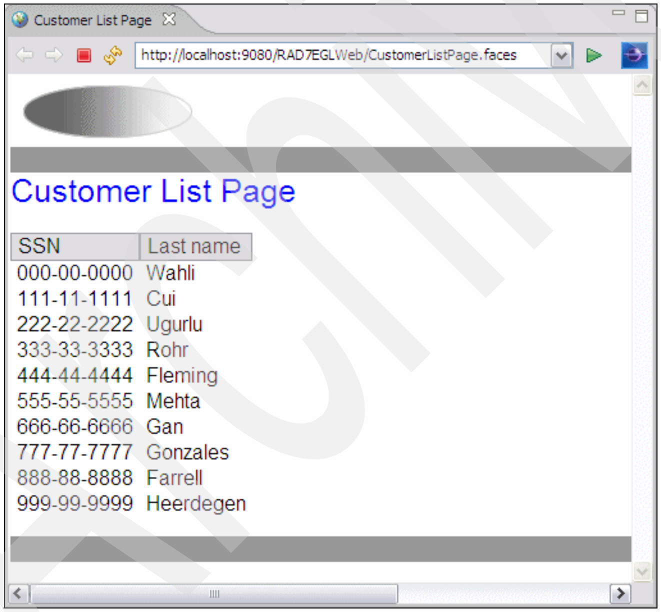
- Click **Finish** to close the Add and Remove Projects dialog, and the EAR project is deployed to the server.

Be aware that the deployment may take a while. Notice the progress information at the bottom right of the Application Developer window.

Testing an EGL Web page (EGL Faces JSP Web page)

At this point we have an EGL JSF handler that fills a dynamic sized program array with `Customer` records from a SQL database, and a Web page to display this array. It is time we test and determine that we are actually getting data from the database and displaying it.

- In the Project Explorer locate the EGL Web page, `CustomerListPage.jsp` (under `WebContent`).
- Right-click **CustomerlistPage.jsp** and select **Run As** → **Run on Server**.
- The first time you perform this action, you are likely to receive the Run on Server dialog:
  - Select **Set server as project default** (do not ask again).
  - Click **Finish**.
- This action opens an embedded Web browser with the results of the EGL Web page (Figure 15-21).



Customer List Page

SSN	Last name
000-00-0000	Wahli
111-11-1111	Cui
222-22-2222	Ugurlu
333-33-3333	Rohr
444-44-4444	Fleming
555-55-5555	Mehta
666-66-6666	Gan
777-77-7777	Gonzales
888-88-8888	Farrell
999-99-9999	Heerdegen

Figure 15-21   Embedded Web Browser with successful customer listing

At this point, the most likely sources of error that would preclude you from seeing these results include these:

- You cannot open the Derby database because the database connection is already busy: Go to the Database Explorer view and disconnect the ITSOBANK connection. Rerun the Web page as detailed above.
- An data entry error was made in configuring the JNDI resource: Re-open the EAR deployment descriptor and correct any discrepancies.
- Other errors: Consider re-creating this example in a new EGL Web project.

Changing the default Web browser

When we run the EGL Web page, Rational Application Developer used its default internal embedded Web browser. It is easy to use any external Web browser that you may have installed:

- Select **Window** → **Preferences** → **General** → **Web Browser**.
- Select **Use external Web browser**.
- Select the external Web browser you prefer.
- Click **OK**.

Extending the Web list page with a hyperlink to the detail page

Per our intended design, we have one change remaining to the Web page, CustomerListPage.jsp. We have to add a link that displays the second Web page.

Next, we describe how to add the link to the **SSN** column. You can then add another link on the **Last name** column on your own; the procedure is the same.) We can add the link(s) now, and then create the second Web page afterwards.

While we use the term link in the generic sense, technically what we are about to add is a *data aware hyper-link* (outputLinkEx). This type of link automatically retrieves the given key value from a list of records and forwards this value to the Web page with which it is associated.

For this next set of tasks, we use the Page Designer, Properties view, and Palette view. Open the CustomerListPage.jsp and select the Design tab:

- In the Ssn column, select (click) **{Ssn}^{abc}**. A box appears around the word (Figure 15-22).

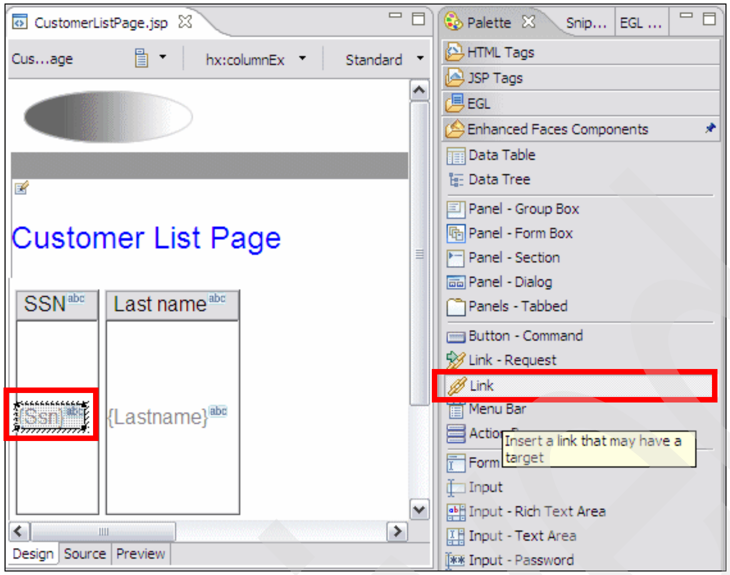


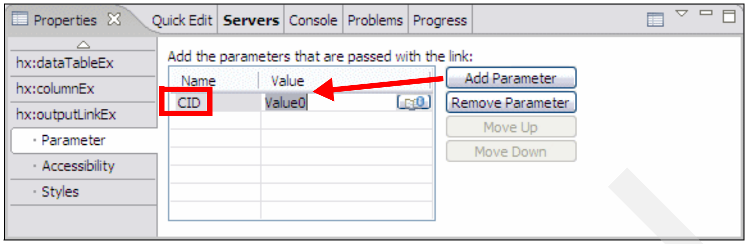
Figure 15-22 Adding a data aware hyper-link to a data column

- After {Ssn}^{abc} is selected, go to the Palette view and open the Enhanced Faces Components drawer. Locate the **Link** component, and double-click it.
- In the Configure URL dialog, enter the value `CustomerDetailPage.faces` in URL field, and click **OK**.

  Notice the Link icon that is added in front of Ssn^{abc}.

> **Note:** Alternatively, to double-clicking **Link** you can drag the Link control on top of the Ssn^{abc} control in the Design tab. The result is the same.

- In the Design tab **{Ssn}^{abc}** should still be selected (otherwise click on it again).
- In the Properties view, select the **Parameter** tab (under `hx:outputLinkEx`), and click **Add Parameter**. This action adds a line to the parameters table (Figure 15-23).



Name	Value
CID	Value0

Figure 15-23 Adding a Parameter to the hyperlink

- We want to pass the SSN of the customer as a parameter to the detail page. Parameters must have a name and a value. We want to dynamically bind the value to the SSN of the customer.
  - Click in the Name field and overtype `Name0` with **CID**.

    `CID` is the variable name that is passed in the HTTP request header. HTTP request header variables are passed by name; hence, this variable name is significant, as is its case.
  - Click in the Value field and a button with an image similar to a book (indicating the availability of a lookup) appears on the right side of the cell.
  - Click the button to open the Select Page Data Object dialog.

    Expand **customers - Customer[]** and select **Sss - Ssn**, then click **OK**.
  - The new value in the Value cell is now `#{varcustomers.Ssn}`. This variable value is created and managed for us automatically. and it equals the value of the `Ssn` column for any specific row that display. These variables are automatically managed and populated from dynamic JSP visual controls, and are in scope inside the JSP.
- Save the JSP.

You can see the JSP code of the hyperlink in the Source tab:

```
<hx:outputLinkEx id="linkEx1" styleClass="outputLinkEx"
   value="CustomerDetailPage.faces">
   <h:outputText id="textSsn1" value="#{varcustomers.Ssn}"
      binding="#{CustomerListPage.customers_Ssn_Ref}"
      styleClass="outputText">
   </h:outputText>
   <f:param name="CID" id="param1" value="#{varcustomers.Ssn}"></f:param>
</hx:outputLinkEx>
```

Test the hyperlink

Test the hyperlink by running the JSP. In the Project Explorer right-click `CustomerListPage.jsp` and select **Run As** → **Run on Server**.

Test the link in the SSN column by clicking on a SSN. The request fails with an a HTTP Error Code: 404 (`Failed to find resource /CustomerDetailPage.jsp`).

For now, this is expected. This link forwards us to a Web page that does not exist. We create this Web page in the next section.

Creating the Web detail page

Now we create the second EGL Web page, `CustomerDetailPage.jsp`, that is called from the hyperlink in the list page.

- In the Project Explorer right-click to produce a context menu, and select **New** → **Web Page**.
- In the New Web Page dialog, enter `CustomerDetailPage.jsp` as the File Name (with `.jsp` suffix).
- Apply a Page Template to this page by selecting **My Templates** → **theme/A_gray.htpl**.
- Click **Finish** and the `CustomerDetailPage.jsp` opens in Page Designer.

Set the page title of the detail page

Repeat the instructions in “Set the page title of the detail page” on page 709 to set the title of the detail page to `Customer Detail Page`:

- Insert a heading 2 before the text `Place your page content here`.
- Enter the text `Customer Detail Page`.
- Delete the old text (`Place your page content here.`).
- Make the heading 2 blue in the Properties view (`color: blue`).
- In the Source tab set `<title>Customer Detail Page</title>`.
- Save the JSP.

> **Tip:** If you are interrupted by a pop-up stating that User Operation is Waiting, click the red icon 🟥 to stop the server publishing activity.

Creating and modifying the EGL JSF page handler

Right-click anywhere in the Design tab and select **Edit Page Code**. Change the EGL code in the EGL Source editor to match Example 15-3 (the lines in bold are new, Use content assist (Ctrl-Spacebar) to help with the coding, or use copy/paste from `C:\7501code\egl\CustomerPageDetail.egl`.

Example 15-3 CustomerDetailPage.egl.

```
package jsfhandlers;

import generated.data.Customer;

handler CustomerDetailPage type JSFHandler
   {onConstructionFunction = onConstruction,
   view = "CustomerDetailPage.jsp",
   scope = request}

   customer   Customer;

   function onConstruction(CID string)
      SysLib.writeStdout("Customer: " + CID);
      customer.ssn = CID;
      get customer;
   end

   function doButtonClick()
      forward to "CustomerListPage";
   end
end
```

Here is a short explanation of the code:

- In the attributes for the EGL primary part handler is a new attribute, `scope = request`.

  By default, the scope for any EGL Web page, and the associated handler, is `session`. As a result, the `onConstruction` function is run only one time for each new user session. This is a feature, because it allows for a form of caching.

  This caching works fine for the customer list page, but does not give us the desired results for the customer detail page. Without this setting the `onConstruction` function is never re-run, and you would never see anything other than the original customer record.

- The function `onConstruction` accepts a single argument of type string (the customer SSN). EGL functions can receive and return arguments of great complexity with advanced scoping, reference, and mapping.
- The first line in the `onConstruction` function writes a diagnostic output to the Console view. This is a cheap version of debugging and tracing. For a production system, this line should be disabled through a switch.
- The `doButtonClick` function could have been called anything, `doButtonClick` is not a keyword of any sorts. This function is invoked by clicking a button in the Web page.

- The command `forward to` is an EGL command verb allowing you to programmatically control end user navigation by loading a new EGL Web page.

Compiling the EGL JSF page handler

Save the EGL source code and close the editor. To compile the page handler right-click **EGLSource** → **CustomerDetailPage.egl** and select **Generate**.

Completing the Web detail page

We want to display the customer record in the detail page. For this task we use the Page Designer, Page Data view, and Palette view:

- In the Page Data view, select the EGL SQL record **JSF Handler** → **customer**.
- Drag and drop customer into the Web page under the title `Customer Detail Page` (refer to Figure 15-17 on page 702).
- This action opens the Insert Control dialog:
  - Select **Displaying an existing record (read only)**.
  - Leave al fields selected.
  - Change the labels to nicer text: SSN, First name, Last name
  - Click **Finish** to close the dialog.
- This action adds an HTML table with `outputText` controls and field labels to the Web page.
- Optionally delete the `x{Error Messages}` line. Click on the control and press **Delete**.

  This visual control give the end user feedback for data entry errors and the like. Its use is not required. Many folks select not to use this visual control for style reasons.

Add a command button to the Web page

We require a button to go back to the list page from the detail page:

- Place the cursor after the table (or after the {Error Messages} control) and press **Enter** twice.
- In the Palette view open the Enhanced Faces Components drawer and locate the **Button - Command** component.
- Select the **Button - Command** component and drag/drop it into the Web page below the table and error messages. (You can also double-click the component in the Palette.)

- This action places a **Submit** button into the Web page (Figure 15-24).

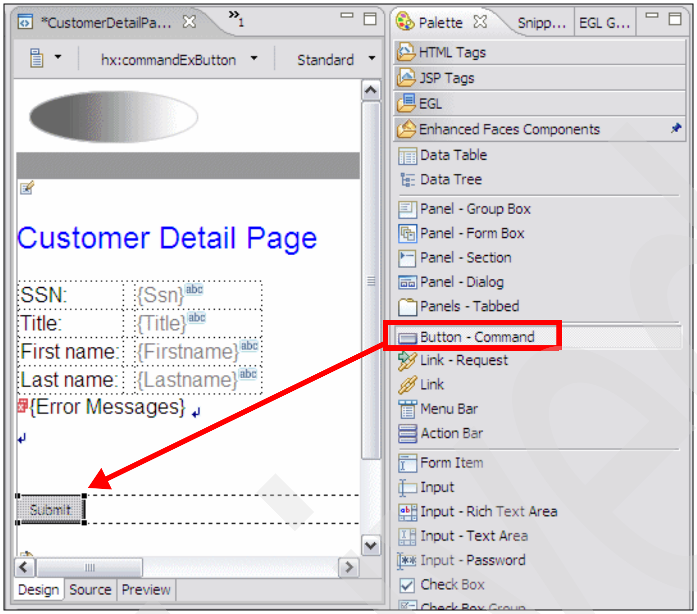
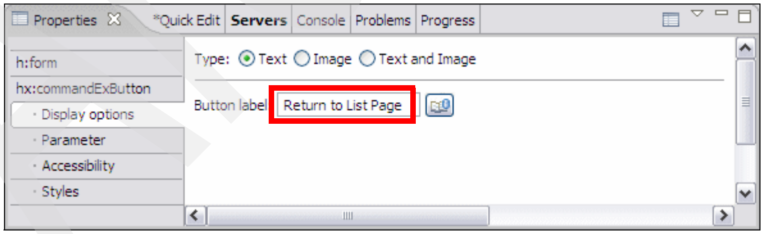


Figure 15-24 Inserting a command button

Set the properties for the Submit button

To set the properties of the Submit button, do these steps:

- Select the **Submit** button in the Design tab.
- In the Properties view the **hx:commandExButton** tab is selected. Select **Display options** and change the Button label text to **Return to List Page** (Figure 15-25).



Type: ● Text ○ Image ○ Text and Image

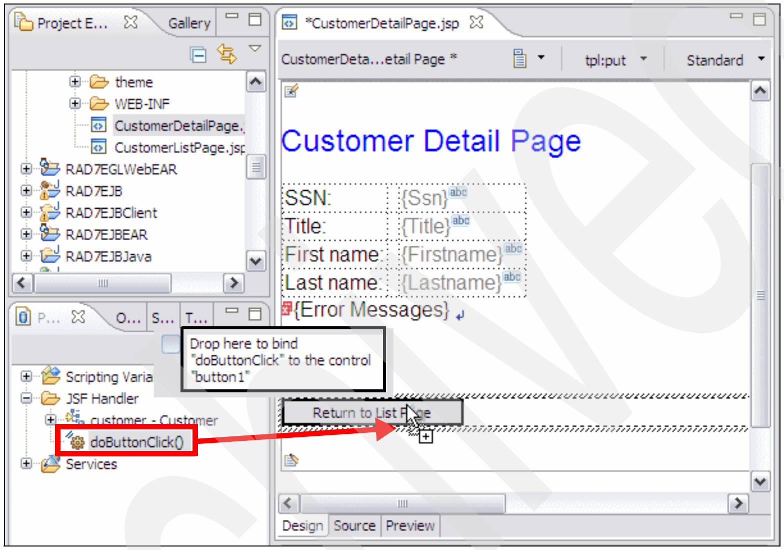
Button label: Return to List Page

Figure 15-25 Command button display properties

Bind the Command Button to the EGL handler function

When the command button is clicked we have to invoke the `doButtonClick` function in the EGL handler. This is done by binding the command button to the handler function:

- In the Page Data view locate **JSF Handler** → **doButtonClick()**.
- Drag and drop **doButtonClick()** on top of the command button (`Return to List Page`) in the Design tab of Page Designer. You should witness a balloon help indicating you are about to perform a bind function (Figure 15-26).



Drop here to bind "doButtonClick" to the control "button1"

Figure 15-26 Binding the command button to an EGL handler function

You can see the binding in two places:

- In the Source tab of Page Designer:

```
<hx:commandExButton type="submit" value="Return to List Page"
   id="button1" styleClass="commandExButton"
   action="#{CustomerDetailPage.doButtonClick}"
   actionListener="#{CustomerDetailPage._commandActionListener}">
</hx:commandExButton>
```

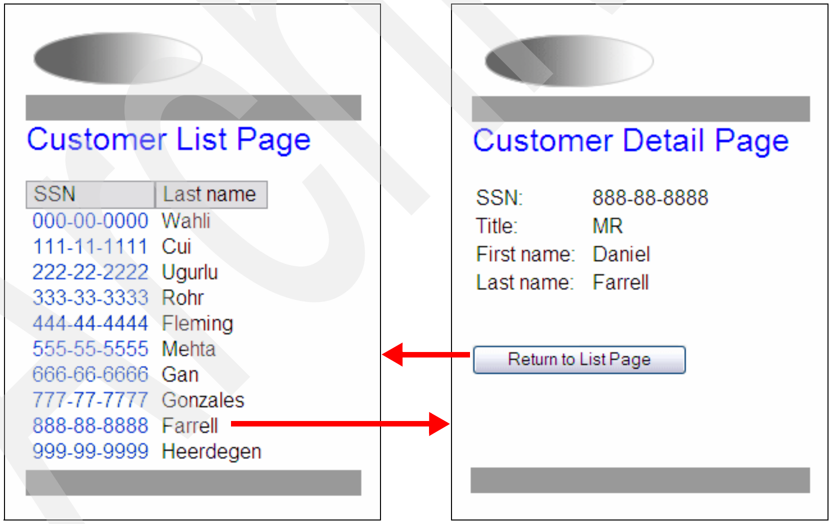
- In the Properties view of the command button—after clicking the icon ▤, which displays all the attributes in a table—look for the **action** attribute. Click the icon again to get back to the normal view.
- Save and close the JSP.

All that remains is to test the application. While we could invoke the second Web page, `CustomerDetailPage.jsp` directly, the table would be empty because no SSN is passed to the page. It is easier to always run the list page, `CustomerListPage.jsp`, and then go back and forth between pages.

Final test of the Web application

It is time to test the final version of this two page EGL Web site:

- In the Project Explorer right-click **EGLSource** and select **Generate**. This action compiles every EGL source file in this project. (Note that we already compiled the EGL code, but better be safe.)
- Right-click **CustomerListPage.jsp** and select **Run As** → **Run on Server**.
- In the Web browser with the customer list, click any of the links in the `SSN` column of the HTML table.
- This action causes the `CustomerDetailPage.jsp` to be displayed with the selected customer record. An entry is made to the Console view.
- Use the command button to return from the customer detail page to the customer list page.
- Figure 15-27 shows the run of the Web application.



Customer List Page

SSN	Last name
000-00-0000	Wahli
111-11-1111	Cui
222-22-2222	Ugurlu
333-33-3333	Rohr
444-44-4444	Fleming
555-55-5555	Mehta
666-66-6666	Gan
777-77-7777	Gonzales
888-88-8888	Farrell
999-99-9999	Heerdegen

Customer Detail Page

SSN:	888-88-8888
Title:	MR
First name:	Daniel
Last name:	Farrell

Figure 15-27 Final test of the Web application

Summary

With enough practice, all of the work that was exampled in this chapter could be performed in about 30 minutes. In this chapter we created a simple batch job, and a simple two page Web site. While all of this work was done in EGL, note that EGL outputs standard Java/J2SE and Java/J2EE. The EGL Web pages run on the same industrial strength application servers with ability for full fault tolerance and security as the commercial Web sites do.

To keep our discussion of EGL brief, the examples in this chapter only read data; they did not write data. Also, after the EGL batch job, we suppressed any error program recovery code when reading data. Writing data and having full program error recovery is easy in EGL. Hopefully these examples gave you enough to begin your mastery of EGL.

Implementing a detail page with update

As an exercise, we challenge you to implement a detail page to update customer information.

To implement a detail page with update capability is not much different from the current detail page. We provide only short instructions on how to accomplish this.

You can find the JSP and EGL code in the `C:\7501code\egl\update` folder.

Create a customer update page

The customer update page displays the customer with entry fields for changing the title, first name, and last name:

- Create a new page named `CustomerUpdatePage.jsp`. Follow the instructions for `CustomerDetailPage.jsp` to create the page and the title.
- Update the generated EGL handler code with two methods:

```
function doGobackClick()
   forward to "CustomerListPage";
end

function doUpdateClick()
   SysLib.writeStdout("Customer updating: " + customer.ssn);
   try
      replace customer with #sql{ update itso.customer
         set title = :customer.Title, firstname = :customer.Firstname,
            lastname = :customer.Lastname where ssn = :customer.Ssn };
   onException (e AnyException)
      SysLib.writeStdout("Error: " + e.messageid);
      SysLib.writeStdout("Error: " + e.message  );
```

```
    end
    forward to "CustomerListPage";
end
```

- The basic layout of the update page is shown in Figure 15-28.

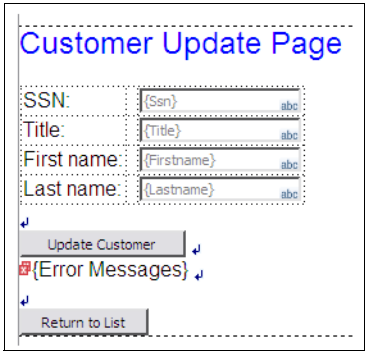


Figure 15-28 Customer Update Page

- Drop the customer bean from the Page Data view into the Design tab:
  - This time, select **Update an existing record**. All the fields are input fields.
  - Change the labels to nicer text.
  - Click **Options** and select **Submit button** with the name **Update Customer**.
  - Click **OK** to create the form with a table with input fields, and a button for updating under the table.
- Add a **Button - Command** and name it **Return to List**.
- Bind the **Update Customer** button to the **doUpdateClick** function. This invokes the SQL update statement.
- Bind the **Return to List** button to the **doGoback** function that returns to the customer list.
- Select the SSN input field, and in the Properties view, **h:inputText** → **Behavior** tab, select **Control is read-only** (the SSN cannot be updated). Note that we cannot use an output field, because we must have the SSN submitted with the other fields to have the key for the table update.

Change the customer list page

We add a link to the customer list page to invoke the update page and to redisplay the updated customer list:

- Drop a **Link** control onto the last name column value **{Lastname}**.
- Set the properties so that the link invokes **CustomerUpdatePage.faces**, with a parameter (the customer SSN value). Follow the instructions given for the link to the customer detail page.
- When the customer list is redisplayed we want to see the changes. Open `CustomerListPage.egl` and add **scope = request** to the handler. This forces a new retrieve of the customer list.

Recompile and test

Right-click **EGLSource** and select **Generate**.

Finally, run the `CustomerListPage.jsp` on the server. Click a last name in the customer list, make changes to the fields, and click **Update Customer**. If you change the last name the customer list displays the new name.

More information

For more information on EGL, refer to these resources:

- Redbooks publication: *Transitioning: Informix 4GL to Enterprise Generation Language (EGL)*, SG24-6673
- Redbooks publication: *Legacy Modernization with WebSphere Studio Enterprise Developer*, SG24-6806
- Whitepaper: *Exploiting Enterprise Generation Language for Business Oriented Developers*:

  ftp://ftp.software.ibm.com/software/rational/web/whitepapers/G507-0995-01-bodev.pdf

- IBM developerWorks articles on Enterprise Generation Language:

  http://www.ibm.com/developerworks/rational/products/egl/
  http://www.ibm.com/developerworks/rational/products/egl/egldoc.html
  http://www.ibm.com/developerworks/rational/library/05/510_java/
  http://www.ibm.com/developerworks/rational/library/07/0126_sayles/

- Wikipedia:

  http://en.wikipedia.org/wiki/Enterprise_Generation_Language

- DevX.com:

  http://www.devx.com/ibm/Article/31574

16

Develop Web applications using EJBs

This chapter introduces Enterprise JavaBeans (EJB) and demonstrates by example how to create, maintain, and test such components in the J2EE platform.

We describe how to develop entity beans, and explain the relationships between the entity beans and session beans. Then we integrate the EJBs with a front-end Web application for the sample application. We include examples for creating, developing, and testing the EJBs using Rational Application Developer.

The chapter is organized into the following sections:

- Introduction to Enterprise JavaBeans
- Sample application overview
- Preparing for the sample
- Developing an EJB application
- Testing EJBs with the Universal Test Client
- Adapting the Web application
- More information

Introduction to Enterprise JavaBeans

Enterprise JavaBeans (EJB) is an architecture for server-side component-based distributed applications written in Java.

> **Note:** This chapter provides a condensed description of the EJB architecture and several coding examples. For more complete coverage on EJBs, refer to “More information” on page 792 at the end of this chapter, and see *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819. Although Application Developer includes support for EJB 2.1, much of the information is still relevant.

Enterprise JavaBeans overview

Since its introduction in December 1999, the technology has gained momentum among platform providers and enterprise development teams. This is because the EJB component model simplifies the development of business components that are:

- **Secure**: Enterprise JavaBeans allow the declaration of method-level security rules for any bean. Users and user groups can be granted or denied execution rights to any bean or method. In WebSphere, these same user groups can be granted or denied access to Web resources, and the user IDs can be in a seamless way passed from the Web resources to the EJBs by the underlying security framework. Not only that, but the authenticated credentials can also be forwarded to other systems, possibly legacy systems (compatible LTPA clients).
- **Distributed**: Enterprise JavaBeans automatically provide distribution capabilities to your application, allowing for the building of enterprise-scale systems. Your system’s modules can be deployed to many different physical machines and many separate OS processes to achieve your performance, scalability, and availability requirements.
- **Persistent**: Making an object persistent means preserving its persistent state (the values of its non-transient variables) even after the termination of the system that created that object.

  In most cases, the state of a persistent object is stored in a relational database. Unfortunately, the object-oriented (OO) and relational paradigms differ a lot from each other. Relational models are less expressive than OO models because they provide no way to represent behavior, encapsulation, or complex relationships like inheritance. Additionally, SQL data types do not exactly match Java data types, leading to conversion problems. All these problems can be automatically solved when using EJBs.

- **Transactional**: Enterprise beans support multiple concurrent transactions with commit and rollback capabilities across multiple data sources in a full two-phase commit-capable environment for distributed transactions.
- **Scalable**: The 24-hour, 7-day-a-week nature of the Web has also made uptime a crucial issue for businesses. However, not everyone needs a system designed for 24x7 operation or that is able to handle millions of concurrent users. We should be able to design a system so that scalability can be achieved without sacrificing ease of development, or standardization.

  WebSphere's EJB support can provide this kind of highly scalable, highly available system. It does so by utilizing the following features:

  – Object caching and pooling: WebSphere Application Server automatically pools enterprise beans at the server level, reducing the amount of time spent in object creation and garbage collection.

  – Workload optimization: WebSphere Application Server Network Deployment provides advanced EJB workload optimization features. Servers can be grouped in clusters and then managed together using a single administration facility.

  – Automatic fail-over support: With several servers available in a cluster to handle requests, it is less likely that occasional hardware and software failures produce throughput and reliability issues. In a clustered environment, tasks are assigned to servers that have the capacity to perform the task operations. If one server is unavailable to perform the task, it is assigned to another cluster member. No changes to the code are necessary to take advantage of these features.
- **Portable**: A strategic issue for businesses nowadays is achieving platform and vendor independence. The EJB architecture, which is an industry standard, can help achieve this goal. EJBs developed for the J2EE platform can be deployed to any compliant application servers.

The EJB architecture shown in Figure 16-1 reduces the complexity of developing business components by providing automatic support for such system level services, thus allowing developers to concentrate on the development of business logic. Such focus can bring a competitive advantage to a business.

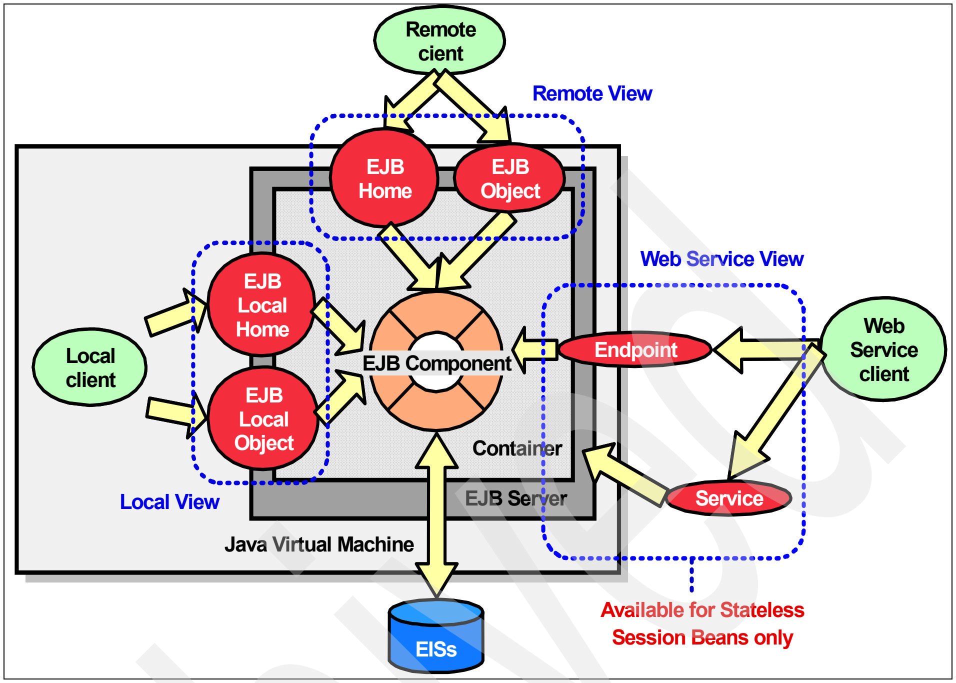


Figure 16-1 EJB architecture overview

In the following sections we briefly explain each of the EJB architecture elements depicted in:

- EJB server
- EJB container
- EJB components

EJB server

An EJB server is the part of an application server that hosts EJB containers. It is also called an Enterprise Java Server (EJS). WebSphere Application Server is an EJS.

The EJB server provides the implementation for the common services available to all EJBs. The EJB server's responsibility is to hide the complexities of these services from the component requiring them. The EJB specification outlines eight services that must be provided by an EJB server:

- Naming
- Transaction
- Security

- Persistence
- Concurrency
- Life cycle
- Messaging
- Timer

EJB container

The EJB container functions as a runtime environment for enterprise beans by managing and applying the primary services that are needed for bean management at runtime. In addition to being an intermediary to the services provided by the EJB server, the EJB container will also provide for EJB instance life cycle management and EJB instance identification. EJB containers create bean instances, manage pools of instances, and destroy them.

Containers are transparent to the client in that there is no client API to manipulate the container, and there is no way for a client to tell in which container an enterprise bean is deployed.

One of the container's primary responsibilities is to provide the means for remote clients to access components that live within them. Remote accessibility enables remote invocation of a native component by converting it into a network component. EJB containers use the Java RMI interfaces to specify remote accessibility to clients of the EJBs.

The responsibilities that an EJB container must satisfy can be defined in terms of the primary services. Specific EJB container responsibilities are as follows:

- **Naming**: The container is responsible for registering the unique lookup name in the JNDI namespace when the server starts up, and binding the appropriate object type into the JNDI namespace.
- **Transaction**: The EJB container can handle the demarcation of transactions automatically, depending on the EJB type and the transaction type attribute, both described in the EJB module's deployment descriptor. When the container demarcates the transactions, applications can be written without explicit transaction demarcation code (for example, begin, commit, rollback).
- **Security**: The container provides security realms for enterprise beans. It is responsible for enforcing the security policies defined at the deployment time whenever there is a method call, through access control lists (ACL). An ACL is a list of users, the groups they belong to, and their rights, and it ensures that users access only those resources and perform those tasks for which they have been given permission.

- **Persistence**: The container is also responsible for managing the persistence of a certain type of bean (discussed later in this chapter) by synchronizing the state of the bean's instance in memory with the respective record in the data source.
- **Concurrency**: The container is responsible for managing the concurrent access to components, according to the rules of each bean type.
- **Life cycle**: The container controls the life cycle of the deployed components. As EJB clients start sending requests to the container, the container dynamically instantiates, destroys, and reuses the beans as appropriate. The container can ultimately provide for some resource utilization optimizations, and employ techniques for bean instance pooling.
- **Messaging**: The container must provide for the reliable routing of asynchronous messages from messaging clients (JMS or otherwise) to message-driven beans (MDBs). These messages can follow either the peer-to-peer (queue-based) or publish/subscribe (topic-based) communication patterns.
- **Timer**: Enterprise applications can model business processes that are dependent on temporal events. To implement this characteristic, the container must provide a reliable and transactional EJB Timer Service that allows callbacks to be scheduled for time-based events. Timer notifications can be scheduled to occur at a specific time, after a specific elapsed duration, or at specific recurring intervals.

Note the similarity here to the list in "EJB server" on page 722. This is due to the unspecified division of responsibilities between the EJB server and container.

EJB components

EJB components represent the actual EJBs themselves and include all the classes, interfaces, and constructs that make up the enterprise bean. We distinguish between:

- EJB types
- EJB interfaces and classes
- EJB client view
- EJB relationships
- EJB query language

EJB types

There are three types of enterprise beans: entity, session, and message-driven beans (Figure 16-2).

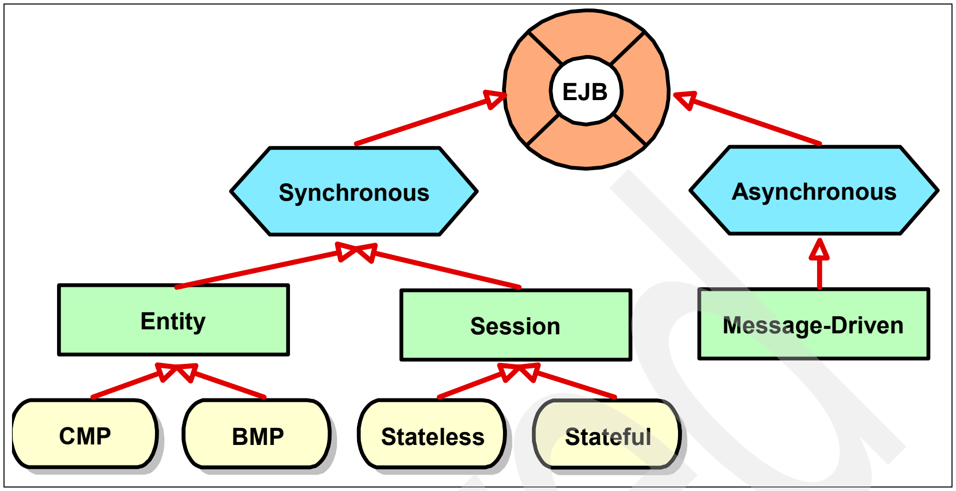


Figure 16-2 EJB types

- **Entity beans**: Entity beans are modeled to represent business or domain specific concepts, and are typically the nouns of your system, such as *customer* and *account*. Entity beans are persistent; that is, they maintain their internal state (attribute values) between invocations and across server restarts. Due to their persistent nature, entity beans usually represent data (entities) stored in a database.

  While the container determines *when* an entity bean is stored in persistent storage, *how* the bean is stored can either be controlled by the container, through *container-managed persistence* (CMP), or by the bean itself, through *bean-managed persistence* (BMP). Container-managed persistence is typically obtained by defining the mapping between the fields of the entity bean and columns in a relational database to the EJB server.

- **Session beans**: A session bean is modeled to represent a task or workflow of a system, and to provide coordination of those activities. It is commonly used to implement the facade of EJB modules. Although some session beans can maintain state data, this data is not persistent, it is just conversational.

  Session beans can either be *stateless* or *stateful*. Stateless session beans are beans that maintain no conversational state, and are pooled by the container to be reused. Stateful session beans are beans that keep track of the conversational state with a specific client. Thus, they cannot be shared among clients.

- **Message-driven beans (MDB)**: Like session beans, message-driven beans can also be modeled to represent tasks. However, they are invoked by the receipt of asynchronous messages, instead of synchronous ones. The bean either listens for or subscribes to messages that it is to receive.

Entity and session beans are accessed synchronously through a remote or local EJB interface method invocation. This is referred to as synchronous invocation, because when a client makes an invocation request, it will be blocked, waiting for the return. Clients of EJBs invoke methods on session and entity beans. An EJB client can be remote, such as a servlet, or local, such as another EJB within the same JVM.

Message-driven beans are not accessible through remote or a local interfaces. The only way for an EJB client to communicate with a message-driven bean is by sending a JMS message. This is an example of asynchronous communication. The client does not invoke the method on the bean directly, but rather, uses JMS constructs to send a message. The container delegates the message to a suitable message-driven bean instance to handle the invocation. EJBs of any type can also be accessed asynchronously by means of a timer event, fired by the EJB Timer Service.

EJB interfaces and classes

An EJB component consists of the following primary elements, depending on the type of bean:

- **EJB bean class**: This contains the bean's business logic implementation. Bean classes must implement one of the enterprise bean interfaces, depending on the bean type: `javax.ejb.SessionBean`, `javax.ejb.EntityBean`, `javax.ejb.MessageDrivenBean`. Beans willing to be notified of timer events must implement the `javax.ejb.TimedObject`, so that the container can call the bean back when a timer expires. Message-driven beans must also implement the javax.jms.MessageListener interface, to allow the container to register the bean as a JMS message listener and to call it back when a new message arrives.
- **EJB component interface**: This declares which of the bean's business methods should be exposed to the bean's public interface. Clients will use this interface to access such methods. Clients cannot access methods that are not declared in this interface. A bean can implement a local component interface, a remote component interface, or both, depending on the kinds of clients it expects to serve.
- **EJB home interface**: This declares which bean's life cycle methods (to create, find, and remove beans instances) are available to clients, functioning very much like a factory. Local beans have local home interfaces that extend `javax.ejb.EJBLocalHome`. Remote beans have remote home interfaces that extend `javax.ejb.EJBHome`.
- **Primary key class**: Entity beans must also have a primary key class. Instances of this class uniquely identify an instance of the entity type in the database. Even though not formally enforced, primary key classes must also

correctly implement the equals and hash code methods. As you will see, Application Developer takes care of that for you.

EJB client view

For client objects to send messages to an EJB component, the component must provide a view. A view is a client interface to the bean, and can be local or remote:

- A local interface can be used only by local clients (clients that reside in the same JVM as the server component) to access the EJB.
- A remote interface allows any client (possibly distributed) to access the component.

The motivation for local interfaces is that remote calls are more expensive than local calls. Which one to use is influenced by how the bean itself is to be used by its clients, because local and remote depict the clients' view of the bean. An EJB client can be a remote client, such as a servlet running on another process, or can be a local client, such as another EJB in the same container.

Even though a component can expose both a local and a remote view at the same time, this is typically not the case. EJBs that play the role of facades usually offer only a remote interface. The rest of the components generally expose only a local interface.

In remote invocation, method arguments and return values are passed by value. This means that the complete objects, including their non-transient reference graphs, have to be serialized and sent over the network to the remote party, which reconstructs them as new objects. Both the object serialization and network overhead can be a costly proposition, ultimately reducing the response time of the request.

On the other hand, remote interfaces have the advantage of being location independent. The same method can be called by a client that is inside or outside of the container.

Additionally, Web service clients can access stateless session beans through the Web service client view. The view is described by the WSDL document for the Web service the bean implements, and corresponds to the bean's Web Service endpoint interface.

Relationships

Relations are a key component of object-oriented software development. Non-trivial object models can form complex networks with these relationships.

The container automatically manages the state of CMP entity beans. This management includes synchronizing the state of the bean with the underlying database when necessary and also managing any container-managed relationships (CMRs) with other entity beans. The bean developer is relieved of the burden that is writing database-specific code and, instead, can focus on business logic.

Multiplicity is an important characteristic of relations. Relations can have the following multiplicities:

- **One-to-one:** In a one-to-one (1:1) relationship, a CMP entity bean is associated with a single instance of another CMP entity bean, and vice versa.
- **One-to-many:** In a one-to-many (1:m) relationship, a CMP entity bean is associated with multiple instances of another CMP entity bean. For example, an `Account` bean could be associated with multiple instances of a `Transaction` bean, kept as a log of transactions.
- **Many-to-many:** In a many-to-many (m:m) relationship, multiple instances of a CMP entity bean are associated with multiple instances of another CMP entity bean**.** For example, a `Customer` bean can be associated with multiple instances of an `Account` bean, and a single `Account` bean can, in turn, be associated with many `Customer` beans.

There are also three different types of relationships:

- **Association:** An association is a loose relationship between two independent objects.
- **Aggregation:** Aggregation identifies that an object is made up of separate parts. That is, the aggregating object is dependent on the aggregated objects. The life time of the aggregated objects is not controlled by the aggregator. If the aggregating object is destroyed, the aggregated objects are not necessarily destroyed.
- **Composition:** Composition defines a stronger dependency between the objects. Composition is similar to aggregation, but with composition, the life time of the objects that make up the whole are controlled by the compositor.

It is the developer's task to implement the differences among the three kinds of relationships. These differences can require considerations of characteristics, such as the navigation of the relationship and the encapsulation of the related objects.

Component-level inheritance is not part of the EJB 2.1 specification, even though it is planned for future releases. In want of standardized component-level inheritance, IBM WebSphere Application Server V6.1 and Application Developer implements proprietary component-level inheritance.

EJB query language (EJB QL)

The EJB query language is a query specification language, similar to SQL, for entity beans with container-managed persistence. With it, the bean provider is able to specify the semantics of EJB custom finder or EJB select methods in a portable way and in terms of the object model's entities, instead of the relational model's entities. This is possible because EJB QL is based on the abstract schema types of the entity beans.

> **Note:** Both finder and EJB select methods are used to query the back-end where the actual data is stored. The difference is that finder methods are accessible to clients, whereas select methods are internal to the implementation and are not visible to clients.

An EJB QL query is a string consisting of the following clauses:

- A SELECT clause, which determines the type of the objects or values to be selected.
- A FROM clause, which provides declarations that designate the domain to which the specified expressions apply.
- An optional WHERE clause, which can be used to restrict the results that are returned by the query.
- An optional ORDER BY clause, which can be used to order the results that are returned by the query.
- The result type can be an `EJBLocalObject`, an `EJBObject`, a CMP-field value, a collection of any of these types, or the result of an aggregate function.

EJB QL queries are defined by the bean provider in the deployment descriptor. The SQL statements for the actual database access is generated automatically by the deployment tooling. As an example, this query retrieves customers that have accounts with a large balance:

```
select object(c) from Customer c, in(c.accounts) a where a.balance > ?1
```

As you can see, this EJB QL statement is independent of the database implementation. It follows a CMR relationship from customer to account and queries the account balance. Finder and select methods specified using EJB QL are portable to any EJB 2.1 environment.

EJB features in Application Developer

The following features, supported by Application Developer, are for the EJB 2.1 specification and require a J2EE 1.4 compatible application server, such as WebSphere Application Server V6.1:

- Ability to create enterprise beans and access beans is provided.
- Ability to build data persistence into enterprise beans is provided.
- Ability to generate deployment code is provided.
- Ability to automatically validate the enterprise beans for specification compliance is provided.
- Stateless session beans can implement a Web service endpoint.
- Enterprise beans of any type can utilize external Web services.
- The container-managed timer service is provided.
- Message-driven beans support more messaging types in addition to JMS.
- The EJB query language has been enhanced to include support for aggregate functions, ordering of results, and additional scalar functions. Also, the rules for allowing null values to be returned by finder and select methods have been clarified.

Sample application overview

In this chapter, we reuse the design of the application, described in Chapter 7, “Develop Java applications” on page 227 with same small changes. However, the content of this chapter does not strictly depend on it. You can complete the sample in this chapter without knowledge of the sample developed in the Java chapter.

The focus of this chapter is on implementing EJBs for the business model, instead of regular JavaBeans. The rest of the application’s layers (control and view) still apply as designed.

Figure 16-3 shows the sample application model layer design.

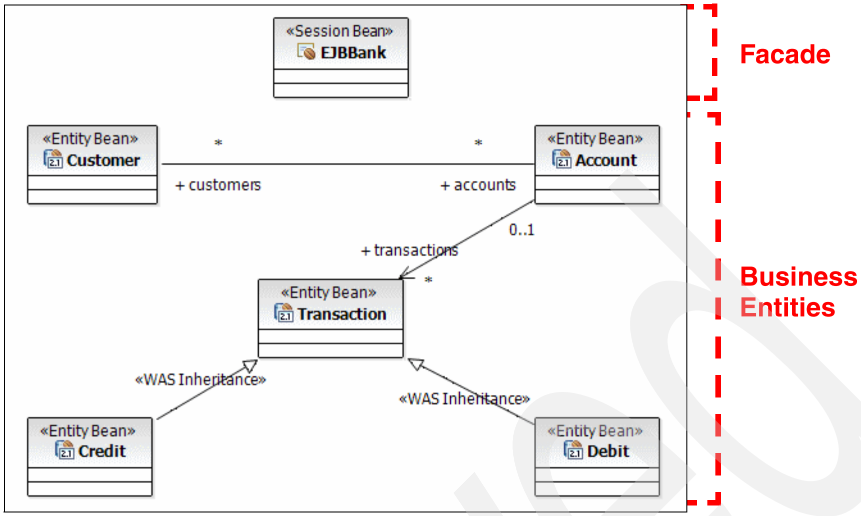


Figure 16-3 EJB module class diagram for the sample application

The **`EJBBank`** session bean acts as a facade for the EJB model. Our business entities (`Customer`, `Account`, `Transaction`, `Credit`, and `Debit`) are implemented as CMP entity beans with local interfaces, as opposed to regular JavaBeans. By doing so, we automatically gain persistence, security, distribution, and transaction management services. On the other hand, this also implies that the control and view layers are not able to reference these entities directly, because they can be placed in a different JVM. Only the session bean (`EJBBank`) can access the business entities through their local interfaces.

You might be asking yourself, then, why we do not expose a remote interface for the entity beans as well? The problem with doing that is two-fold. First, in such a design, clients would probably make many remote calls to the model to resolve each client request. This is not a recommended practice because remote calls are more expensive than local ones. Finally, allowing clients to see into the model breaks the layer's encapsulation, promoting unwanted dependencies and coupling.

Because the control layer is not able to reference the model objects directly, we reuse the `Customer`, `Account`, `Transaction`, `Credit`, and `Debit` from the Java application in Chapter 7, "Develop Java applications" on page 227as data transfer objects, carrying data to the servlets and JSPs, but allowing no direct access to the underlying model.

Figure 16-4 shows the application component model and the flow of events.

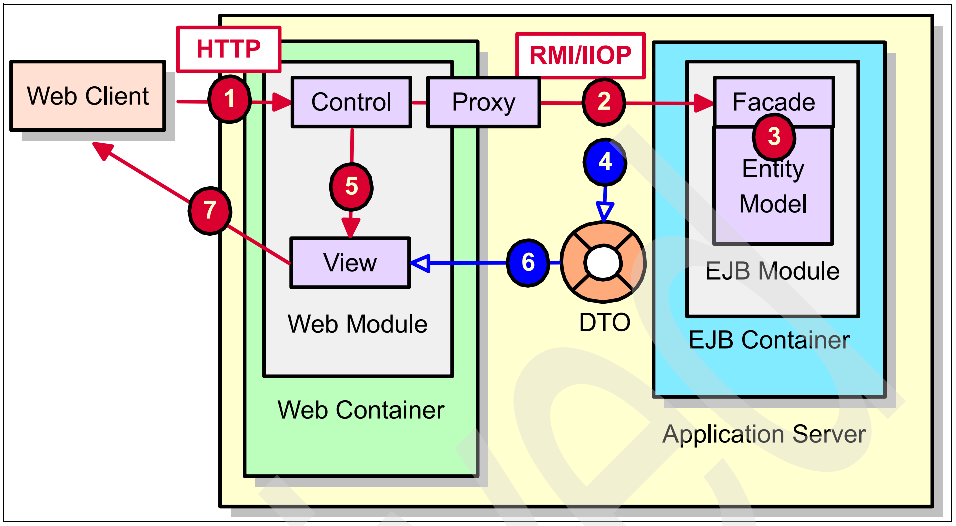


Figure 16-4 Application component model and workflow

The flow of events, as shown in Figure 16-4, is as follows:

1. The first event that occurs is the HTTP request issued by the Web client to the server. This request is answered by a servlet in the control layer, also known as the front controller, which extracts the parameters from the request. The servlet sends the request to the appropriate control JavaBean. This bean verifies whether the request is valid in the current user and application states.
2. If so, the control layer sends the request through the JavaBean proxy to the session EJB facade. This involves using JNDI to locate the session bean's home interface and creating a new instance of the bean.
3. The session EJB executes the appropriate business logic related to the request. This includes having to access entity beans in the model layer.
4. The facade creates a new DTO and populates it with the response data. The DTO is returned to the calling controller servlet.
5. The front controller servlet sets the response DTO as a request attribute and forwards the request to the appropriate JSP in the view layer, responsible for rendering the response back to the client.
6. The view JSP accesses the response DTO to build the user response.
7. The result view, possibly in HTML, is returned to the client.

Preparing for the sample

This section describes the steps to prepare for developing the sample EJB application.

Required software

To complete the EJB development sample in this chapter, you must have the following software installed:

- IBM Rational Application Developer V7.0
- Database software, either of these products:
  - Derby V10.1 (installed by default with Application Developer)
  - IBM DB2 Universal Database V8.2

> **Note:** For more information on installing the software, refer to Appendix A, “Product installation” on page 1281.

Enabling the EJB development capability

To develop EJBs, we have to enable the EJB development capability in Rational Application Developer:

- Select **Window** → **Preferences**.
- Select **General** → **Capabilities** → **Enterprise Java Developer** and click **OK**.

Creating and configuring the EJB projects

In Application Developer, you create and maintain Enterprise JavaBeans and associated Java resources in EJB projects. The environment has facilities that help you create all types of EJBs, define relationships, and create resources such as access beans, converters, and composers. Within an EJB project, these resources can be treated as a portable, cohesive unit.

> **Note:** Converters and composers are used for non-standard relational mapping. A converter allows you to transform a user-defined Java type to an SQL type back and forth. Composers are used when entity attributes have multi-column relational representations.

An EJB module typically contains components that work together to perform some business logic. This logic can be self-contained, or access external data and functions as needed. It should be comprised of a facade and the business entities. The facade is usually implemented using one or more remote session beans and message-driven beans. The model is commonly implemented with related local entity beans.

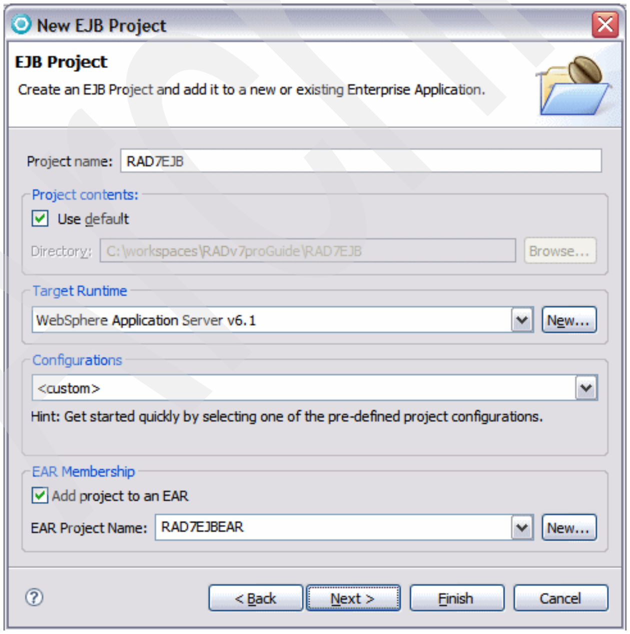
In this chapter we develop the entity beans shown in Figure 16-3 on page 731.

Creating an EJB project

To develop EJBs, you have to create an EJB project. You can also create an EJB project and an EJB client project, which holds the interfaces and classes required for a client accessing the EJBs. It is also typical to create an enterprise application (EAR) project that is the container for deploying the EJB project.

To create a J2EE EJB project, do these steps:

- Open the J2EE perspective.
- In the Workbench, select **File** → **New** → **Project**.
- In the New Project dialog, select **EJB** → **EJB Project** and click **Next**.
- Refer to the New EJB Project dialog (Figure 16-5).


New EJB Project

EJB Project

Create an EJB Project and add it to a new or existing Enterprise Application.

Project name: RAD7EJB

Project contents:

☑ Use default

Directory: C:\workspaces\RADv7proGuide\RAD7EJB

Target Runtime

WebSphere Application Server v6.1

Configurations

<custom>

Hint: Get started quickly by selecting one of the pre-defined project configurations.

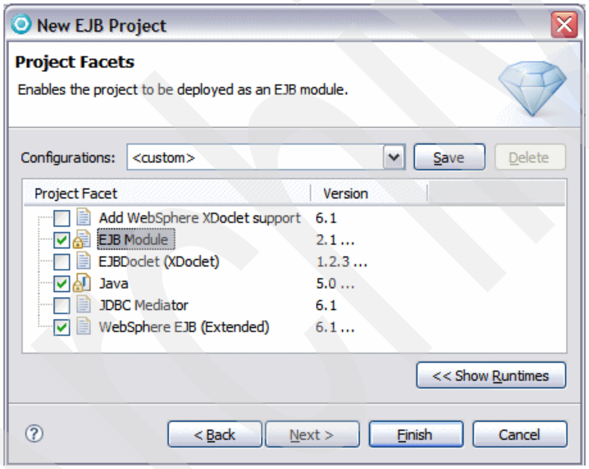
EAR Membership

☑ Add project to an EAR

EAR Project Name: RAD7EJBEAR

Figure 16-5 Create an EJB project wizard (1)

- In the New EJB Project dialog (Figure 16-5):
  - Enter **RAD7EJB** in the Name field.
  - Select **Add project to an EAR** (default) and enter **RAD7EJBEAR** for EAR Project Name field. Default behavior is to create a new EAR project, but you can also select an existing project from the drop-down combo box. If you would like to create a new project and also configure its location, click **New**. For our example, we use the given default value.
  - Additional advanced options are also displayed:
    - Target server: Select **WebSphere Application Server V6.1**.
    - Configurations: optionally: EJB Project with XDoclet
  - Click **Next**.
- In the Project Facets dialog (Figure 16-6), accept the default and click **Next**:
  - EJB Module: **2.1** (default)
  - Java: **5.0** (default)
  - WebSphere EJB (Extended): **6.1** (default)


New EJB Project

Project Facets

Enables the project to be deployed as an EJB module.

Configurations: <custom>

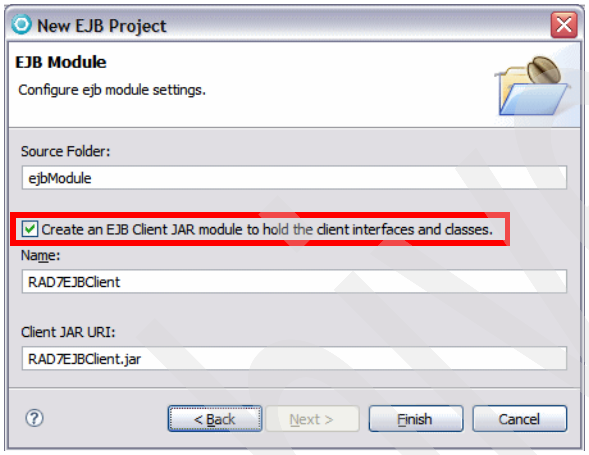
Project Facet	Version
□ Add WebSphere XDoclet support	6.1
☑ EJB Module	2.1 ...
□ EJBDoclet (XDoclet)	1.2.3 ...
☑ Java	5.0 ...
□ JDBC Mediator	6.1
☑ WebSphere EJB (Extended)	6.1 ...

Figure 16-6 Create an EJB project wizard (2)

- In the EJB Module dialog (Figure 16-7) enter the following items:
  - Source folder: `ejbModule` (default)
  - Select **Create an EJB Client JAR Project to hold client interfaces and classes** (default).
  - Name: `RAD7EJBClient`
  - Client JAR URI: `RAD7EJBClient.jar`
  - Click **Finish**.

Note: The EJB client jar holds the home and component interfaces of the enterprise beans, and other classes that these interfaces depend on, such as their superclasses and implemented interfaces, the classes and interfaces used as method parameters, results, and exceptions.

The EJB client jar can be deployed together with a client application that accesses the EJBs. This results in a smaller client application as compared to deploying the EJB project with the client application.



EJB Module

Configure ejb module settings.

Source Folder: ejbModule

☑ Create an EJB Client JAR module to hold the client interfaces and classes.

Name: RAD7EJBClient

Client JAR URI: RAD7EJBClient.jar

Figure 16-7 Create an EJB project wizard (3)

- If the current perspective is not the J2EE perspective when you create the project, Application Developer prompts, asking if you want to switch to the J2EE perspective. Click **Yes**.

Important: After creating the EJB project (`RAD7EJB`) you see an error in the Problems view saying *An EJB module must contain one or more enterprise beans.* An EJB project must contain at least one enterprise bean. The error is corrected automatically when we create an enterprise bean in the later steps.

Importing a Web project and a Java project

To finish the sample EJB application, we use a Web front-end application. We renamed the Web application developed in Chapter 12, “Develop Web applications using JSPs and servlets” on page 465 to `RAD7EJBWeb` to make the examples more independent. However, the architecture of the Web application is still the same.

Do these steps to import the RAD7EJBWeb and associated projects from the additional material:

- From the Workbench, select **File** → **Import** → **Other** → **Project Interchange** and click **Next**.
- In the Import Projects dialog, select the c:\7501code\ejb\RAD7EJBWeb.zip file and select all three projects (RAD7EJBJava, RAD7EJBWeb, and RAD7EJBWebEAR) from the project list. Click **Finish**.

Make the Java DTOs available to the EJB project

The RAD7EJBJava project is a stripped down version of the RAD7Java project developed in Chapter 7, "Develop Java applications" on page 227. It contains simple JavaBeans that are used as data transfer objects (DTO) between the layers of the application.

We have to add the RAD7EJBJava project to the EJB project and the enterprise application so that we can reference the data objects and exceptions defined in the Java project from the EJB project.The Java project has the actual data model of the sample application. To configure the dependencies do these steps:

- Open the deployment descriptor of the RAD7EJBEAR project:
  - Select the **Module** tab and in the Project Utility JARs section (where RAD7EJBClient.jar is listed) click **Add**. Select the RAD7EJBJava project and click **Finish**.
  - Save and close the deployment descriptor.
- Right-click **RAD7EJB** and select **Properties**:
  - In the Properties dialog, select **J2EE Module Dependencies**.
  - Select RAD7EJBJava.jar and click **OK**.
- Right-click **RAD7EJBClient** and select **Properties**:
  - In the Properties dialog, select **J2EE Module Dependencies**.
  - Select RAD7EJBJava.jar and click **OK**.

Setting up the sample database

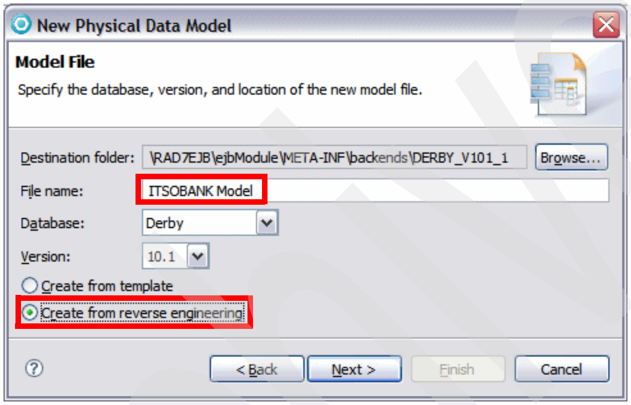
The entity EJBs are based on the ITSOBank database. Therefore, we have to define a database connection within Application Developer that the mapping tools use to extract schema information from the database.

Refer to "Setting up the ITSOBANK database" on page 1312 for instructions on how to create the ITSOBANK database. For the EJBs we can either use the DB2 or Derby database. For simplicity we use the built-in Derby database in this chapter.

Import the physical data model

We have to import the physical data model (database schema) into the EJB project. This has to be done for the mapping tools to be able to map the EJBs to the database tables. Follow these steps:

- Select **RAD7EJB** → **ejbModule** → **META-INF** from the Project Explorer.
- Select **File** → **New** → **Other** → **Data** → **Physical Data Model** and click **Next**.
- In the Model File dialog (Figure 16-8), do these steps and click **Next**:
  - File name: **ITSOBANK Model**
  - Database: **Derby**
  - Version: **10.1**
  - Select **Create from reverse engineering**



Model File

Specify the database, version, and location of the new model file.

Destination folder: \RAD7EJB\ejbModule\META-INF\backends\DERBY_V101_1

File name: ITSOBANK Model

Database: Derby

Version: 10.1

○ Create from template

● Create from reverse engineering

Figure 16-8 Physical Data Model: Database

- In the Select Connection dialog, select **Create a new connection** and click **Next**.

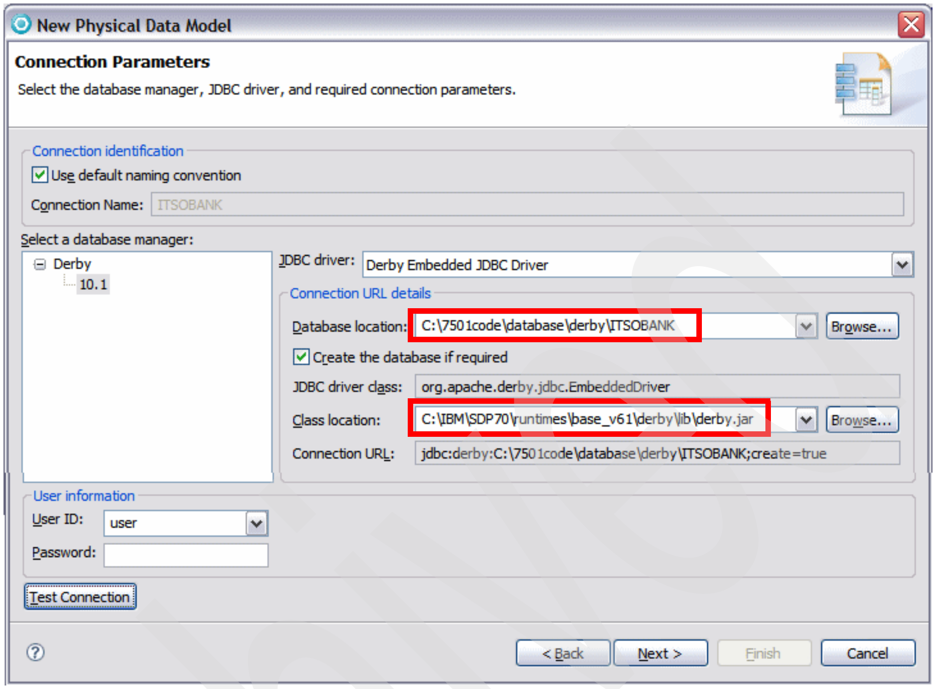
> **Tip:** You can also reuse the `ITSOBANK` connection from Chapter 9, “Develop database applications” on page 355.

- In the Connections Parameters dialog (Figure 16-9):
  - Accept the connection name of `ITSOBANK`.
  - Click **Browse** for Database location and locate:

    `C:\7501code\database\derby\ITSOBANK`
  - Click **Browse** for Class location and locate:

    `C:\<RAD_HOME>\runtimes\base_v61\derby\lib\derby.jar`
  - Enter any user ID.

– Click **Test Connection** to verify that the connection works, and click **Next**.



New Physical Data Model

Connection Parameters

Select the database manager, JDBC driver, and required connection parameters.

Connection identification

☑ Use default naming convention

Connection Name: ITSOBANK

Select a database manager:

- Derby
  - 10.1

JDBC driver: Derby Embedded JDBC Driver

Connection URL details

Database location: C:\7501code\database\derby\ITSOBANK

☑ Create the database if required

JDBC driver class: org.apache.derby.jdbc.EmbeddedDriver

Class location: C:\IBM\SDP70\runtimes\base_v61\derby\lib\derby.jar

Connection URL: jdbc:derby:C:\7501code\database\derby\ITSOBANK;create=true

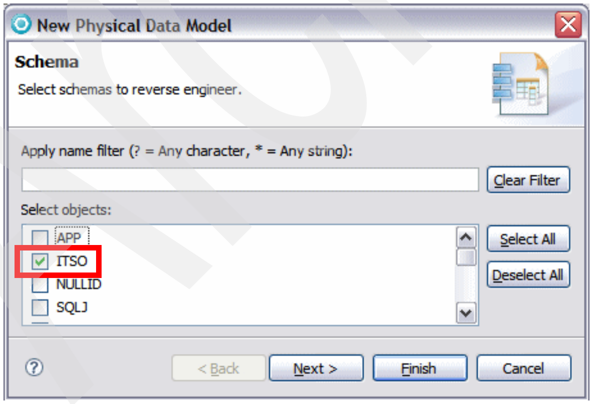
User information

User ID: user

Password:

Figure 16-9 Physical Data Model: Connection to the Derby ITSOBANK

► In the Schema dialog, select **ITSO** (Figure 16-10).



New Physical Data Model

Schema

Select schemas to reverse engineer.

Apply name filter (? = Any character, * = Any string):

Select objects:

□ APP
☑ ITSO
□ NULLID
□ SQLJ

Figure 16-10 Physical Data Model: Schema

► Click **Next** to go through the rest of the pages or click **Finish**.

- Notice the database and tables imported into the `RAD7EJB` project (Figure 16-11). There is also a `META-INF\backends\DERBY_V101_1` folder.

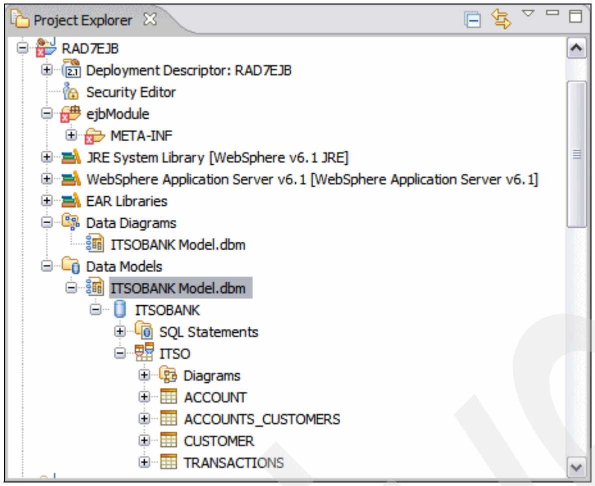


Figure 16-11 Physical Data Model: Imported database with tables

- Close the Physical Data Model editor.

Configuring the data source for the ITSOBANK

There are a couple of methods that can be used to configure the data source, including using the WebSphere Administrative Console or using the WebSphere enhanced EAR, which stores the configuration in the deployment descriptor and is deployed with the application.

The definition of the data source in the WebSphere Administrative Console is covered in “Configuring the data source in WebSphere Application Server” on page 1313.

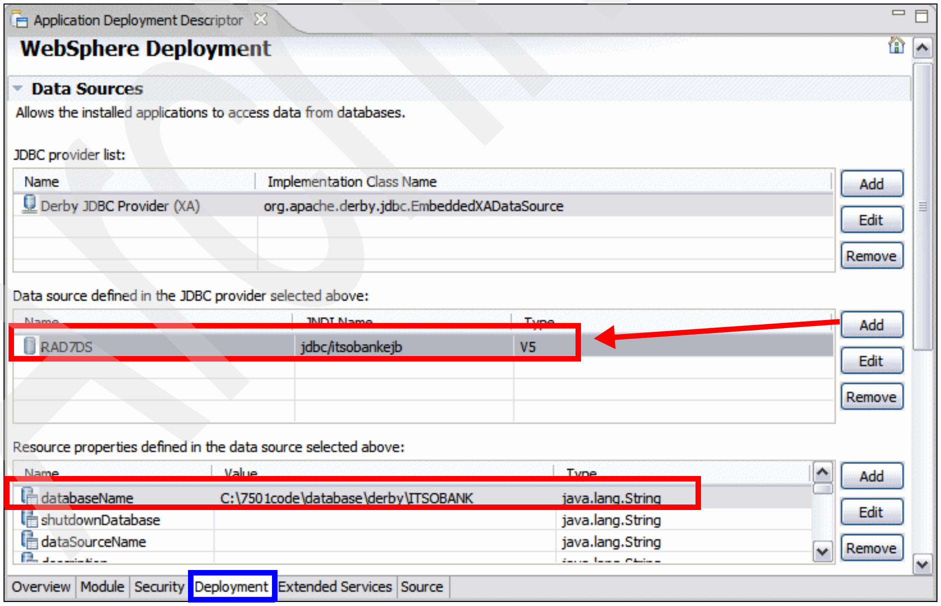
This section describes how to configure the data source using the WebSphere enhanced EAR capabilities. The enhanced EAR is configured in the Deployment tab of the EAR Deployment Descriptor editor. If you select to import the complete sample code, you only have to verify that the value of the `databaseName` property in the deployment descriptor matches the location of the database.

Note: For more information on configuring data sources and general deployment issues, refer to Chapter 24, “Deploy enterprise applications” on page 1103.

Configure the data source using enhanced EAR

To configure a new data source using the enhanced EAR capability in the deployment descriptor, do these steps:

- Open the deployment descriptor of the **RAD7EJBEAR** project and select the **Deployment** tab.
- Select **Derby JDBC Provider (XA)** from the JDBC provider list. This JDBC Provider is configured by default.
- Click **Add** next to data source.
- In the Create a Data Source dialog:
  - Select **Derby JDBC Provider (XA)** under the JDBC provider and **Version 5.0 data source**, and click **Next**.
  - Enter `RAD7DS` (Name), `jdbc/itsobankejb` (JNDI name), `Data Source for ITSOBANK EJBs` (Description). Select **Use this data source in container managed persistence (CMP)**. Click **Next**.
- In the Create Resource Properties dialog, select **databaseName** and enter the value `C:\7501code\database\derby\ITSOBANK`. Clear the description.
- Click **Finish**.
- Save and close the deployment descriptor (Figure 16-12).



WebSphere Deployment

Data Sources

Allows the installed applications to access data from databases.

JDBC provider list:

Name	Implementation Class Name
Derby JDBC Provider (XA)	org.apache.derby.jdbc.EmbeddedXADataSource

Data source defined in the JDBC provider selected above:

Name	JNDI Name	Type
RAD7DS	jdbc/itsobankejb	V5

Resource properties defined in the data source selected above:

Name	Value	Type
databaseName	C:\7501code\database\derby\ITSOBANK	java.lang.String
shutdownDatabase		java.lang.String
dataSourceName		java.lang.String
description		java.lang.String

Figure 16-12 EAR enhanced deployment descriptor

Set up the default CMP data source

Several data sources can be defined for an enterprise application. For the EJB container to be able to determine which data source should be used, we must configure the RAD7EJB project to point to the correct data source as follows:

- Open the deployment descriptor of the **RAD7EJB** project.
- On the Overview tab, scroll down to **WebSphere Bindings** → **JNDI - CMP Connection Factory Binding**:
  - Enter jdbc/itsobankejb in the JNDI name field.
  - Select **Per_Connection_Factory** for the Container authorization type.
- Save and close the deployment descriptor.

Developing an EJB application

Our first step towards implementing the ITSOBANK model with EJBs is creating the entity beans Customer, Account, Transaction, Debit, and Credit (Figure 16-13).

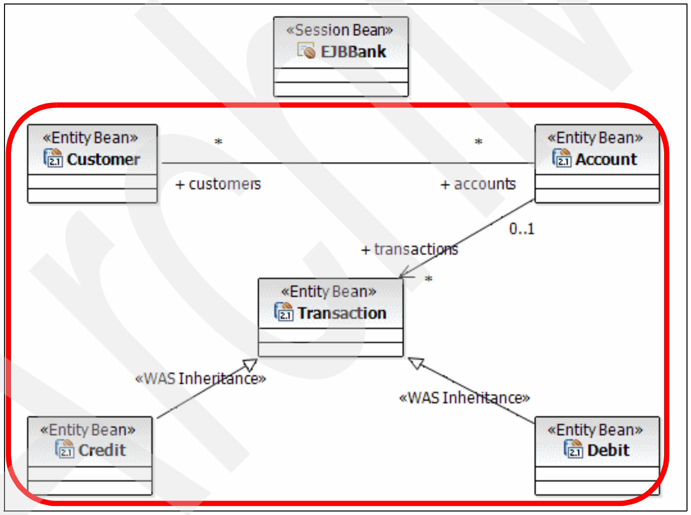


Figure 16-13   Business entities

In this section, we focus on defining and implementing the business logic for the entity beans. In “Object-relational mapping” on page 764, we define the mapping to the relational database.

Creating the entity beans

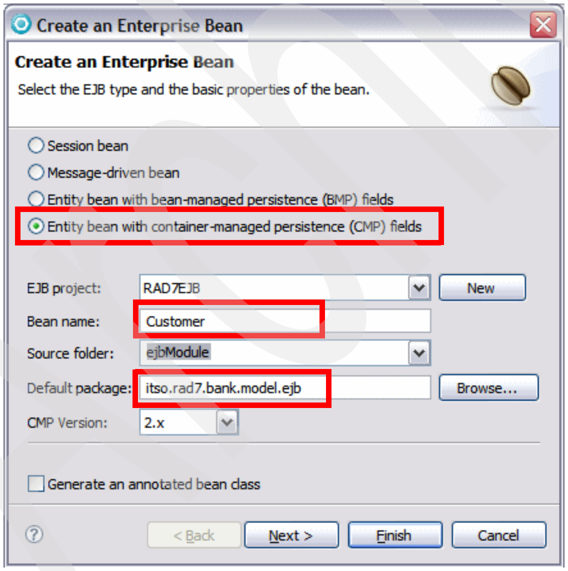
This section describes how to create and implement the `ITSOBANK` entity beans in the following sequence:

- Define the Customer bean.
- Define the Account and Transaction beans.
- Define the Credit and Debit derived beans.

Define the Customer bean

To define the **`Customer`** bean, do these steps:

- Select and expand **RAD7EJB** in the Project Explorer.
- Right-click **Deployment Descriptor: RAD7EJB** and select **New** → **Enterprise Bean**.
- In the Create an Enterprise Bean dialog, do these steps (Figure 16-14):
  - Select **Entity bean with container-managed persistence (CMP) fields**.
  - Bean name: `Customer`
  - Default package: `itso.rad7.bank.model.ejb`
  - Leave the remaining options with the default values and click **Next**.



Create an Enterprise Bean

Create an Enterprise Bean

Select the EJB type and the basic properties of the bean.

○ Session bean

○ Message-driven bean

○ Entity bean with bean-managed persistence (BMP) fields

● Entity bean with container-managed persistence (CMP) fields

EJB project: RAD7EJB

Bean name: Customer

Source folder: ejbModule

Default package: itso.rad7.bank.model.ejb

CMP Version: 2.x

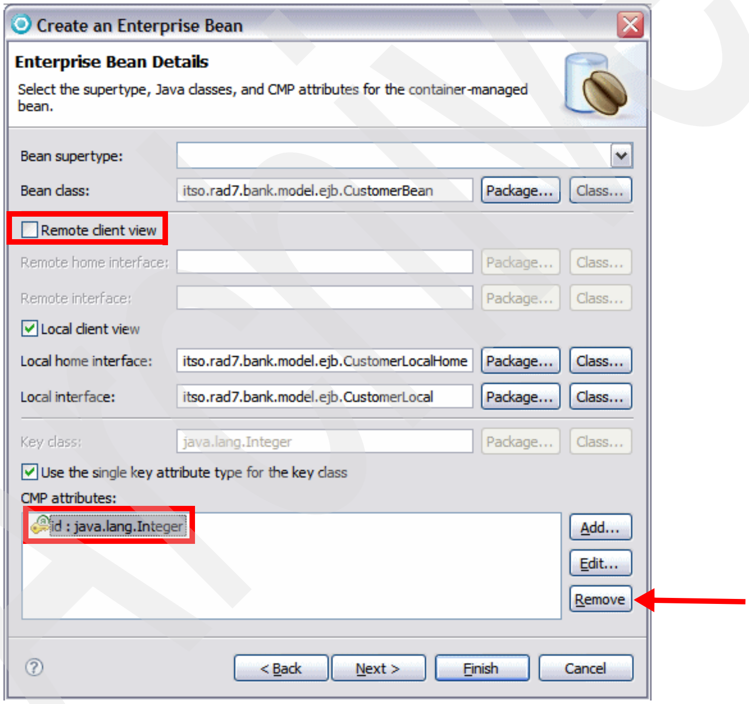
□ Generate an annotated bean class

Figure 16-14 Create an enterprise bean (1)

- In the Enterprise Bean Details, you select the supertype, remote, and/or local client views, and attributes (Figure 16-15):
  - Leave Bean supertype empty. We use supertypes in “Define the Credit and Debit derived beans” on page 749.
  - If selected, clear **Remote client view**.

    > **Note:** Most of the time, the suggested values for the type names (derived from the bean name) are fine, so you do not have to worry about them. According to best practices, entity beans should have only local interfaces, so **Local client view** is selected by default.

  - Select **Local client view** (default).
  - CMP attributes: Select **`id`:java.lang.Integer** and click **Remove**. The `Customer` class has a key field named ssn of type `java.lang.String`.



Create an Enterprise Bean

Enterprise Bean Details

Select the supertype, Java classes, and CMP attributes for the container-managed bean.

Bean supertype:

Bean class: itso.rad7.bank.model.ejb.CustomerBean

☐ Remote client view

Remote home interface:

Remote interface:

☑ Local client view

Local home interface: itso.rad7.bank.model.ejb.CustomerLocalHome

Local interface: itso.rad7.bank.model.ejb.CustomerLocal

Key class: java.lang.Integer

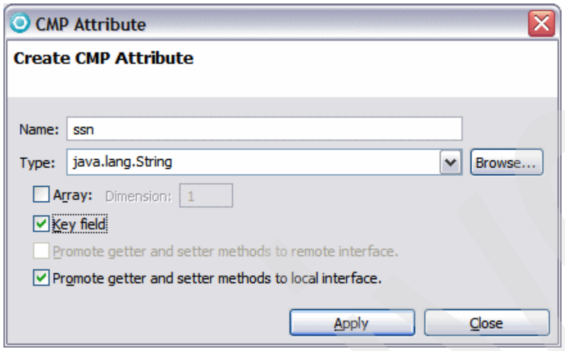
☑ Use the single key attribute type for the key class

CMP attributes:

id : java.lang.Integer

Figure 16-15 Create an entity bean (2)

- Add the CMP attributes by clicking **Add**.
  - The Create CMP Attribute dialog (Figure 16-16) lets you specify the characteristics of the new CMP attribute:
    - Name: **ssn**
    - Type: Select **java.lang.String**
    - Select **Key field**
    - Select **Promote getter and setter methods to local interface**
    - Click **Apply**.



CMP Attribute

Create CMP Attribute

Name: ssn

Type: java.lang.String

□ Array: Dimension: 1

☑ Key field

□ Promote getter and setter methods to remote interface.

☑ Promote getter and setter methods to local interface.

Figure 16-16 Create CMP attributes

Note: You have to create CMP attributes.

If you do not define at least one key CMP attribute, you cannot create the CMP entity bean. Here are some considerations:

- Array: If the attribute is an array, select the **Array** check box and specify the number of the dimensions for it.
- Key field: By selecting **Key field**, you indicate that the new field should be part of the entity's unique identifier. You can declare as many attributes as you want to perform this role. Application Developer is very smart here. If you specify just one key attribute of an object type, it declares that type as the *key class*. If you select an attribute of a non-object type (`int`, `double`), or if you select more than one key attribute, the environment automatically creates a new key class for you, implements all its methods (including `equals` and `hashCode`), and declares it as the key class.
- The two last check boxes let you indicate whether you want to promote the new attribute (through its getter and setter) to either the remote or the local interfaces, or to both. The availability of these options depends on which client views you selected, and if the attribute is a key field.

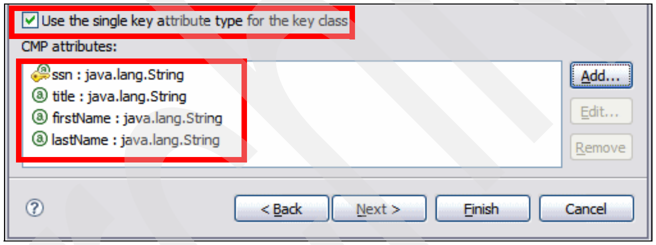
- For the Customer bean, repeat the process of adding a CMP attribute for the fields listed in Table 16-1. Click **Apply** after adding each attribute. Click **Close** when done.

Table 16-1 Customer bean's CMP attributes

Name	Type	Attribute type check box
ssn	java.lang.String	Key field
title	java.lang.String	Promote getter and setter methods to local interface
firstName	java.lang.String	Promote getter and setter methods to local interface
lastName	java.lang.String	Promote getter and setter methods to local interface

> **Tip:** You can enter String, instead of java.lang.String, in the Type field and Application Developer changes the type to java.lang.String.

- After closing the CMP Attribute dialog, the Enterprise Bean Details page (bottom part) is shown in Figure 16-17. Select **Use the single key attribute type for the key class**. Click **Next**.



☑ Use the single key attribute type for the key class

CMP attributes:

- ssn : java.lang.String
- title : java.lang.String
- firstName : java.lang.String
- lastName : java.lang.String

Figure 16-17 Creating an entity bean (2) after adding CMP attributes

- In the EJB Java Class Details page, accept the defaults and click **Next**.

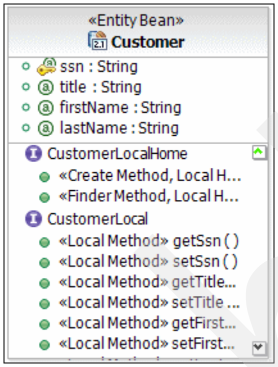
> **Important:** The Bean superclass in the EJB Java Class Detail page is not related to the Bean supertype field in the previous page (Figure 16-15 on page 744). The former is used to define Java class inheritance for the implementation classes that make up the EJB. The latter is used to define the EJB inheritance hierarchy. Refer to "Define a bean supertype" on page 749 for use of the Bean supertype and EJB inheritance feature.

- In the Select Class Diagram for Visualization dialog, click **New**.
- In the New Class Diagram dialog, enter `RAD7EJB/diagrams` in the Enter or select the parent folder field, `ejbs` in the File name field, and click **Finish**.

> **Note:** Although this page allows you to specify a class diagram, we found that it defaulted to use the diagram named `default.dnx`, located in the root of the `RAD7EJB` project. We chose to place all diagrams in a separate folder named diagrams.

- The new UML class diagram is displayed with the `Customer` entity bean (Figure 16-18). Save and close the diagram.

Figure 16-18 Class diagram with Customer entity bean


«Entity Bean»
Customer
ssn : String
title : String
firstName : String
lastName : String
CustomerLocalHome
«Create Method, Local H...
«Finder Method, Local H...
CustomerLocal
«Local Method» getSsn ( )
«Local Method» setSsn ( )
«Local Method» getTitle...
«Local Method» setTitle ...
«Local Method» getFirst...
«Local Method» setFirst...

> **Note:** Enterprise Bean Creation wizard shortens the development cycle of EJBs by generating code templates based on our input in the wizard panels. The wizard generates the bean class and the component and home interfaces. We discuss the generated classes in the later sections.

Define the Account and Transaction beans

Repeat the same process for the next two CMP entity beans, **`Account`** and **`Transaction`**, according to the data in Table 16-2 and Table 16-3.

> **Important:** Make sure to select **Use the single key attribute type for the key class** (Figure 16-17 on page 746).
>
> In the Select Class Diagram for Visualization page, expand and select **RAD7EJB** → **diagrams** → **ejbs.dnx**.

Table 16-2 Account bean CMP attributes

Name	Type	Attribute type check box
`id`	`java.lang.String`	Key field
`balance`	`int`	Promote getter and setter methods to local interface

Table 16-3 Transaction bean CMP attributes

Name	Type	Attribute type check box
`id`	`java.lang.String`	Key field
`amount`	`int`	Promote getter and setter methods to local interface
`timestamp`	`java.util.Date`	Promote getter and setter methods to local interface

Tip: Approaches to resolve situations with no natural unique identifier:

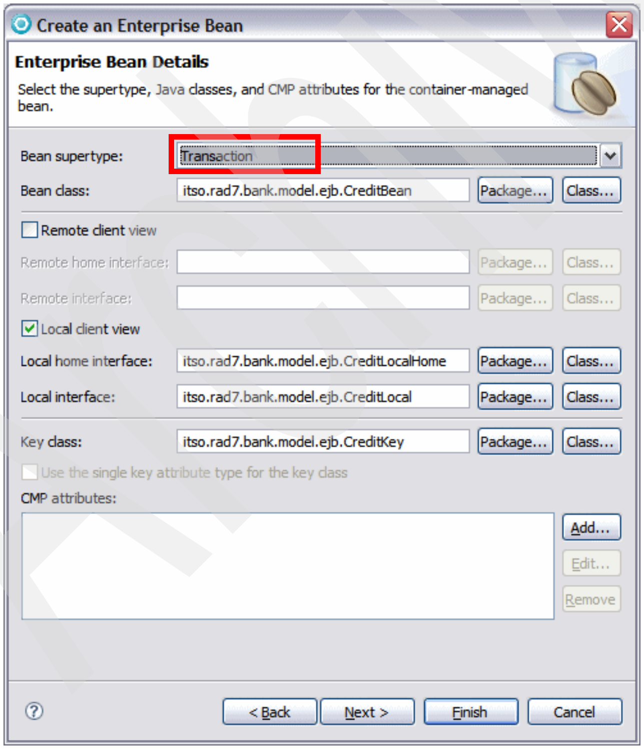
- It is common to find entities that do not have a natural unique identifier, as is the case with our `Account` and `Transaction` objects. The EJB 2.0 specification approached this problem when it introduced the unknown primary key class for CMP entity beans. Even though WebSphere Application Server has implemented this part of the specification since Version 5, Application Developer does not include support for this.
- There are basically two approaches to the problem. The first is to have the back-end database generate the unique identifiers. This is feasible because even though you can have as many application servers as you like, the data pertinent to a single entity hopefully will be stored in just one database. The downside to this approach is that every time an entity is created, a database table must be locked in order to generate the ID, and thus becomes a bottleneck in the process. The upside is that sequential identifiers can be generated this way.
- The second approach is to generate the universally unique identifiers (UUIDs, unique even among distributed systems) in the application server tier. This is a little tricky, but can be accomplished. One way to do it is to come up with a utility class that generates the UUIDs based on a unique JVM identifier, the machine's IP address, the system time, and an internal counter. This technique is usually more efficient than having the back-end database generate UUIDs because it does not involve table locking. This was our selected approach for the **`Transaction`** bean. We used the `com.ibm.ejs.util.Uuid` class—which comes with Application Developer and WebSphere Application Server—to generate UUIDs.
- For the **`Account`** class, we generate account numbers using the first part of the customer ssn and the system timestamp.

Define the Credit and Debit derived beans

Complete the creation of our business entities by defining the last two beans: **`Credit`** and **`Debit`**. Both are subtypes of **`Transaction`**, so the process of creating them is slightly different:

- In the Create an Enterprise Bean dialog, do these steps to define the **`Credit`** bean (subtype of **`Transaction`**):
  - Select **Entity bean with container-managed persistence (CMP) fields**.
  - Bean name: `Credit`
  - Default package: `itso.rad7.bank.model.ejb`
  - Leave the remaining options with the default values and click **Next**.
- In the Enterprise Bean Details dialog, select **`Transaction`** in the Bean supertype drop-down (Figure 16-19) and click **Next**.

  This page lets you select the supertype (allowing you to define the inheritance structures), type names, which views you would like to create, and finally the key class and CMP attributes. (Note that we do not define additional attributes.)


Create an Enterprise Bean

Enterprise Bean Details

Select the supertype, Java classes, and CMP attributes for the container-managed bean.

Bean supertype: Transaction

Bean class: itso.rad7.bank.model.ejb.CreditBean

□ Remote client view

Remote home interface:

Remote interface:

☑ Local client view

Local home interface: itso.rad7.bank.model.ejb.CreditLocalHome

Local interface: itso.rad7.bank.model.ejb.CreditLocal

Key class: itso.rad7.bank.model.ejb.CreditKey

□ Use the single key attribute type for the key class

CMP attributes:

Figure 16-19 Define a bean supertype

- In the EJB Java Class Details page, click **Next**.
- In the Select Class Diagram for Visualization page, expand and select **RAD7EJB** → **diagrams** → **ejbs.dnx** and click **Finish**.
- Repeat the process for the `Debit` bean. None of the beans have additional attributes, apart from the attributes inherited from the `Transaction` EJB. They only differ in behavior.
- At this point, we have all the five entities created. The class diagram is shown in Figure 16-20. As you can see, for each entity bean, we have a primary key attribute, regular attributes, and home and component interfaces.
- Save and close the class diagram.

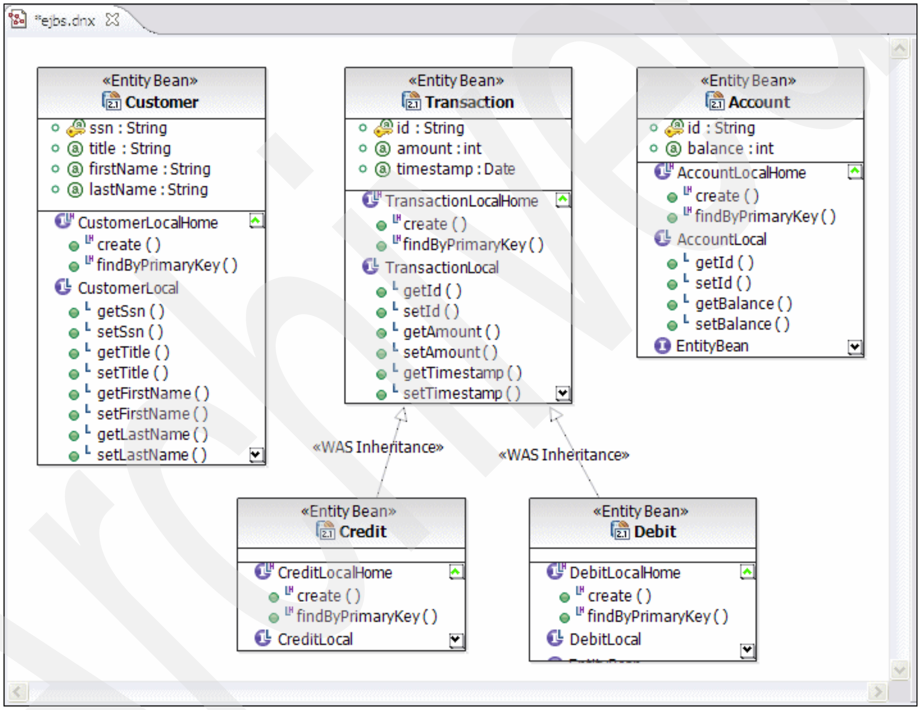


Figure 16-20 Class diagram at entity bean creation phase

Creating the entity relationships

Now that the five business entities have been created, it is time to specify their relationships: A one-to-many unidirectional association, implementing an analysis composition, and a many-to-many bidirectional association (Figure 16-21).

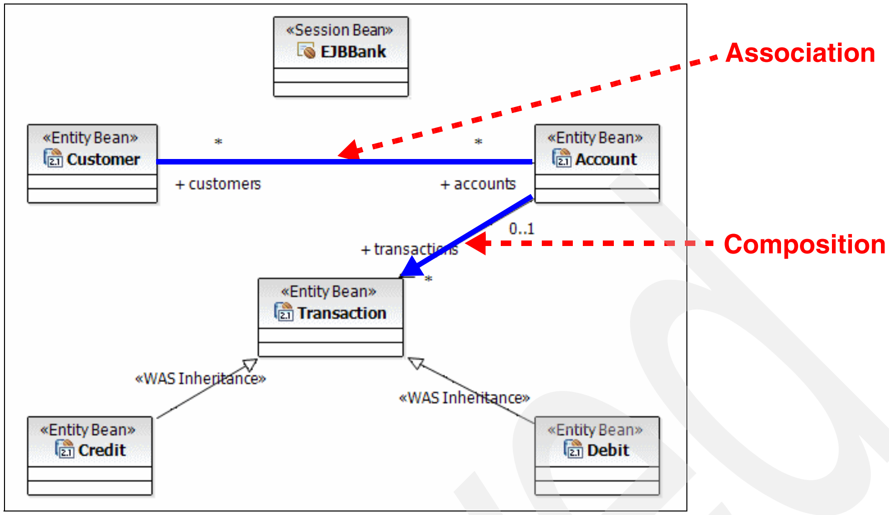


Figure 16-21 Relationships in the model

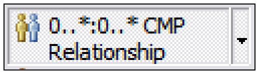
Application Developer offers a couple of facilities to streamline the process of both creating and maintaining container-managed relationships (CMRs). You can, for instance, visually create relationships with the UML Diagram editor, and the environment will automatically generate all the necessary code and deployment descriptor changes. You can, alternatively, edit the deployment descriptor directly, and Application Developer will also generate the appropriate code changes. Finally, you can use the workbench menus to accomplish the same task.

In the following sections we use the first two strategies described above to create the association and the composition relationships, respectively.

Customer Account association relationship

The first relationship that we add is the association between the `Customer` bean and the `Account` bean. In this example, we demonstrate how to define relationships using the UML Diagram editor. Follow these steps:

- Open the Java perspective.
- From the Project Explorer view, expand **RAD7EJB** → **diagrams**.
- Double-click **ejbs.dnx** to open it with the UML diagram editor.
- In the Palette click the down-arrow to the right of `0..1:0..1 CMP Relationship` and select **0..*:0..* CMP Relationship**.



- Left-click and hold the mouse arrow over the **Customer** bean, and then drag the mouse towards the **Account** bean. Release the button over the **Account** bean to create the association relationship (Figure 16-22).

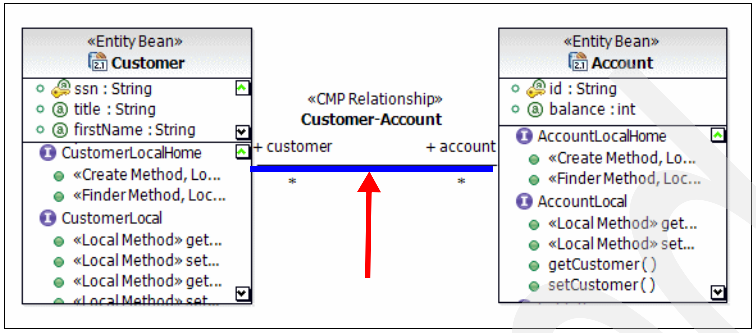


Figure 16-22 Association relationship between Customer and Account (part 1)

> **Note:** The names at the ends of the association (in this case `customer` and `account`) are used to generate accessor methods in the two beans' interface.
>
> The plus sign (+) means that the relationship is visible to the associated entity at the respective end.

- Double-click **+ customer** and change to **+ customers**.
- Double-click **+ account** and change to **+ accounts.**

The resulting relationship and the added methods to support it are shown in Figure 16-23.

Note that we changed the names to indicate that multiple account instances can be related to one customer (and vice versa).

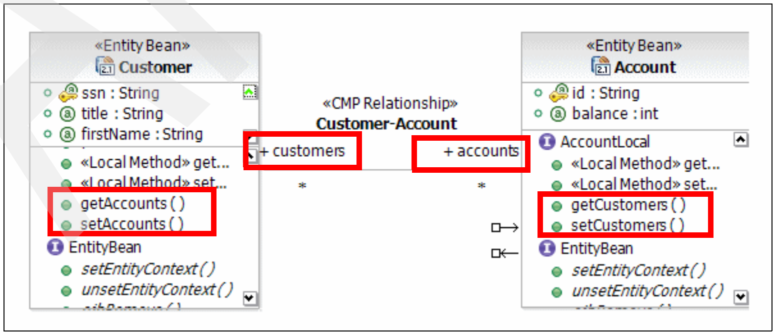


Figure 16-23 Finished customer account relationship

- Save the changes and close the editor.

> **Important:** If you ever have to delete the relationship, right-click the relationship, and select **Delete from Deployment Descriptor**.
>
> If you select **Delete from Diagram**, or simply press Delete, the relationship is only removed from the diagram, but stays in the model.
>
> If a relationship is deleted from the diagram but stays in the model, it can be redrawn by right-clicking one of the related entities and selecting **Filters** → **Show / Hide Relationships**. The resulting Show/Hide Relationships dialog can then be used to show or hide specific relationship types.

Account Transaction composition relationship

The second relationship that has to be created is the composition between the **`Account`** and the **`Transaction`** beans. It represents the account's transaction log that has to be maintained over time. In this example, we create the relationship using the Deployment Descriptor editor to demonstrate an alternative to the UML Diagram editor. Follow these steps:

- Open the EJB Deployment Descriptor.
- On the **Bean** page, select the **Account** bean.
- Scroll down to the **Relationships** section.
- Click **Add** to create a new relationship for the `Account` bean (Figure 16-24).

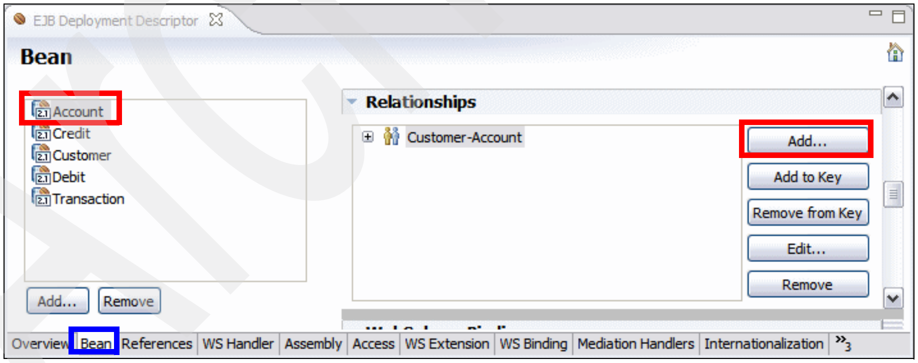
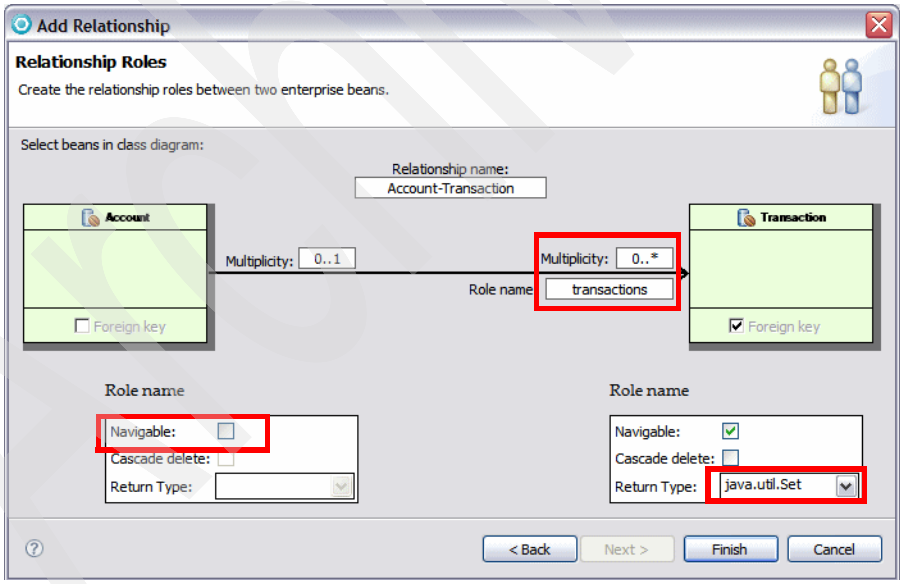


Figure 16-24 Defining relationships with the EJB Deployment Descriptor editor

- The Relationship wizard opens, showing an UML view of the relationship.

  Select the **`Account`** bean on the left-hand side and the **`Transaction`** bean on the right-hand side. Click **Next**.

- In the Relationship Roles dialog, the wizard automatically creates the Role name and multiplicity fields:
  - `account`, 0..1
  - `transaction`, 0..1
- Modify the relationship:
  - One account can be associated with many transactions, therefore change the role to **transactions** and the multiplicity to **0..*** (click on the fields to change them).
  - We also want to guarantee that the same transaction is not added to the account twice. For the Return Type, select **`java.util.Set`**.
  - The Account-Transaction relationship is a composition relationship, so the composed objects should have no knowledge about the composer. Clear **Navigable** under the Account. This removed the role name under the relationship line and the arrow.
  - Note that Foreign Key is selected in the Transaction box.
  - The completed relationship definition is shown in Figure 16-25.
  - Click **Finish**.



Add Relationship

Relationship Roles

Create the relationship roles between two enterprise beans.

Select beans in class diagram:

Relationship name: Account-Transaction

Account: Multiplicity: 0..1; ☐ Foreign key

Transaction: Multiplicity: 0..*; Role name: transactions; ☑ Foreign key

Role name (Account): Navigable: ☐; Cascade delete: ☐; Return Type:

Role name (Transaction): Navigable: ☑; Cascade delete: ☐; Return Type: java.util.Set

Figure 16-25 Add Relationship wizard

- Select the Customer-Account relationship and click **Edit** to see its diagram.
- Save the deployment descriptor and close it.

As we have seen in the EJB diagram, defining a relationship adds methods to the class. For example, for the Account we have now these methods:

```
public java.util.Collection getCustomers();
public void setCustomers(java.util.Collection aCustomers);
public java.util.Set getTransactions();
public void setTransactions(java.util.Set aTransactions);
```

To see the Account-Transaction relationship in the UML diagram, select **Filters** → **Show / Hide Relationships**. Select all relationships and click **OK**.

Customizing the entity beans

Now that the five entity beans with two relationships have been created, let us do a little programming. For each of the beans created, three types were generated: The bean class, the home interface, and the local component interface.

- View the Account bean, for instance, to see the generated code. Expand **Deployment Descriptor: RAD7EJB** → **Entity Beans** → **Account** (Figure 16-26).

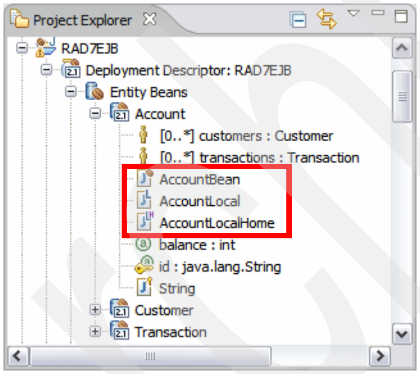
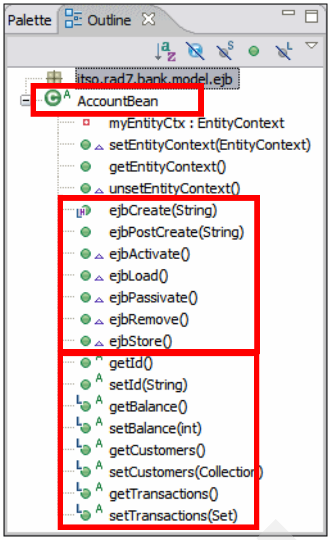


Figure 16-26 Generated types for the Account bean

> **Note:** There is a fourth type associated with the Account bean: String. It is the primary key class. If we had chosen a non-object key field or multiple key fields, a key class would also have been generated.

- Open the **AccountBean** class and select the **Outline** view (Figure 16-27).



AccountBean class

Life cycle methods:
Callback methods used by the EJB container at predefined events

Business methods:
Manipulate the CMP attributes

Figure 16-27 Outline view of the AccountBean class

Manage the home and component interfaces

As you can see, Application Developer has already generated the life cycle methods and some business methods, the latter being the getters and setters. Some of these methods belong to the bean's home interface. Others belong to the local interface. Some of them are private to the component and should not be exposed.

Whether a method is exposed to the interface or not is a design choice. We will make some modifications to the generated interfaces in order to accommodate our design decisions.

Limit access to the Transaction bean

We do not want clients to instantiate the `Transaction` EJB or modify fields in an already created instance. The reason for the former is that in our model, this represents an abstract entity. While there is no such thing as an abstract EJB, in the same sense as an abstract Java class, we can obtain a similar behavior by removing any `ejbCreate` methods from the remote and home interfaces. The reason why we do not want clients to change a transaction object is that these should appear immutable, as they represent a log of historical transactions.

To block access for clients to the setter methods, as well as the constructor and setters for the `Transaction` EJB, do these steps:

- Open the **AccountBean**:
  - In the Outline view for the `AccountBean` class, right-click **setBalance** and select **Enterprise Bean → Demote from Local Interface**.

    Notice that the L-shaped icon next to `setBalance` disappears. The `setBalance` method is now inaccessible to clients. You might have to close and open the bean to see this change reflected in the Outline view.
- Open the **TransactionBean**:
  - In the Outline view, right-click **ejbCreate(String)** and select **Enterprise Bean → Demote from Local Home Interface**.
  - Right-click **setAmount(int)** and select **Enterprise Bean → Demote from Local Interface**.
  - Right-click **setTimestamp(Date)** and select **Enterprise Bean → Demote from Local Interface**.

Modify ejbCreate methods for Transaction, Credit, and Debit

When creating a transaction object, we want to be able to specify the transaction amount and let the bean generate the identifier and timestamp automatically. Do the following steps to accomplish this:

> **Tip:** The Java code for this section can be copied from the file `c:\7501code\ejb\source\Transactions.jpage`.

- Open the **TransactionBean**:
  - Modify the `ejbCreate` and `ejbPostCreate` methods so they look like Example 16-1.

> **Note:** As noted earlier, the class `com.ibm.ejs.util.Uuid`, referenced in Example 16-1, is used to generate unique identifiers. We use it here to automatically generate identifiers for any transaction object. Thus clients do not have to worry about this.

Example 16-1 TransactionBean ejbCreate and ejbPostCreate

```
public java.lang.String ejbCreate(int amount)
   throws javax.ejb.CreateException {
   setId((new com.ibm.ejs.util.Uuid()).toString());
   setAmount(amount);
   setTimestamp(new java.util.Date());
   return null;
}

public void ejbPostCreate(int amount)
   throws javax.ejb.CreateException {
}
```

  - Select the `ejbCreate` method in the Outline view and **Enterprise Bean → Promote to Local Home Interface**. This action updates the `TransactionLocalHome` class.

- Open the **`CreditLocalHome`** class. You should see a warning in the problems view:

```
CHKJ2504W: The ejbCreate matching method must exist on itso.rad7.bank.mo
del.ejb.CreditBean (EJB 2.0: 10.6.12)
```

  - Modify the `create` method signature to match the signature:

```
public itso.rad7.bank.model.ejb.CreditLocal create(int amount)
   throws javax.ejb.CreateException;
```

- Open the **`DebitLocalHome`** class and change the `create` method signature in the same way.

Adding the business logic

Getters and setters are generated automatically, but the business methods must be implemented manually.

We implement these business logic methods:

- Transaction: **`getTransactionType`**—Returns the string `Credit` or `Debit`, depending on the subclass.
- Account: **`processTransaction`**—Update the balance based on the transaction.

Tip: The Java code for this section can be copied from the file `c:\7501code\ejb\source\Transactions.jpage`.

Add business logic to the Transaction bean

To add the `getTransactionType` method to the `Transaction` EJB, do these steps:

- Open the **TransactionBean.java** and add the `getTransactionType` method to the class after the last method:

```
public String getTransactionType()
      throws itso.rad7.bank.exception.ITSOBankException {
   throw new itso.rad7.bank.exception.ITSOBankException(
         "Transaction.getTransactionType invoked!");
}
```

The implementation of the method is not relevant in the base `Transaction` bean, because the method is conceptually abstract. In a Java class hierarchy, this method would be made abstract, but this is not possible in an EJB hierarchy. If we were to define the method as abstract, the deployed code would have errors, and the EJB would not be deployable.

We thus choose to throw an exception from the method in case a programming error results in the execution of the method.

- Promote the `getTransactionType` method to the local interface (select the method in the Outline view and **Enterprise Bean** → **Promote to Local Interface**.

Modify the CreditBean class

Open the **CreditBean** class and insert the `getTransactionType` method:

- Select **Source** → **Override/Implement Methods**. Click **Deselect All** and only select the `getTransactionType` method. Click **OK** to generate the method into the source code.
- Add this code to the class:

```
/** Insert before the first method **/
public static final String TYPE_KEY = "Credit";

/** Update the getTransactionType method **/
public String getTransactionType() throws ITSOBankException {
   return TYPE_KEY;
}
```

  – Save the changes and close the editor.

Modify the DebitBean class

Open the **DebitBean** class and make the same changes, but using this constant:

```
public static final String TYPE_KEY = "Debit";
```

> **Note:** The `TYPE_KEY` constant defined in the `Debit` and `Credit` classes is used by the EJB container to determine what type a given record in the database refers to, because both `Credit` and `Debit` instances are persisted to the same database table. The value of this constant is compared to the value of the *discriminator column* (`TRANSTYPE`) in the `TRANSACTIONS` table.

Add business logic to the Account bean

When we specified the account bean CMP fields, we configured the wizard to expose both the setter and the getter for the balance attribute to the local interface. This is why there is a small L-shaped icon next to the `getBalance` and `setBalance` methods. While it is fine for a client to retrieve the account balance, we do not want the clients to change the balance by calling the `setBalance` method. Later we implement and expose another business method, `processTransaction`, that manipulates the balance, adhering to business rules.

From a business perspective, it makes no sense to allow the creation of accounts that are not associated to any customers. Thus, the first modification that we want to make is to guarantee that accounts are not created unless a primary customer is specified.

> **Tip:** The Java code for this section can be copied from the file `c:\7501code\ejb\source\AccountBean.jpage`.

- Open the **AccountBean**.
- Refactor the `ejbCreate(String)` method signature.

> **Tip:** For refactoring, if you need to edit the signature of any method that already belongs to either the remote or home interface, the easiest way is as follows:
>
> - First demote the method from the interface.
> - Edit the signature.
> - Promote the method back to the interface.
>
> If you do not demote the method from the interfaces first, you will have to manually edit the method signatures in these interfaces. Using the approach mentioned here, you let Application Developer update the method signatures in the interfaces when the method is promoted back to the interfaces.
>
> Since the *throws* clause for a method is part of the method signature, this procedure should also be followed when changing this clause.

- In the Outline view, select the existing **ejbCreate(String)** method and **Enterprise Bean** → **Demote from Local Home Interface** to remove the create method declaration from the home interface.

  Modify the `ejbCreate(String)` and `ejbPostCreate(String)` methods to create an account for a customer:

```
public java.lang.String ejbCreate(CustomerLocal primaryCustomer)
    throws javax.ejb.CreateException {
    String id = (primaryCustomer.getSsn().substring(0,3) + "-"
            + System.currentTimeMillis()).substring(0,16);
  setId(id);
  System.out.println("Creating account " + id + " for customer " +
                primaryCustomer);
```

```
      return null;
   }

   public void ejbPostCreate(CustomerLocal primaryCustomer)
         throws javax.ejb.CreateException {
      getCustomers().add(primaryCustomer);
   }
```

> **Important:** The getCustomers method was generated when we created the association relationship between the Account and the Customer beans. According to the J2EE specification this method cannot be called from the ejbCreate method. In the ejbPostCreate method, on the other hand, the instance can reference the associated entity objects, and a call to getCustomers can be made.

Now add the create method back to the home interface by selecting **ejbCreate(CustomerLocal)** in the Outline view and **Enterprise Bean** → **Promote to Local Home Interface**.

- When we created the relationship between the Account and the Customer beans, the method setCustomers(Collection) was created. It does not have to be available to clients. Right-click **setCustomers(Collection)** in the Outline view and demote it from the local interface by selecting **Enterprise Bean** → **Demote from Local Interface**.
- Demote the accessors for the Account-Transaction relationships:

  As you might recall, the relationship between the Account bean and the Transaction bean is a composition. This means that references to transaction objects cannot be exported to objects outside of the Account bean.

  When we created the relationship, getTransactions and setTransactions methods were generated and added to the local interface for the Account bean. We want clients to be able to access the transaction log. If we were to allow access to these methods, clients would be able not only to do so, but also to add transactions to the log without correctly processing them using the processTransaction method, which we define later in this section. The result would be that the affected account object would be put into an invalid state. This would be breaching the object’s encapsulation, and thus should not be allowed.

  Demote both getTransactions and setTransactions(Set) methods, then add the getLog method as follows:

  – Select both **getTransactions** and **setTransactions(Set)** in the Outline view and **Enterprise Bean** → **Demote from Local Interface**.

  - Add the getLog method to the AccountBean:

    ```
    public java.util.Set getLog() {
       return java.util.Collections.unmodifiableSet(getTransactions());
    }
    ```

  - Right-click **getLog** in the Outline view and select **Enterprise Bean** → **Promote to Local Interface**.

- Add the processTransaction method to the AccountBean:

  ```
  public void processTransaction(TransactionLocal transaction)
        throws itso.rad7.bank.exception.ITSOBankException {
     if (transaction instanceof CreditLocal)
        setBalance(getBalance() + transaction.getAmount());
     else
        setBalance(getBalance() - transaction.getAmount());
     getTransactions().add(transaction);
  }
  ```

  The processTransaction method receives a transaction object local reference and changes the account balance by adding or subtracting the transaction amount based on the type of the transaction object (Credit or Debit).

  Add the method to the bean's local interface (as described above).

Creating custom finders

When you create an entity bean, you always get the findByPrimaryKey finder method on the home interface. Sometimes, though, you have to find an entity based on criteria other than just the primary key. For these occasions, the EJB 2.1 specification provides a query language called EJB QL. Custom finder methods are declared in the home interface and defined in the EJB deployment descriptor using the EJB QL.

Our example requires two similar simple custom finders: One for the Account bean and the other for the Customer bean.

Account findAll query

To add the findAll query to the Account bean, do these steps:

- From the Project Explorer, expand **RAD7EJB** → **Diagrams** → **ejbs.dnx** and open the class diagram in the UML Diagram editor.
- Right-click the **Account** bean and select **Add EJB** → **Query**.
- In the Add Finder Descriptor wizard, do these steps (Figure 16-28):
  - Method: Select **New**.

The only finder we have in our `AccountLocalHome` interface is the default `findByPrimaryKey`. Application Developer will take care of updating the home interface for you by adding the declaration of the new finder method.

- Method Type: Select **find method**.

  The Method Type field lets you select whether you want to create a descriptor for a finder method or for an `ejbSelect` method. The difference between the two is that finder methods get promoted to the home interface, whereas `ejbSelect` methods are useful as internal helper methods and cannot be called by clients.

- Type: Select **Local**.

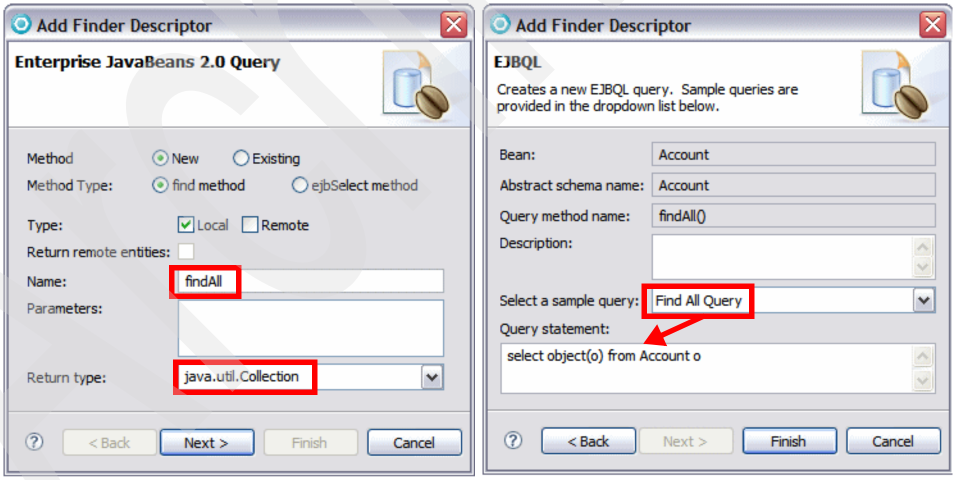
  The Type field lets you select to which home interface, either local or remote, you would like the finder method promoted. Note that we do not have a remote interface in the entity beans.

- Name: `findAll`

- Return type: Select **java.util.Collection** (the method returns mutliple accounts)

- Click **Next**.

► In the Add Finder Descriptor dialog, select **Find All Query** to generate the EJB QL code**,** and click **Finish**.



Add Finder Descriptor

Enterprise JavaBeans 2.0 Query

Method ● New ○ Existing

Method Type: ● find method ○ ejbSelect method

Type: ☑ Local □ Remote

Return remote entities: □

Name: findAll

Parameters:

Return type: java.util.Collection

Add Finder Descriptor

EJBQL

Creates a new EJBQL query. Sample queries are provided in the dropdown list below.

Bean: Account

Abstract schema name: Account

Query method name: findAll()

Description:

Select a sample query: Find All Query

Query statement:

select object(o) from Account o

Figure 16-28 Adding a finder method

► Open the EJB deployment descriptor. On the Bean page, select **Account**, scroll-down to **Queries**, and you can find the `findAll` query. You can define and update queries also in the deployment descriptor.

Customer findByName query

Repeat the same steps for the Customer bean to create a **findByName** query, which returns customer by providing a partial lastname:

- Create a new local findByName method with one parameter (click **Add**) named **partialName** of type String, and a return type of java.util.Collection.
- Select **Single Where Predicate** for the sample query, then modify the code:

```
select object(o) from Customer o where o.lastName like ?1
```

- Close the editor.

Object-relational mapping

Container-managed persistence (CMP) entity beans delegate their persistence to the container. As mentioned in “Relationships” on page 727, this means that it is the responsibility of the EJB container to handle the details of how to store the internal state of the EJBs.

For the container to be able to do this, we have to provide information about how the EJB fields are mapped to the relational database. This information is stored in the deployment descriptor, and during deployment, the JDBC code to perform the operations is generated by the container.

When the beans are actually deployed, associations to real data sources can be made to dynamically bind the bean to the data. In this way, the CMPs are abstract classes that associate to data, but do not provide any implementation for accessing data themselves.

To facilitate development, deployment, and testing, Application Developer contains the tools for the developer to both define the mappings and create deployment bindings.

The advantages to separating the development and persistence concerns are numerous. Apart from achieving database implementation independence, the developer is free to work with object views of the domain data instead of data views and writing SQL, and is allowed to focus on the business logic, instead of the technical details of accessing the database.

As mentioned, the CMP can be developed largely independently of the data source, and allows a clear separation of business and data access logic. This is one of the fundamental axioms of aspect-oriented programming, where the aspect of persistence can be removed from the development process and applied later, in this case at deployment time.

Application Developer offers three different mapping strategies:

- *Top-down* is done when you start from an object-oriented model and let the environment generate the data model automatically for you, including the object-relational mapping and the DDL that you would use to create the tables in the database This approach is also called *forward engineering*.

  > **Note:** This strategy is preferred when the data back-end does not exist and is created from scratch.

- *Bottom-up* is done when you start from a data model and let Application Developer generate the object model automatically for you, including the creation of the entity beans based on tables and columns that you select. It is the opposite of top-down approach. This approach is also called *reverse engineering*.

  > **Note:** This strategy is not recommended for object-oriented applications, because the data model is less expressive than the object model. It should be used only for prototyping purposes.

- *Meet-in-the-middle* is the compromise strategy, in which you keep both your existing object-oriented and data models, creating a mapping between the two. Meet-in-the-middle is the most common approach used in the e-business projects.

For the sample application, we use the meet-in-the-middle strategy because we do have an existing database for application data.

ITSOBANK database schema

This chapter uses the same relational model created in Chapter 7, “Develop Java applications” on page 227. Figure 16-29 shows the existing data model.

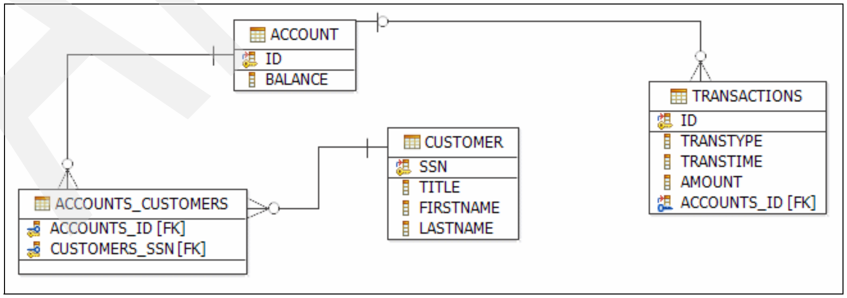


Figure 16-29 Existing relational data model

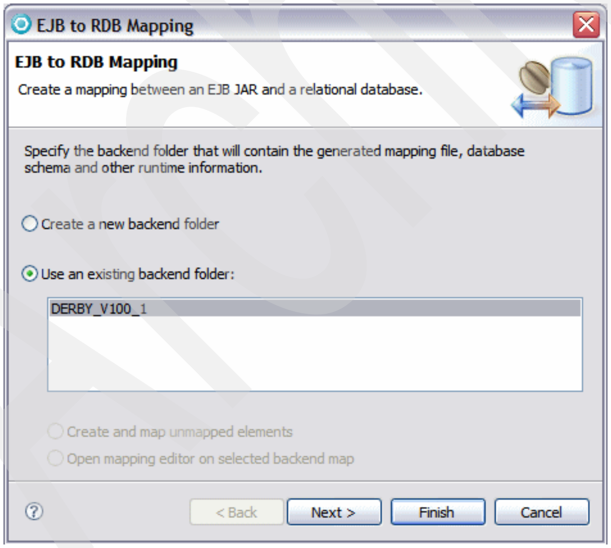
Our objective is to map the object model created in “Developing an EJB application” on page 742, to the existing relational model. As mentioned, this is a more realistic approach than creating a new top-down mapping from scratch, although a top-down mapping would be simpler and more convenient if a relational database model did not already exist.

Generating the EJB to RDB mapping

We now create a mapping between the object-oriented EJB model and the relational database model, as defined by the `ITSOBANK` database schema. To generate the EJB-to-RDB mapping, do these steps:

- Right-click **RAD7EJB** and select **EJB to RDB Mapping** → **Generate Map** from the context menu.
- In the EJB to RDB Mapping wizard, select **Use an existing backend folder**. Select **DERBY_V101_1** and click **Next** (Figure 16-30).

> **Note:** This is the folder that was created when we imported the database metadata to the EJB project in “Import the physical data model” on page 738.



EJB to RDB Mapping

EJB to RDB Mapping

Create a mapping between an EJB JAR and a relational database.

Specify the backend folder that will contain the generated mapping file, database schema and other runtime information.

○ Create a new backend folder

● Use an existing backend folder:

DERBY_V100_1

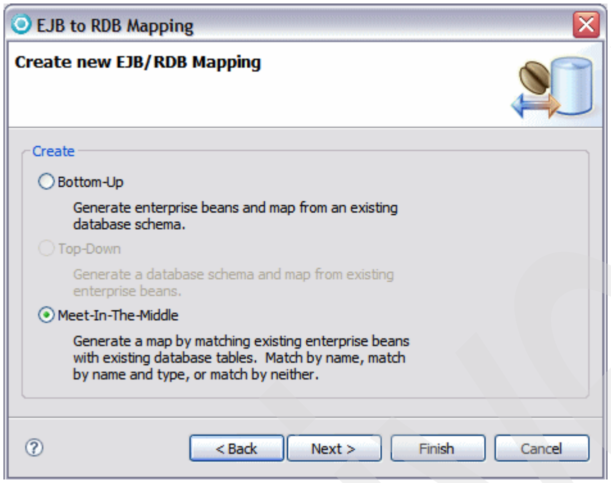
○ Create and map unmapped elements

○ Open mapping editor on selected backend map

Figure 16-30 Using an existing backend folder

- In the Create new EJB / RDB Mapping dialog, select **Meet In the Middle** and click **Next** (Figure 16-31).

The Create new EJB / RDB Mapping dialog lets you select which kind of mapping you would like to create. The options displayed are the ones discussed earlier: Bottom-Up, Top-Down (greyed out because we are using an existing backend folder) and Meet-In-The-Middle.



Create new EJB/RDB Mapping

Create

○ Bottom-Up

Generate enterprise beans and map from an existing database schema.

○ Top-Down

Generate a database schema and map from existing enterprise beans.

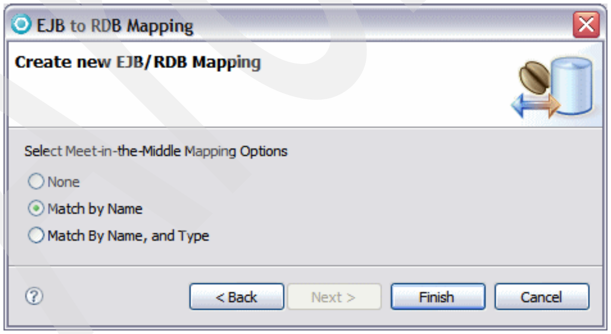
● Meet-In-The-Middle

Generate a map by matching existing enterprise beans with existing database tables. Match by name, match by name and type, or match by neither.

Figure 16-31 Creating a meet-in-the-middle EJB-to-RDB mapping

- In the Select Meet-in-Middle Mapping options page, select **Match by Name** (Figure 16-32) and click **Finish**.

  When selecting Match by Name, Application Developer attempts to match the entity beans to table names and entity bean field names to column names. When the EJBs and their fields are appropriately named, the amount of manual work can be minimized.



Create new EJB/RDB Mapping

Select Meet-in-the-Middle Mapping Options

○ None

● Match by Name

○ Match By Name, and Type

Figure 16-32 Select Match by Name option

Completing the EJB-to-RDB mapping

After completing the EJB to RDB wizard from the previous section, the Mapping editor opens on a file named `Map.mapxmi` (Figure 16-33).

The wizard has already mapped the `Customer` and `Account` beans to the correct tables, and some of the fields are mapped to the correct columns. The mapped items carry a little triangle as an indicator, and they are listed in the bottom pane.

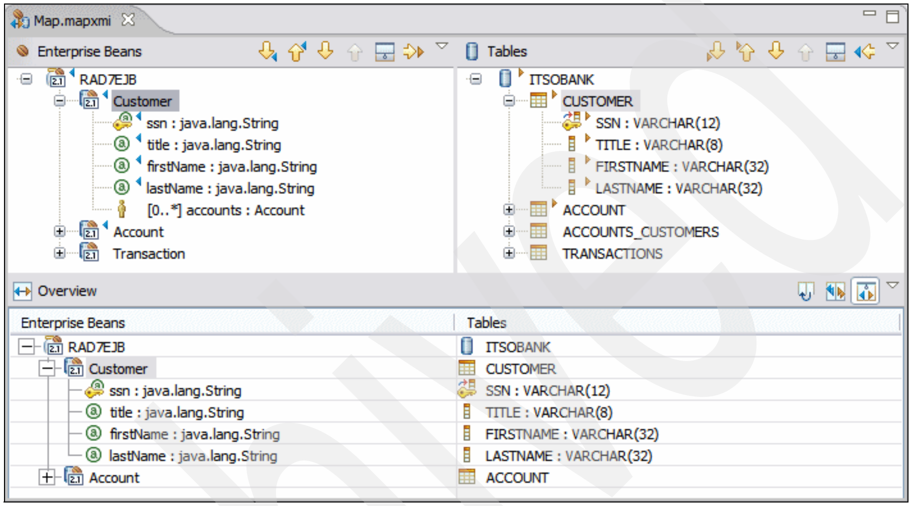


Figure 16-33 Generated object-relational mapping

There are two possible methods of completing the mapping: Drag-and-drop, and using the context menus and selecting **Create Mapping**.

To complete the EJB-to-RDB mapping using the drag-and-drop approach, we map EJBs to tables, attributes to columns, and relationships to foreign keys.

Map the EJBs to tables

A bean must be mapped to a table before you can map the attributes of the bean to the columns.

- Drag the **Transaction** EJB and drop it on the `TRANSACTIONS` table.
- Expand the **Transaction** EJB.
- Drag the **Credit** EJB and drop it on the `TRANSACTIONS` table.
- Drag the **Debit** EJB and drop it on the `TRANSACTIONS` table.

We have to define the *discriminator column* to tell whether a transaction is a debit or a credit. A *discriminator column* is used to specify the type of the instance in the hierarchy. The contents of this column correspond to the value of the TYPE_KEY constant field, which was added to the Credit and Debit beans in "Modify ejbCreate methods for Transaction, Credit, and Debit" on page 757. To define a discriminator column do these steps:

- In the Outline view, select **Transaction <--> TRANSACTIONS** (Figure 16-34).

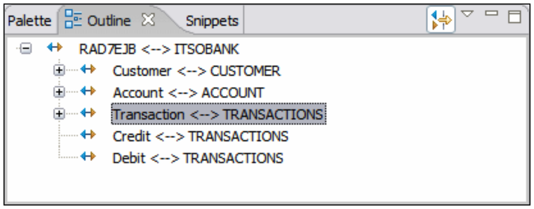
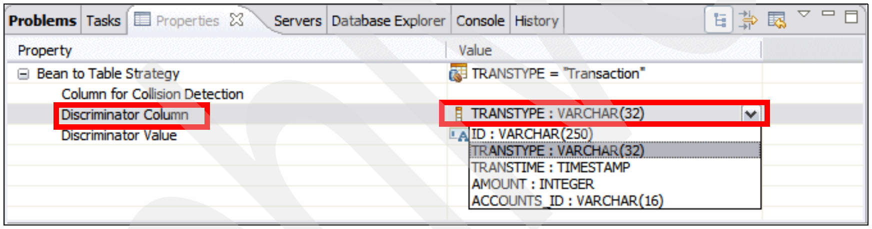


Figure 16-34 Outline view with relationships

- In the Properties view, expand **Bean to Table Strategy**.
- Select Discriminator Column and **TRANSTYPE : VARCHAR(32)** from the drop-down list (Figure 16-35).



Property	Value
Bean to Table Strategy	TRANSTYPE = "Transaction"
Column for Collision Detection	
Discriminator Column	TRANSTYPE : VARCHAR(32)
Discriminator Value	

Figure 16-35 Discriminator Column Selection

- Select the **Credit** bean (in the Overview or Outline). In the Properties view verify the value of the TRANSTYPE attribute as Credit. If the actual value in the database table was different, we would specify it here.
- Fro the **Debit** bean the discriminator value is Debit.

Mapping EJB attributes to columns

Some fields have not been matched automatically. We can perform the manual mapping by dragging and dropping attributes in the left pane to relational columns on the right, or vice-versa.

- Expand the **Transaction** EJB and the **TRANSACTIONS** table.
- Drag the **id** attribute of the Transaction EJB to the ID column.
- Drag the **amount** attribute to the AMOUNT column.
- Drag the **timestamp** attribute to the TRANSTIME column.
- We cannot map the TRANSTYPE and ACCOUNTS_ID columns to anything.

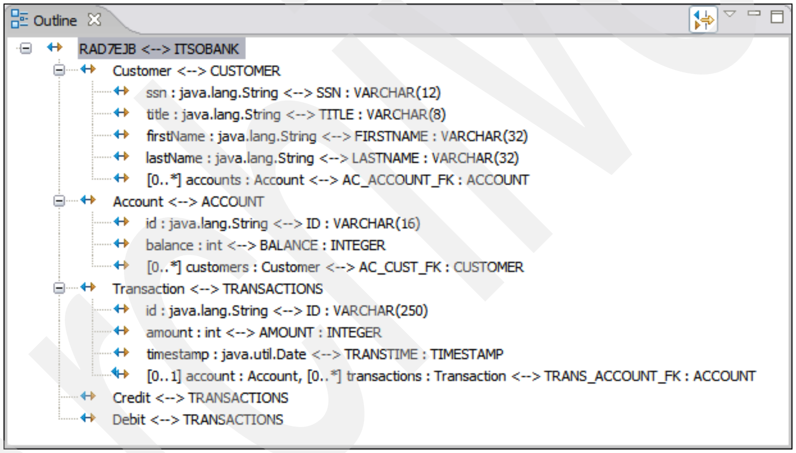
Map container-managed relationships to foreign keys

Relationships are implemented in the database tables through an extra table (`ACCOUNTS_CUSTOMERS`) and foreign keys (for example, `TRANS_ACCOUNT_FK` in the `TRANSACTIONS` table).

To map the relationships to the foreign keys, do these steps:

- In the `Map.mapxmi` editor, drag **[0..1] account: Account** from the `Transaction` bean and drop it on **TRANS_ACCOUNT_FK**. Notice that this also maps `[0..*] transactions: Transaction` (under `Account`) to the same foreign key.
- Expand the **Customer** EJB and the **ACCOUNTS_CUSTOMERS** table. Drag **[0..*] accounts: Account** to **AC_ACCOUNT_FK**.
- Expand the **Account** EJB. Drag the **[0..*] customers: Customer** to **AC_CUST_FK**.

The Outline view of the mapping editor summarizes the mapping activities (Figure 16-36).



```
RAD7EJB <--> ITSOBANK
  Customer <--> CUSTOMER
    ssn : java.lang.String <--> SSN : VARCHAR(12)
    title : java.lang.String <--> TITLE : VARCHAR(8)
    firstName : java.lang.String <--> FIRSTNAME : VARCHAR(32)
    lastName : java.lang.String <--> LASTNAME : VARCHAR(32)
    [0..*] accounts : Account <--> AC_ACCOUNT_FK : ACCOUNT
  Account <--> ACCOUNT
    id : java.lang.String <--> ID : VARCHAR(16)
    balance : int <--> BALANCE : INTEGER
    [0..*] customers : Customer <--> AC_CUST_FK : CUSTOMER
  Transaction <--> TRANSACTIONS
    id : java.lang.String <--> ID : VARCHAR(250)
    amount : int <--> AMOUNT : INTEGER
    timestamp : java.util.Date <--> TRANSTIME : TIMESTAMP
    [0..1] account : Account, [0..*] transactions : Transaction <--> TRANS_ACCOUNT_FK : ACCOUNT
  Credit <--> TRANSACTIONS
  Debit <--> TRANSACTIONS
```

Figure 16-36 Outline view of Mapping editor

- Save and close the Mapping editor.

The mapping file `Map.mapxmi` is stored under `META-INF/backends/DERBY_V101_1`. Based on the mapping JDBC code will be generated when we deploy the EJBs to the server.

Implementing the session facade

The front-end application communicates with the EJB model through a session facade. This design patterns makes the entity EJBs invisible to the EJB client.

The next EJB that we have to build is the session facade: The `Bank` stateless session bean (Figure 16-37).

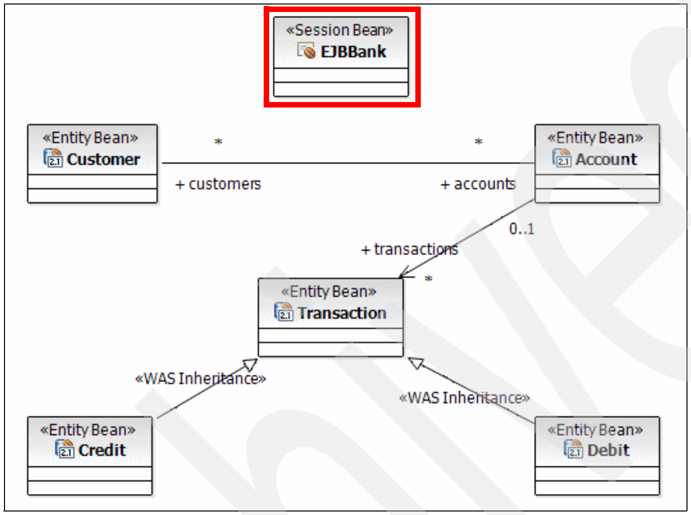
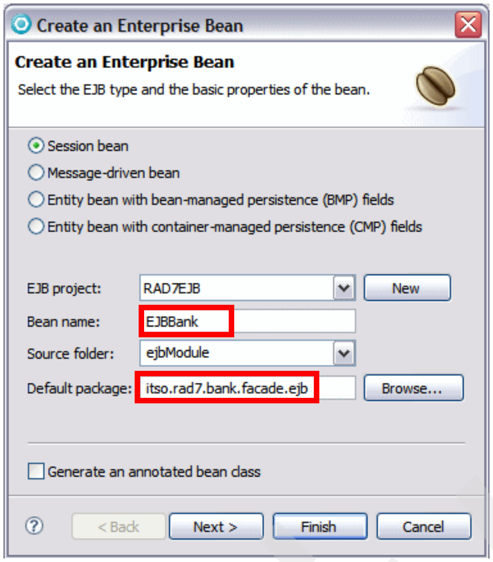


Figure 16-37 Business model session facade

Creating the EJBBank session bean

To create the session bean, do these steps:

- Open the **Diagrams/ejbs.dnx** diagram in the Diagram editor.
- In the Palette view, select **EJB** → **Session Bean** and click the canvas.
- In the Create an Enterprise Bean dialog enter the following (Figure 16-38), and then click **Next**:
  - EJB project: **RAD7EJB**.
  - Bean name: `EJBBank`
  - Source folder: `ejbModule`
  - Default package: `itso.rad7.bank.`**`facade`**`.ejb`



Create an Enterprise Bean

Create an Enterprise Bean

Select the EJB type and the basic properties of the bean.

● Session bean
○ Message-driven bean
○ Entity bean with bean-managed persistence (BMP) fields
○ Entity bean with container-managed persistence (CMP) fields

EJB project: RAD7EJB

Bean name: EJBBank

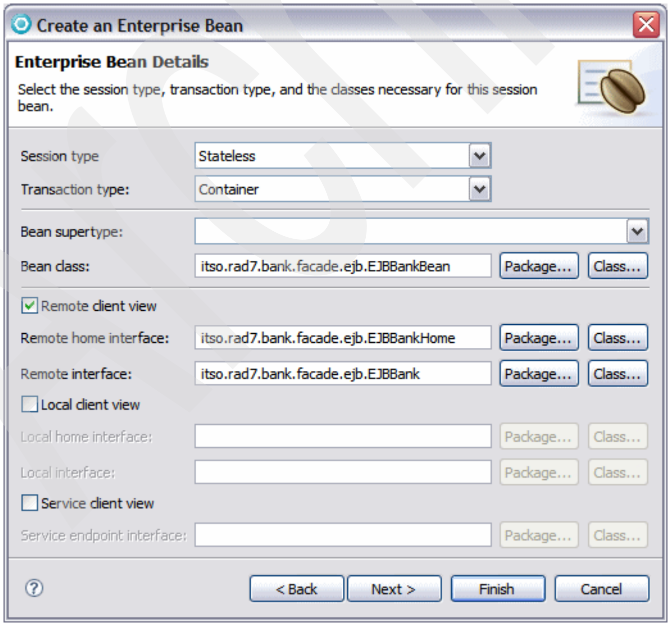
Source folder: ejbModule

Default package: itso.rad7.bank.facade.ejb

□ Generate an annotated bean class

Figure 16-38 Creating a new session bean (1)

- In the Enterprise Bean Details dialog, ensure that **Stateless** is selected for Session type and **Container** is selected in the Transaction type (Figure 16-39) and click **Finish**.



Create an Enterprise Bean

Enterprise Bean Details

Select the session type, transaction type, and the classes necessary for this session bean.

Session type: Stateless

Transaction type: Container

Bean supertype:

Bean class: itso.rad7.bank.facade.ejb.EJBBankBean

☑ Remote client view

Remote home interface: itso.rad7.bank.facade.ejb.EJBBankHome

Remote interface: itso.rad7.bank.facade.ejb.EJBBank

□ Local client view

Local home interface:

Local interface:

□ Service client view

Service endpoint interface:

Figure 16-39 Creating a new session bean (2)

Note that the default selection for the session bean is to create a remote client view instead of a local client view. This is because the environment knows that session beans are normally used to implement the facade and, as such, require remote interfaces as opposed to local interfaces.

- Save and close the class diagram.

Completing the session bean

We now add the facade methods that are used by clients to perform banking operations.

> **Tip:** The Java code for this section can be copied from the file `c:\7501code\ejb\source\EJBBankBean.jpage`.

> **Integer versus BigDecimal**: The database and the entity EJBs keep the balance and amounts as integers (`int`). The data transfer objects in the `RAD7EJBJava` project and in the Web front-end define the balance and amounts as `BigDecimal`. We have to convert between the two formats by dividing integer values and multiplying decimal values by one hundred.
>
> These conversions are done in the session facade (`EJBBank`) so that the view and the model are unaware of the difference.

- Open the **EJBBankBean** (under **Deployment Descriptor** → **Session Beans** → **EJBBank**) in the Java source editor.
- Add the import statements and a constant:

```
import itso.rad7.bank.exception.*;
import itso.rad7.bank.model.*;
import itso.rad7.bank.model.ejb.*;

import java.util.Collection;
import java.util.Iterator;
import java.util.Set;
import java.math.BigDecimal;
import java.sql.Timestamp;

// after class EJBBank
// constant for conversion between int and BigDecimal
private static BigDecimal hundred = new BigDecimal(100);
```

Add the getCustomer method

The `getCustomer` method retrieves a customer by ssn:

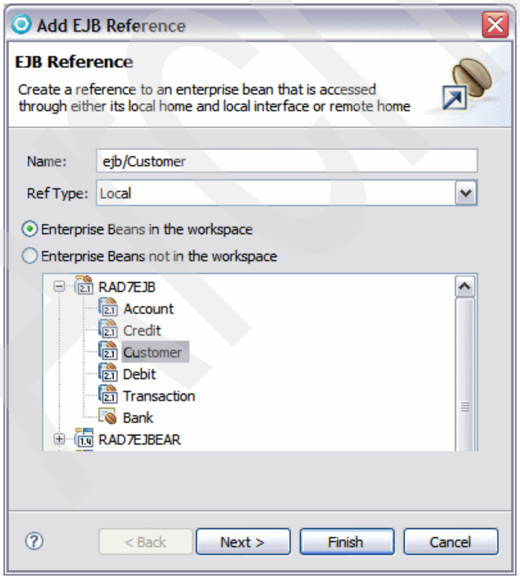
- Complete the **`getCustomer`** method:
  - Enter the method stub for the `getCustomer` after the last method of the `EJBBankBean` class:

    ```
    public Customer getCustomer(String ssn)
          throws InvalidCustomerException { }
    ```

    Note that an error is reported: `This method must return a result of type Customer`. The method code is not complete yet.
  - Place the cursor in the `getCustomer` method body between `{ }`.
  - Switch to the Snippets view, expand **EJB** and double-click **Call an EJB "find" method**.

    > **Tip:** The Snippets view is usually located in the same pane as the Properties view.

  - In the Insert EJB Find wizard click **New EJB Reference**.
  - In the Add EJB Reference dialog, select **Enterprise Beans in the workspace**, expand and select **RAD7EJB** → **Customer**, ensure that **Local** is selected for RefType, and click **Finish** (Figure 16-40).



Add EJB Reference

EJB Reference

Create a reference to an enterprise bean that is accessed through either its local home and local interface or remote home

Name: ejb/Customer

Ref Type: Local

● Enterprise Beans in the workspace

○ Enterprise Beans not in the workspace

Figure 16-40 Adding the ejb/Customer EJB reference

– In Insert EJB Find wizard, the reference **ejb/Customer** is selected. Click **Next**.

> **Note:** The name defaults to `ejb/<BeanName>`, where *<BeanName>* is the name of the bean that you are adding a reference to. You can change this, but we use the default.

– In the Select Method page, select **findByPrimaryKey(String primaryKey)** and click **Next**.

– In the Enter Parameter Values page overtype the value of the first row with **ssn** and click **Finish**.

Several things have happened when we return to the Java editor:

– A new EJB reference has been added to the Bank session bean, visible in the descriptor.

– A new line of code has been added at the current cursor location:

```
CustomerLocal aCustomerLocal =
            find_CustomerLocalHome_findByPrimaryKey(ssn);
```

– The new local method `find_CustomerLocalHome_findByPrimaryKey` has been added after the last method of the class:

```
protected CustomerLocal find_CustomerLocalHome_findByPrimaryKey(
         String primaryKey) {
   CustomerLocalHome aCustomerLocalHome = (CustomerLocalHome)
         ServiceLocatorManager
            .getLocalHome(STATIC_CustomerLocalHome_REF_NAME,
               STATIC_CustomerLocalHome_CLASS);
   try {
      if (aCustomerLocalHome != null)
         return aCustomerLocalHome.findByPrimaryKey(primaryKey);
   } catch (javax.ejb.FinderException fe) {
      // TODO Auto-generated catch block
      fe.printStackTrace();
   }
   return null;
}
```

– Several constants (and imports) have been added to the class:

```
... String STATIC_CustomerLocalHome_REF_NAME = "ejb/Customer";
... Class STATIC_CustomerLocalHome_CLASS = CustomerLocalHome.class;
```

– Finally, the `serviceLocatorMgr.jar` is added to the `RAD7EJBEAR` Enterprise Application. This JAR, which contains the implementation of the `ServiceLocatorManager` class is also added to the Java build path and the Java JAR Dependencies for the `RAD7EJB` project.

- Modify the find_CustomerLocalHome_findByPrimaryKey method as shown below. The difference, which consists of throwing an application-specific exception when the customer cannot be found, is highlighted in bold:

```
protected CustomerLocal find_CustomerLocalHome_findByPrimaryKey
      (String primaryKey) throws InvalidCustomerException {
   ......
   try {
      if (aCustomerLocalHome != null)
         return aCustomerLocalHome.findByPrimaryKey(primaryKey);
   } catch (javax.ejb.FinderException fe) {
      // Customer Home not found
      throw new InvalidCustomerException(primaryKey);
   } ......
```

- Complete the getCustomer method. The method uses the find_CustomerLocalHome_findByPrimaryKey method to look up the customer in the database and then builds a data transfer object and returns that to the caller.

```
public Customer getCustomer(String ssn) throws InvalidCustomerException{
   CustomerLocal aCustomerLocal =
               find_CustomerLocalHome_findByPrimaryKey(ssn);
   Customer customer = new Customer(ssn, aCustomerLocal.getTitle(),
         aCustomerLocal.getFirstName(), aCustomerLocal.getLastName());
   return customer;
}
```

Add the getAccount method

The getAccount method retrieves an account by account ID:

- Complete the **getAccount** method using a similar approach to the getCustomer method. You have to first add the skeleton getAccount method and add the EJB Reference for Account bean. The finished getAccount method is listed here:

```
public Account getAccount(String accountNumber)
         throws InvalidAccountException {
   AccountLocal anAccountLocal =
      find_AccountLocalHome_findByPrimaryKey(accountNumber);
   Account account = new Account(accountNumber, new BigDecimal(
            anAccountLocal.getBalance()).divide(hundred));
   return account;
}
```

The generated find_AccountLocalHome_findByPrimaryKey method is similar to the matching customer method and throws InvalidAccountException.

Add the remaining business logic

After implementing the necessary utility functions to look up accounts and customers, we can implement the business methods:

- Updating a customer (**`updateCustomer`**)
- Retrieving the accounts for a customer (**`getAccounts`**)
- Retrieve the transactions for an account (**`getTransactions`**)
- Retrieve customers by partial name (**`getCustomers`**)

These methods are shown in Example 16-2. You can simply copy the four methods from `c:\7501code\ejb\source\EJBBankBean.jpage`.

Example 16-2 Completed business logic methods

```
public void updateCustomer(String ssn, String title, String firstName,
         String lastName) throws InvalidCustomerException {
   CustomerLocal aCustomerLocal =
            find_CustomerLocalHome_findByPrimaryKey(ssn);
   aCustomerLocal.setTitle(title);
   aCustomerLocal.setFirstName(firstName);
   aCustomerLocal.setLastName(lastName);
}

public Account[] getAccounts(String ssn) throws InvalidCustomerException {
   CustomerLocal aCustomerLocal =
         find_CustomerLocalHome_findByPrimaryKey(ssn);
   Collection colAccounts = aCustomerLocal.getAccounts();
   Iterator itAccounts = colAccounts.iterator();
   Account[] arrAccount = new Account[colAccounts.size()];
   int i = 0;
   while (itAccounts.hasNext()) {
      AccountLocal accountLocal = (AccountLocal) itAccounts.next();
      Account account = new Account();
      account.setAccountNumber(accountLocal.getPrimaryKey().toString());
      account.setBalance(accountLocal.getBalance().divide(hundred));
      arrAccount[i++] = account;
   }
   return arrAccount;
}

public Transaction[] getTransactions(String accountNumber)
         throws InvalidAccountException {
   AccountLocal anAccountLocal =
               find_AccountLocalHome_findByPrimaryKey(accountNumber);
   Set setTransactions = anAccountLocal.getLog();
   Iterator itTransactions = setTransactions.iterator();
   Transaction[] arrTransaction = new Transaction[setTransactions.size()];
   for (int i=0; itTransactions.hasNext(); i++) {
      TransactionLocal transactionLocal = (TransactionLocal) itTransactions
```

```
                                                          .next();
      Transaction transaction = null;
      if (transactionLocal instanceof CreditLocal) {
         transaction = new Credit(
            (new BigDecimal(transactionLocal.getAmount())).divide(hundred));
      } else {
         transaction = new Debit(
            (new BigDecimal(transactionLocal.getAmount())).divide(hundred));
      }
      transaction.setTimeStamp
                (new Timestamp(transactionLocal.getTimestamp().getTime()));
      arrTransaction[i] = transaction;
      sort: for (int j=0; j<i; j++) {
         if ( transaction.getTimeStamp().before
                             ( arrTransaction[j].getTimeStamp() ) ) {
            for (int k=i-1; k>=j; k--)
                  { arrTransaction[k+1] = arrTransaction[k]; }
            arrTransaction[j] = transaction;
            break sort;
         }
      }
   }
   return arrTransaction;
}

public Customer[] getCustomers(String partialName) throws ITSOBankException {
   CustomerLocalHome aCustomerLocalHome = (CustomerLocalHome)
      ServiceLocatorManager.getLocalHome(STATIC_CustomerLocalHome_REF_NAME,
                                         STATIC_CustomerLocalHome_CLASS);
   try {
      Collection aCollection = aCustomerLocalHome.findByName(partialName);
      Iterator itCustomer = aCollection.iterator();
      Customer[] arrCustomer = new Customer[aCollection.size()];
      int i = 0;
      while (itCustomer.hasNext()) {
         CustomerLocal aCustomerLocal = (CustomerLocal) itCustomer.next();
         Customer cust = new Customer
                  (aCustomerLocal.getPrimaryKey().toString(),
                  aCustomerLocal.getTitle(), aCustomerLocal.getFirstName(),
                  aCustomerLocal.getLastName());
         arrCustomer[i++] = cust;
      }
      return arrCustomer;
   } catch (Exception e) {
      throw new ITSOBankException("Customers not found: " + partialName);
   }
}
```

Transaction methods deposit, withdraw, and transfer

To implement the transaction methods, we require code to create a new transaction. The method is similar to creating code for an EJB lookup method:

- Create a method stub for the **deposit** method:

  ```
  public void deposit(String accountNumber, BigDecimal amount)
        throws InvalidAccountException, ITSOBankException { }
  ```

  – In the Snippets view double-click **EJB** → **Call an EJB “create” method**.

  – In the Insert EJB Create wizard click **New EJB Reference**, and add a reference to the **Credit** bean, then click **Finish**.

- Similar resources to the Insert EJB Find wizard are created for you:

  – An EJB reference named ejb/Credit is created for the entity bean.
  – A createCreditLocal method is created in the EJBBankBean class.
  – Constants and imports are added.

- Modify the generated createCreditLocal method:

  ```
  protected CreditLocal createCreditLocal(int amount)
        throws ITSOBankException {
     CreditLocalHome aCreditLocalHome = (CreditLocalHome)
        erviceLocatorManager
           .getLocalHome(STATIC_CreditLocalHome_REF_NAME,
                 STATIC_CreditLocalHome_CLASS);
     try {
        if (aCreditLocalHome != null)
           return aCreditLocalHome.create(amount);
     } catch (javax.ejb.CreateException ce) {
        throw new ITSOBankException(
              "Unable to create credit transaction (amount="+amount+")");
     }
     return null;
  }
  ```

- Complete the deposit method:

  ```
  public void deposit(String accountNumber, int amount)
        throws InvalidAccountException, ITSOBankException {
     CreditLocal aCreditLocal = createCreditLocal
                                 (amount.multiply(hundred).intValue());
     AccountLocal anAccountLocal =
           find_AccountLocalHome_findByPrimaryKey(accountNumber);
     anAccountLocal.processTransaction(aCreditLocal);
  }
  ```

- Implement the **`withdraw`** method, using the same process as the deposit method, except that you use the Insert EJB Create wizard to generate code to create a `Debit` bean. The method is listed here:

```
public void withdraw(String accountNumber, BigDecimal amount)
          throws InvalidAccountException, InvalidAmountException,
          ITSOBankException {
   DebitLocal aDebitLocal = createDebitLocal
                                (amount.multiply(hundred).intValue());
   AccountLocal anAccountLocal =
          find_AccountLocalHome_findByPrimaryKey(accountNumber);
   anAccountLocal.processTransaction(aDebitLocal);
}
```

  Add the exception to the generated `createDebitLocal` method.

- Add the **transfer** method (no new reference is required):

```
public void transfer(String debitAccountNumber,
          String creditAccountNumber, BigDecimal amount)
      throws InvalidAccountException, InvalidAmountException,
                ITSOBankException {
   withdraw(debitAccountNumber, amount);
   deposit(creditAccountNumber, amount);
}
```

Promote the methods to the remote interface

Promote all the business methods to the remote interface:

- In the Outline view, select the business methods using the `Ctrl` key, then select **Enterprise Bean** → **Promote to Remote Interface**.

```
getCustomer(String)
getAccount(String)
updateCustomer(String, String, String, String)
getAccounts(String)
getTransactions(String)
getCustomers(String)
deposit(String, int)
withdraw(String, int)
transfer(String, String, int)
```

  Notice that the R-shaped icons next to methods appears. The methods are added to the `Bank` interface.

The `EJBBank` session bean is now complete. In the following sections we first test the EJBs using the Universal Test Client and then proceed to integrate the EJBs with the sample Web application.

Generating the deployed code

When the definitions of the EJBs are complete it is good practice to generate the deployed code. Deployment of EJBs generates many helper classes based on the mapping of the entity EJBs to database tables. To generate the deployed code:

- Right-click the **RAD7EJB** project and select **Prepare for Deployment**.
- A number of packages and classes are generated into the EJB and EJB client project, for example:
  - Concrete classes (`ConcreteCustomer_xxxxxx`) that implement the abstract EJB entity classes
  - `EJSCMPXxxxxHomeBean` classes
  - `EJSLocalCMPxxxxx` classes for entity beans (local interface)
  - `EJSRemoteStatelessXxxxxx` classes for session beans (remote interface)
  - Packages `itso.rad7.bank.model.ejb.websphere_deploy` and `itso.rad7.bank.model.ejb.websphere_deploy.DERBY_V101_1`, which contain the mapping implementation. Open the `XxxxxBeanFunctionSet` class and you can see the generated SQL statements.
  - `com.ibm.ejs.container` and `com.ibm.ws.ejbdeploy.xxx` packages with helper code

Note: If you encounter errors with the deployment, import the interchange file **RAD7EJBonly.zip** file from `C:\7501code\zInterchangeFiles\ejb`, and deploy again: Right-click the **RAD7EJB** project and select **Prepare for Deployment**.

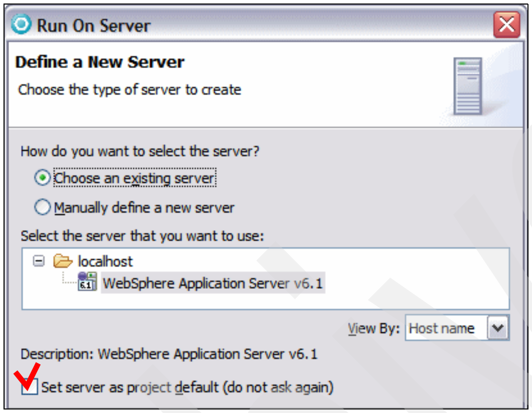
Testing EJBs with the Universal Test Client

Before we integrate the EJB application with the Web application, imported in "Importing a Web project and a Java project" on page 736, we test the entity and session session beans to see that they work as expected. We use the enterprise application Universal Test Client (UTC), which is contained in Application Developer.

In this section we describe some of the operations you can perform with the Universal Test Client. We use the test client to find the `Customer` EJB home, find and create instances of the `Customer` bean, and send messages to those instances.

To test the EJBs, do these steps:

- Start the server if it is not running, then add the `RAD7EJBEAR` project to the server using **Add and Remove Projects**.
- Expand **RAD7EJB** → **Deployment Descriptor:RAD7EJB** → **Entity Beans** → **Customer** and select **Run As** → **Run on Server**.
- In the Run on Server dialog, select **WebSphere Application Server v6.1** and click **Finish** (Figure 16-41). Optionally, select **Set server as project default**.



Define a New Server

Choose the type of server to create

How do you want to select the server?

● Choose an existing server

○ Manually define a new server

Select the server that you want to use:

localhost

WebSphere Application Server v6.1

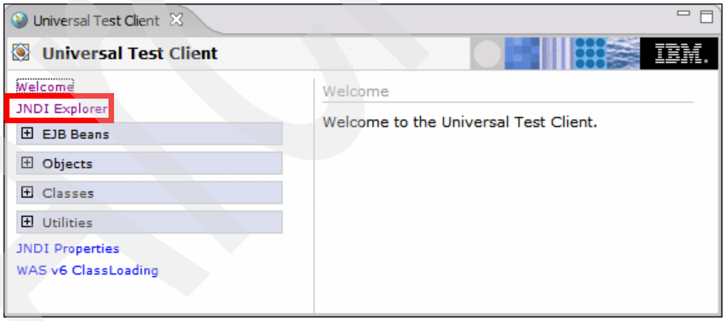
View By: Host name

Description: WebSphere Application Server v6.1

☑ Set server as project default (do not ask again)

Figure 16-41 Run On Server dialog

- In the Universal Test Client Welcome page click **JNDI Explorer** (Figure 16-42).



Welcome to the Universal Test Client.

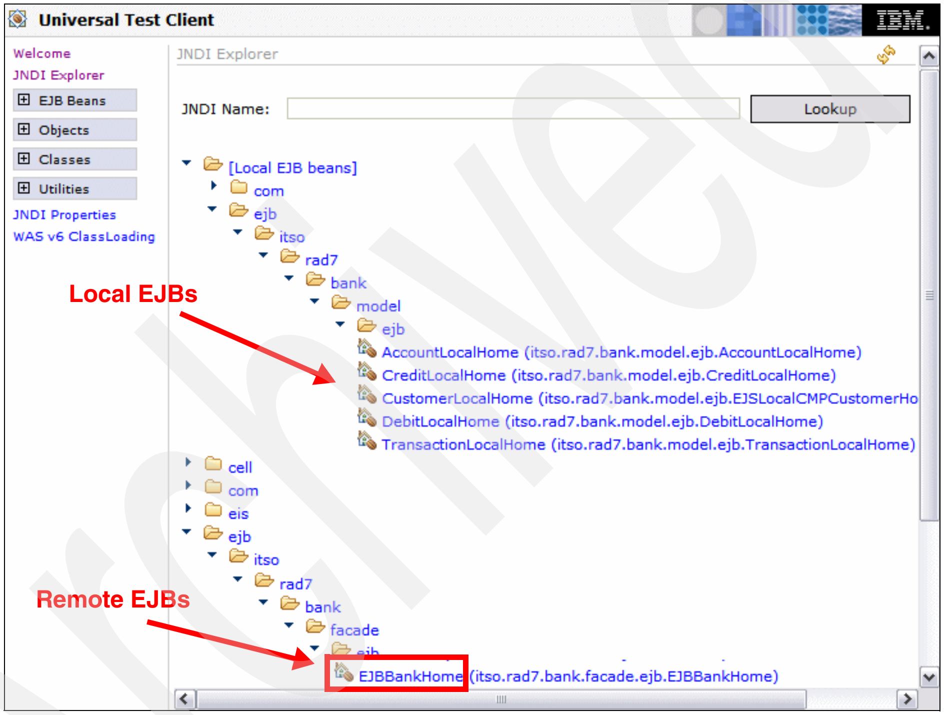
Figure 16-42 Universal Test Client (UTC): Home page

> **Tip:** The default URL of the test client is `http://localhost:9080/UTC/`, so you can also access it through an external browser. If you want to access it from another machine, just substitute `localhost` with the host name or IP address of the developer machine.

- In the JNDI Explorer page (Figure 16-43) expand both **[Local EJB Beans]** → **ejb** → **itso** → **rad7** → **bank** → **model** → **ejb** and **ejb** → **itso** → **rad7** → **bank** → **facade** → **ejb**.

  Notice that the five entity beans appear in the section for local EJBs, because they have no remote interface, while the `EJBBank` session bean appears in the remote scope.

  Also, the data source that we defined in "Configuring the data source for the ITSOBANK" on page 740, appears in the JNDI explorer view under **jdbc**. All EJBs appear as Web links.



```
[Local EJB beans]
  com
  ejb
    itso
      rad7
        bank
          model
            ejb
              AccountLocalHome (itso.rad7.bank.model.ejb.AccountLocalHome)
              CreditLocalHome (itso.rad7.bank.model.ejb.CreditLocalHome)
              CustomerLocalHome (itso.rad7.bank.model.ejb.EJSLocalCMPCustomerHo
              DebitLocalHome (itso.rad7.bank.model.ejb.DebitLocalHome)
              TransactionLocalHome (itso.rad7.bank.model.ejb.TransactionLocalHome)
cell
com
eis
ejb
  itso
    rad7
      bank
        facade
          ejb
            EJBBankHome (itso.rad7.bank.facade.ejb.EJBBankHome)
```

Figure 16-43 UTC JNDI Explorer

Working with the EJBBank session bean

First we test the session bean:

- Click **EJBBankHome** (`itso.rad7.bank.facade.ejb.EJBBankHome`). The result is that the `EJBBank` bean is added to the EJB Beans list.
- Expand **EJB Beans** → **EJBBank** → **EJBBankHome** and select **EJBBank create()**.
- The method signature for the create method is displayed. Click **Invoke**.

- Click **Work with Object** and an `EJBBank 1` instance is added (Figure 16-44).

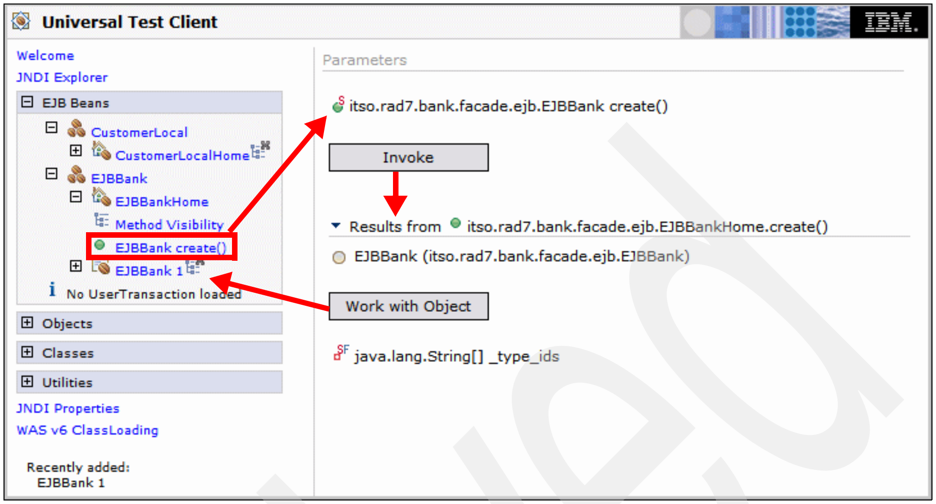
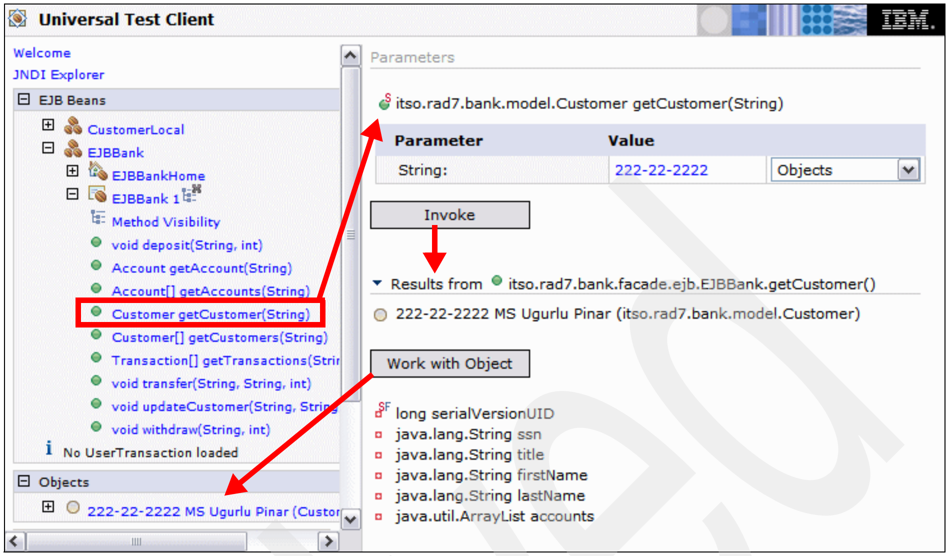


Figure 16-44 UTC: Session bean instance

- Select **EJB Beans** → **EJBBank** → **EJBBank 1** → **Customer getCustomer(..)** (Figure 16-45):
  - The method signature for the method is displayed with a table, allowing you to specify the method parameters (in this sample only one `String` parameter).
  - Enter `222-22-2222` as parameter value and click **Invoke**.
  - The method runs and returns a `Customer` data transfer object (DTO).
  - Click **Work with Object** and the DTO is added to the Objects section (it is not an EJB).
  - You can expand the object and run its methods.


Universal Test Client
IBM.
Welcome
JNDI Explorer
EJB Beans
CustomerLocal
EJBBank
EJBBankHome
EJBBank 1
Method Visibility
void deposit(String, int)
Account getAccount(String)
Account[] getAccounts(String)
Customer getCustomer(String)
Customer[] getCustomers(String)
Transaction[] getTransactions(Strir
void transfer(String, String, int)
void updateCustomer(String, Strin
void withdraw(String, int)
No UserTransaction loaded
Objects
222-22-2222 MS Ugurlu Pinar (Custor
Parameters
Invoke
Work with Object

itso.rad7.bank.model.Customer getCustomer(String)

Parameter	Value	
String:	222-22-2222	Objects

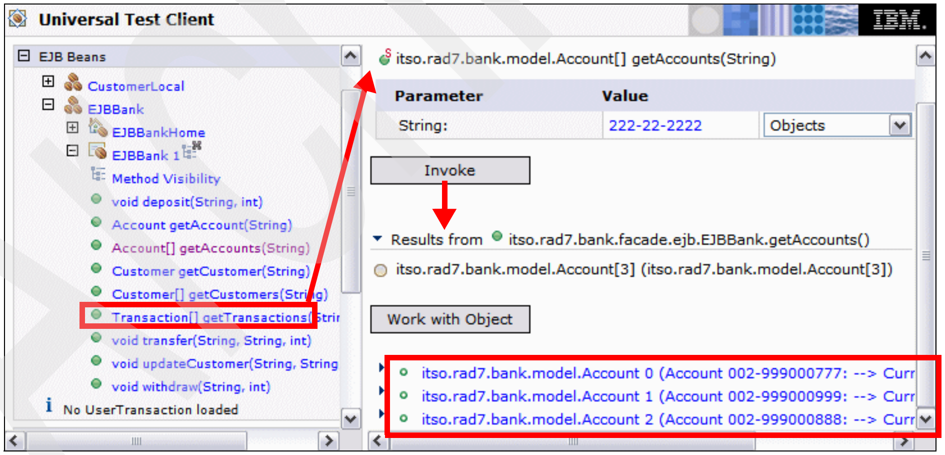
Results from itso.rad7.bank.facade.ejb.EJBBank.getCustomer()

222-22-2222 MS Ugurlu Pinar (itso.rad7.bank.model.Customer)

```
long serialVersionUID
java.lang.String ssn
java.lang.String title
java.lang.String firstName
java.lang.String lastName
java.util.ArrayList accounts
```

Figure 16-45 UTC: Retrieve a customer

- Select the **Account[] getAccounts(String)** method. Enter `222-22-2222` as parameter value and click **Invoke**. Three `Account` DTOs are returned (Figure 16-46).


Universal Test Client
IBM.
EJB Beans
CustomerLocal
EJBBank
EJBBankHome
EJBBank 1
Method Visibility
void deposit(String, int)
Account getAccount(String)
Account[] getAccounts(String)
Customer getCustomer(String)
Customer[] getCustomers(String)
Transaction[] getTransactions(Strir
void transfer(String, String, int)
void updateCustomer(String, String
void withdraw(String, int)
No UserTransaction loaded
Invoke
Work with Object

itso.rad7.bank.model.Account[] getAccounts(String)

Parameter	Value	
String:	222-22-2222	Objects

Results from itso.rad7.bank.facade.ejb.EJBBank.getAccounts()

itso.rad7.bank.model.Account[3] (itso.rad7.bank.model.Account[3])

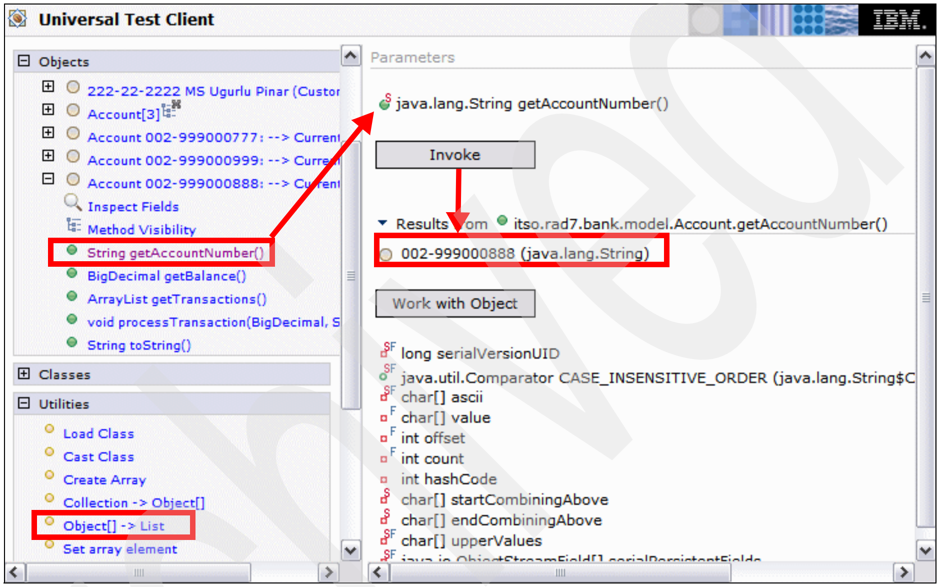
```
itso.rad7.bank.model.Account 0 (Account 002-999000777: --> Curr
itso.rad7.bank.model.Account 1 (Account 002-999000999: --> Curr
itso.rad7.bank.model.Account 2 (Account 002-999000888: --> Curr
```

Figure 16-46 UTC: Invoking getAccounts for a customer

- Click **Work with Object**. Notice how the object is added to the **Objects** compartment and not to EJB Beans.

The test client cannot inspect objects in arrays, so we will have to convert it to a `java.util.List`:

- Expand **Utilities** and click **Object[]** → **List**.
- In the Object[] → List page, select **Account[3]** and click **Convert**.
- Click **Work with Contained Objects** and three `Account` objects are added to the Objects compartment. Invoke their `getAccountNumber` and `getBalance` methods to inspect their attributes (Figure 16-47).


Universal Test Client
IBM.
Objects
222-22-2222 MS Ugurlu Pinar (Custor
Account[3]
Account 002-999000777: --> Current
Account 002-999000999: --> Curre
Account 002-999000888: --> Current
Inspect Fields
Method Visibility
String getAccountNumber()
BigDecimal getBalance()
ArrayList getTransactions()
void processTransaction(BigDecimal, S
String toString()
Classes
Utilities
Load Class
Cast Class
Create Array
Collection -> Object[]
Object[] -> List
Set array element
Parameters
java.lang.String getAccountNumber()
Invoke
Results from itso.rad7.bank.model.Account.getAccountNumber()
Work with Object
long serialVersionUID
java.util.Comparator CASE_INSENSITIVE_ORDER (java.lang.String$C
char[] ascii
char[] value
int offset
int count
int hashCode
char[] startCombiningAbove
char[] endCombiningAbove
char[] upperValues

002-999000888 (java.lang.String)

Figure 16-47 UTC: Working with array results

Running transactions

To run a deposit, select **EJBBank 1** → **void deposit(..)**, enter `002-999000777` as the first parameter (account number) and `222` as the second parameter (amount), then click **Invoke**. A message: `The method completed successfully` is displayed (the method has no result).

Working with entity beans

We can also work directly with the local entity beans:

- We already have the **CustomerLocalHome** under EJB Beans. (We can also find it in the JNDI Explorer under [Local EJB Beans].
- Select the **findByName** method:
  - Enter the parameter value **%i%** (remember that the EJB QL statement looks for **lastname like ?1**).

  - Click **Invoke**, then click **Work with Contained Objects**.
  - Three `CustomerLocal` are added to EJB Beans. If you run their `getLastName` method you can verify that the names contain the letter i.
- In the JNDI Explorer locate and select **[Local EJB Beans]** → **AccountLocalHome**. Expand **AccountLocal** bean under EJB Beans:
  - Execute the **findByPrimaryKey** method with `002-999000777` as parameter.
  - Click **Work with Object** to add the `AccountLocal` instance EJB.
  - Expand the instance and invoke the `getId` and `getBalance` methods. Note that relationship methods with collections do not work in UTC.
- To create an account you first have to retrieve a customer and then run the create method on the account home:
  - Under the **CustomerLocalHome** invoke the **findByPrimaryKey** method with a ssn (`000-00-0000`).
  - Use the JNDI Explorer to locate **AccountLocalHome**.
  - Select the **create** method and select the `CustomerLocal` object as parameter. Click **Invoke** to add an account to the customer.

You can play with the UTC to make sure all of your EJBs work. When you are done, close the UTC window.

Adapting the Web application

The Web application was imported in "Importing a Web project and a Java project" on page 736. This is the same application that was developed in Chapter 12, "Develop Web applications using JSPs and servlets" on page 465. However, instead of the memory bank we are using the EJB implementation of the bank with the relational database.

> **Persistence:** In the first implementation, every time you started the Web application you got the same data because it was created from memory. Now we are running with EJBs accessing the underlying `ITSOBANK` database. All the updates are persistent. The updated balance is stored in the database and the transaction records accumulate for each account.

We now add client code to the Web application to utilize the EJBs that we developed and tested in this chapter. A skeleton **`ITSOBank`** class is already in the Web application. We have to implement the methods of the `Bank` interface.

Tip: The Java code for the `ITSOBank` class can be copied from the file `c:\7501code\ejb\source\ITSOBank.java`.

Implementing the ITSOBank class

Open the `ITSOBank` class in the **RAD7EJBWeb** → **Java Resources** → **itso.rad7.bank.impl** package.

We have to access the **EJBBank** session bean to run with EJBs:

- Add the following code to the class:

```
// add two import statements
import itso.rad7.bank.facade.ejb.EJBBank;
import itso.rad7.bank.facade.ejb.EJBBankHome;

// add before the first method of the class
private EJBBank ejbBank = null;;

// add before the first method of the class
private EJBBank getEJBBank() throws ITSOBankException {
   if (ejbBank == null) {
      // place the cursor on the next line

   }
   return ejbBank;
}
```

- Place the cursor inside the `if` statement and double-click **EJB** → **Call an EJB "create" method** in the Snippets view.
- In the Insert EJB Create wizard click **New EJB Reference**.
- In the Add EJB Reference dialog, select **Enterprise Beans in the workspace**, expand and select **RAD7EJBEAR** → **RAD7EJB** → **EJBBank**, ensure that **Remote** is selected in the Ref type drop-down, and click **Next**.

  Select **Use default connect properties for doing a lookup on this reference** and click **Finish**.
- When you return the Insert EJB Create wizard, click **Finish**.
- The wizard generates a line into the `getEJBBank` method and a new `createEJBBank` method (at the end).
- Complete the `getBankEJB` method with the code from the generated `createEJBBank` method, and delete the `createEJBBank` method:

```
private EJBBank getBankEJB() throws ITSOBankException {
   if (ejbBank == null) {
      EJBBankHome anEJBBankHome = (EJBBankHome) ServiceLocatorManager
         .getRemoteHome(STATIC_EJBBankHome_REF_NAME,
                        STATIC_EJBBankHome_CLASS);
      try {
         if (anEJBBankHome != null)
            ejbBank = anEJBBankHome.create();
         } catch (Exception e) {
         throw new ITSOBankException("Unable to create Bank EJB: "
               + STATIC_EJBBankHome_REF_NAME + " " + e.getMessage());
      }
   }
   return ejbBank;}

// delete the createEJBBank method
```

- Select **Source** → **Organize Imports** to resolve the `ITSOBankException`.

Implement the Bank interface

Replace the method stubs with the methods shown in Example 16-3. These methods forward all calls to the `EJBBank` session EJB.

Example 16-3 Completed facade methods invoke the EJBBANK session EJB

```
public Customer searchCustomerBySsn(String ssn)
         throws InvalidCustomerException {
   try {
      return getEJBBank().getCustomer(ssn);
   } catch (Exception e) {
      throw new InvalidCustomerException(ssn + " (" + e.getMessage() +")");
   }
}
public ArrayList<Account> getAccountsForCustomer(String customerSsn)
         throws InvalidCustomerException {
   try {
      return new ArrayList<Account>(Arrays.asList(
                              getEJBBank().getAccounts(customerSsn) ));
   } catch (Exception e) {
      throw new InvalidCustomerException("Unable to retrieve accounts for: "
                        + customerSsn + " (" + e.getMessage() +")");
   }
}
public ArrayList<Transaction> getTransactionsForAccount
                                    (String accountNumber)
         throws InvalidAccountException {
   try {
      return new ArrayList<Transaction>
            (Arrays.asList(getEJBBank().getTransactions(accountNumber)));
   } catch (Exception e) {
      throw new InvalidAccountException("Unable to retrieve transactions for:"
                + accountNumber + " (" + e.getMessage() +")");
  }
}
public Account searchAccountByAccountNumber(String accountNumber)
      throws InvalidAccountException {
  try {
    return getEJBBank().getAccount(accountNumber);
  } catch (Exception e) {
    throw new InvalidAccountException(accountNumber
                    + " (" + e.getMessage() +")");
  }
}
public void deposit(String accountNumber, BigDecimal amount)
      throws InvalidAccountException, InvalidTransactionException {
  try {
    getEJBBank().deposit(accountNumber, amount);
  } catch (Exception e) {
    throw new InvalidTransactionException("Unable to deposit " + amount
            + " to " + accountNumber + " (" + e.getMessage() +")");
  }
}
public void withdraw(String accountNumber, BigDecimal amount)
      throws InvalidAccountException, InvalidTransactionException {
  try {
    getEJBBank().withdraw(accountNumber, amount);
  } catch (Exception e) {
    throw new InvalidAccountException("Unable to withdraw " + amount +
            " from " + accountNumber + " (" + e.getMessage() +")");
  }
}
public void transfer(String debitAccountNumber, String creditAccountNumber,
      BigDecimal amount) throws InvalidAccountException,
      InvalidTransactionException {
  try {
    getEJBBank().transfer(debitAccountNumber, creditAccountNumber, amount);
  } catch (Exception e) {
    throw new InvalidTransactionException("Unable to transfer " + amount +
            " from " + debitAccountNumber + " to "
            + creditAccountNumber + " (" + e.getMessage() +")");
  }
}
public void updateCustomer(String ssn, String title, String firstName,
      String lastName) throws InvalidCustomerException {
  try {
    getEJBBank().updateCustomer(ssn, title, firstName, lastName);
  } catch (Exception e) {
    throw new InvalidCustomerException(ssn + "Unable to update customer ("
                    + e.getMessage() +")");
  }
```

```
}
public Map<String, Customer> getCustomers() {
   Map<String, Customer> customers = new HashMap<String, Customer>();
   try {
      Customer[] custarray = getEJBBank().getCustomers("%");
      for (int i=0; i<custarray.length; i++) {
         customers.put(custarray[i].getSsn(), custarray[i]);
      }
   } catch (Exception e) {
      e.printStackTrace();
   }
   return customers;
}
```

- Select **Source** → **Organize Imports** to resolve the class references.
- Note that we did not implement all the `Bank` interface methods in the `EJBBank` EJB. Some method skeletons in the `ITSOBank` class are empty: `addCustomer`, `closeAccountOfCustomer`, `openAccountForCustomer`, and `removeCustomer`.

Running the Web application

The front-end Web application now works with the EJB session bean, and, therefore, with the database.

To run the Web application:

- Add both `RAD7EJBEAR` and `RAD7EJBWebEAR` to the server using **Add and Remove Projects**.
- Select the **RAD7EJBWeb** project and **Run As** → **Run on Server**.
- Select **redbank**, then enter a social security number (`222-22-2222`) and click **Submit**.

Application problem: If you get an error and the Web application does not connect to the EJB application, then the sequence of the module dependencies is wrong. Here are instructions for how to fix this problem:

- Select the **RAD7EJBWebEAR** project and **Properties** and select the **J2EE Module Dependencies** page:
  - Clear **RAD7EJBWeb.war** and click **Apply**.
  - Select **RAD7EJBWeb.war** and click **Apply** and **OK**.
- After the application is restarted in the server, everything works (hopefully). Note that these dependencies are stored in the **`.settings`** folder of the project.

Follow the instructions in "Testing the Web application" on page 533 to test the Web application running with EJBs.

Note that the transaction listing is sorted by the timestamp. Relationships are retrieved in no particular order by the container. We added extra code to the `EJBBANK` session bean `getTransactions` method to sort the returned objects by timestamp.

The final Web application also includes a `ListCustomer` servlet that uses the `getCustomers` method of the `EJBBank` session bean to list customers by partial last name. You can run this servlet using this URL:

```
http://localhost:9080/RAD7EJBWeb/ListCustomers
http://localhost:9080/RAD7EJBWeb/ListCustomers?partialName=i
```

Complete EJB application interchange file

The completed enterprise applications are available in:

```
C:\7501code\zInterchangeFiles\RAD7EJB.zip
```

More information

For more information on EJB, we recommend the following resources:

- *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819, Redbooks publication, found at:

  `http://www.redbooks.ibm.com/abstracts/sg246819.html`
- *What are Enterprise JavaBeans components? Part 1: The history and goal of EJB Architecture*, white paper, found at:

  `http://www.ibm.com/developerworks/java/library/j-what-are-ejbs/part1/`
- *What are Enterprise JavaBeans components? Part 2: EJB Programming Model*, white paper, found at:

  `http://www.ibm.com/developerworks/java/library/j-what-are-ejbs/part2/`
- *What are Enterprise JavaBeans components? Part 3: Deploying and using Enterprise JavaBeans components*, white paper, found at:

  `http://www.ibm.com/developerworks/java/library/j-what-are-ejbs/part3/`

17

Develop J2EE application clients

This chapter provides an introduction to J2EE application clients and the facilities supplied by the J2EE application client container. In addition, we highlight the features provided by Application Developer for developing and testing J2EE application clients.

The chapter is organized into the following sections:

- Introduction to J2EE application clients
- Overview of the sample application
- Preparing for the sample application
- Developing the J2EE application client
- Testing the J2EE application client
- Packaging the J2EE application client

Introduction to J2EE application clients

A J2EE application server is capable of making several different types of resources available for remote access, such as:

- Enterprise JavaBeans (EJBs)
- JDBC data sources
- Java Message Service (JMS) resources (queues and topics)
- Java Naming and Directory Interface (JNDI) services

These resources are most often accessed from a component that is running within the J2EE application server itself, such as an EJB, servlet, or JSP. However, these resources can also be used from a stand-alone Java application (known as a *J2EE application client*) running in its own Java Virtual Machine (JVM), possibly on a different computer from the server. Figure 17-1 shows the resource access scenarios described.

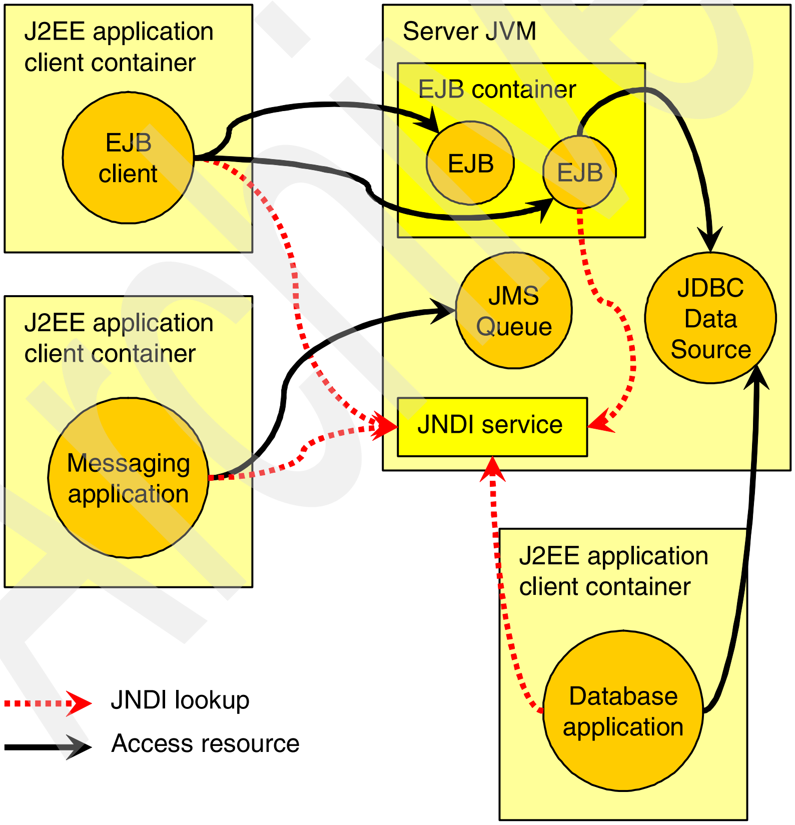


Figure 17-1 Java applications using J2EE server resources

Note: The clients shown in Figure 17-1 might conceptually be running on the same physical node, or even in the same JVM as the application server. However, in this chapter the focus is on clients running in distributed environments. Throughout this chapter, we develop an EJB client, invoking the EJBs from Chapter 16, “Develop Web applications using EJBs” on page 719, to provide a simple ITSO Bank client application.

Because a regular JVM does not support accessing such application server resources, additional setup for the runtime environment is required for a J2EE application. There are two methods to achieve this:

- Add the required packages to the Java Runtime Environment manually.
- Package the application according to the J2EE application client specification and execute the application in a J2EE application client container.

In this chapter, we focus on the second of these options. In addition to providing the correct runtime resources for Java applications accessing J2EE server resources, the J2EE application client container provides additional features, such as mapping references to JNDI names and integration with server security features.

IBM WebSphere Application Server V6.1 includes a J2EE application client container and a facility for launching J2EE application clients. The J2EE application client container, known as *Application Client for WebSphere Application Server*, can be installed separately from the WebSphere Application Server installation CDs, or downloaded from developerWorks, and runs a completely separate JVM on the client machine.

When the JVM starts, it loads the necessary runtime support classes to make it possible to communicate with WebSphere Application Server and to support J2EE application clients that will use server-side resources. Refer to the WebSphere Application Server Information Center for more information about installing and using the Application Client for WebSphere Application Server.

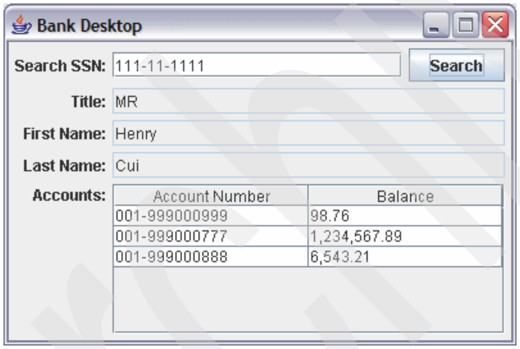
Note: Although the J2EE specification describes the JAR format as the packaging format for J2EE application clients, the Application Client for WebSphere Application Server expects the application to be packaged as a JAR inside an Enterprise Application Archive (EAR). The Application Client for WebSphere Application Server does not support execution of a standalone J2EE client JAR.

Application Developer includes tooling to assist with developing and configuring J2EE application clients, and a test facility that allows J2EE application clients to be executed in an appropriate container. The focus of this chapter is on the Application Developer tooling for J2EE application clients, so we will be looking only at this facility.

Overview of the sample application

The application that we develop in the course of this chapter is a simple J2EE application client application. It invokes the services of the EJB application that was developed in Chapter 16, “Develop Web applications using EJBs” on page 719, to look up the customer information and account overview from a specified customer SSN.

The application uses a graphical user interface, implemented with Swing components, which displays the details for the customer with SSN 111-11-1111 (Figure 17-2).



Search SSN: 111-11-1111

Title: MR

First Name: Henry

Last Name: Cui

Accounts:

Account Number	Balance
001-999000999	98.76
001-999000777	1,234,567.89
001-999000888	6,543.21

Figure 17-2 Interface for the sample application client

Figure 17-3 shows a class diagram of the finished sample application:

- The classes on the right-hand side of the class diagram are classes from the EJB enterprise application, while the three left-hand classes are part of the application client.
- As the class diagram outlines, the application client controller class, `BankDesktopController`, uses the `EJBBank` session EJB to retrieve `Customer` and `Account` object instances, representing the customer and associated account(s) that are retrieved from the `ITSOBANK` database.

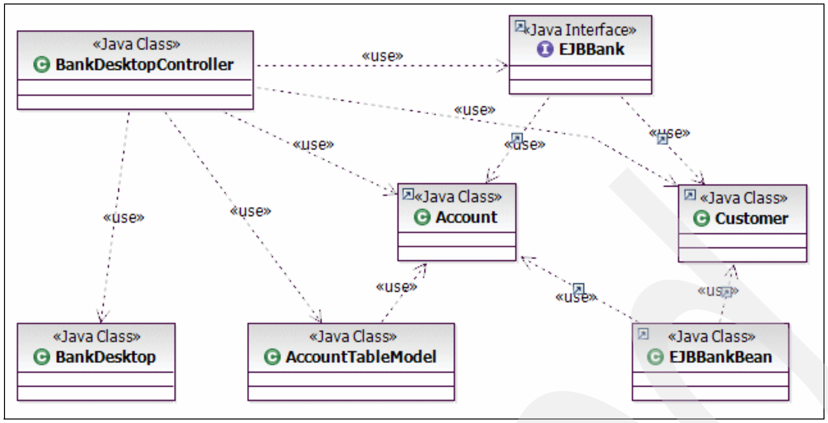


Figure 17-3 Class diagram for the Bank J2EE application client

Preparing for the sample application

Prior to working on the sample for this chapter, we have to set up the database for the sample application, import the EJB projects, and ensure that everything is working.

Importing the base EJB enterprise application sample

To import the base enterprise application sample that we use as a starting point for this chapter, do these steps:

- In the J2EE perspective Project Explorer, select **File** → **Import**.
- In the Import dialog, select **Project Interchange** and click **Next**.
- In the Import Project Interchange Contents dialog, locate the file `c:\7501code\zInterchangeFiles\ejb\RAD7EJB.zip`, click **Select All**, and click **Finish**.

After the Import wizard has completed the import, four projects have been added to the workspace:

- **RAD7EJBEAR**: This is the deployable enterprise application, which functions as a container for the remaining projects. This enterprise application must be executed on an application server.

- **RAD7EJB**: This project contains the EJBs, that makes up the business logic of the ITSO Bank. The `EJBBank` session bean acts as a facade for the EJB application. This project is packaged inside `RAD7EJBEAR` when exported and deployed on an application server.
- **RAD7EJBClient**: This is the client interface for the EJB application. This project is packaged with any application that has to access the EJBs in the `RAD7EJB` project, including the client application that we develop in this chapter.
- **RAD7EJBJava**: This Java project holds the data transfer objects (DTO) that are passed between the session facade and the client applications.

You should see a number of warnings existing in the workspace. If you do not want to see these warnings, do these steps:

- Select **Window Preferences** → **Java** → **Compiler** → **Errors/Warnings** → **Potential programming problems** and change the option for **Serializable class without serialVersionUID** from **Warning** to **Ignore**.
- Expand the **Unnecessary code** section, and change the option for **Local variable is never read** from **Warning** to **Ignore**.
- Expand the **Generic Types** section, and change the option for **Unchecked generic type operation** from **Warning** to **Ignore**. Click **OK**.
- At the resulting Error/Warnings Settings Changed pop-up, click **Yes**.

Setting up the sample database

The entity EJBs are mapped to the `ITSOBANK` relational database implemented in Derby or DB2. Follow the instructions in “Setting up the ITSOBANK database” on page 1312to set up the database.

Configuring the data source

There are a couple of methods that can be used to configure the data source, including using the WebSphere Administrative Console or using the WebSphere Enhanced EAR, which stores the configuration in the deployment descriptor and is deployed with the application:

- Configuring the data source in the server is described in “Configuring the data source in WebSphere Application Server” on page 1313.
- Configuring the data source in the enhanced EAR was covered in “Configure the data source using enhanced EAR” on page 741. The imported project does contain the data source for Derby.

Set up the default CMP data source

Several data sources can be defined for an enterprise application. For the EJB container to be able to determine which data source should be used, we must configure the `RAD7EJB` project to point to the correct data source as follows:

- Open the **Deployment Descriptor** of the **RAD7EJB** project.
- In the Overview tab, scroll down to the JNDI - CMP Connection Factory Binding section.
- Verify the JNDI name field. Either use **jdbc/itsobankejb** (from the enhanced EAR) or **jdbc/itsobank** for the data source of the server.

Testing the imported code

Before continuing with the sample application, we suggest that you test the imported code. Follow the instructions in “Testing EJBs with the Universal Test Client” on page 781, to test the EJBs.

Developing the J2EE application client

We now use Application Developer to create a project containing a J2EE application client. This application client will be associated with its own enterprise application.

> **Note:** While it is possible to use the new client application with the existing `RAD7EJBEAR` enterprise application, this is not the recommended approach. The EJB project contains other server resources that should not be distributed to the clients, such as passwords or proprietary business logic.

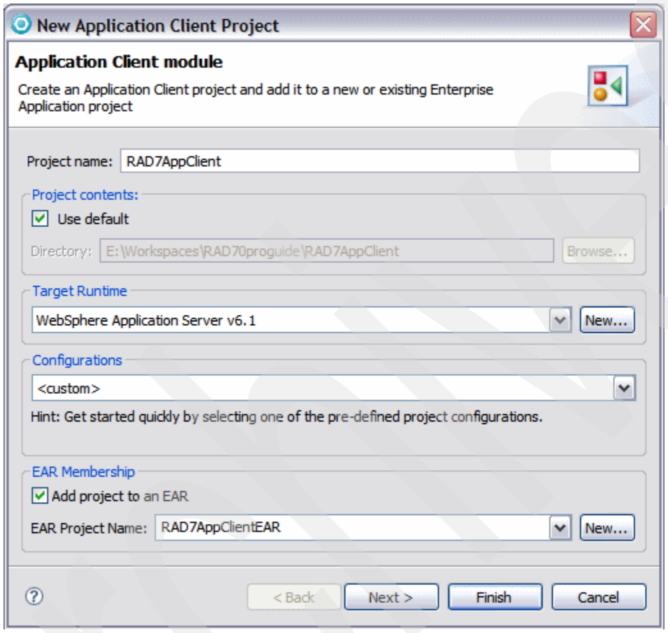
To finish the J2EE application client sample, you need to complete the following tasks:

- Creating the J2EE application client projects
- Configuring the J2EE application client projects
- Importing the graphical user interface and control classes
- Creating the BankDesktopController class
- Completing the BankDesktopController class
- Registering the BankDesktopController class as the main class

Creating the J2EE application client projects

To create a J2EE application client project, do these steps:

- Select **File** → **New** → **Project**.
- Expand the **J2EE** folder, and select **Application Client Project**.
- In the New Application Client Project wizard, enter **RAD7AppClient** as the Project name and **RAD7AppClientEAR** as the EAR Project Name. Click **Next** (Figure 17-4).



New Application Client Project

Application Client module

Create an Application Client project and add it to a new or existing Enterprise Application project

Project name: RAD7AppClient

Project contents:

☑ Use default

Directory: E:\Workspaces\RAD70proguide\RAD7AppClient

Target Runtime: WebSphere Application Server v6.1

Configurations: <custom>

Hint: Get started quickly by selecting one of the pre-defined project configurations.

EAR Membership

☑ Add project to an EAR

EAR Project Name: RAD7AppClientEAR

Figure 17-4 New Application Client Project wizard

- In the Project Facets page, click **Next**.
- In the Application Client module page, clear **Create a default Main class**. Click **Finish**.

When the wizard is complete, the following projects should have been created in your workspace:

- `RAD7AppClientEAR`: This is an enterprise application project that acts as a container for the code to be deployed on the application client node.
- `RAD7AppClient`: This project will contain the actual code for the ITSO Bank application client.

Note: After the wizard has created the new projects, you will see the following error in the Problems view:

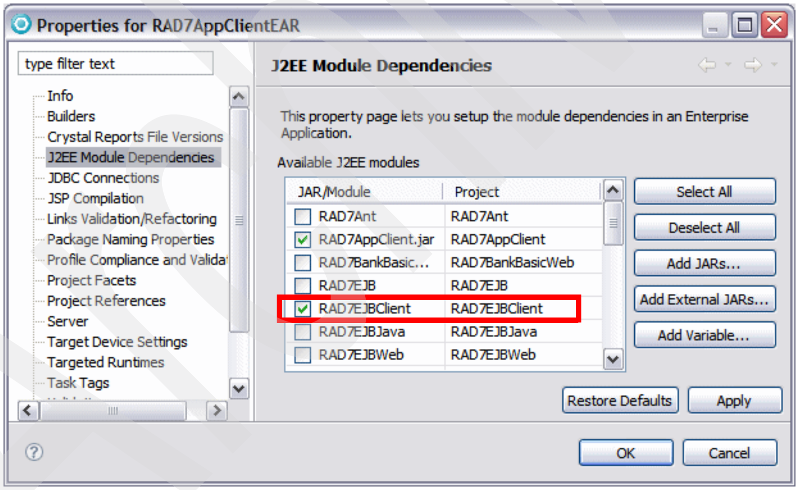
```
IWAE0035E The Main-Class attribute must be defined in the application
client module.
```

This is because we cleared **Create a default Main Class** in the New Application Client Project wizard. We will create a main class, and thus resolve this problem in a subsequent step.

Configuring the J2EE application client projects

The application client project has to reference the `RAD7EJBClient` project. In this section, we configure this dependency by adding the `RAD7EJBClient` as a dependency to both projects:

- In the Project Explorer, right-click **RAD7AppClientEAR** and select **Properties**.
- Select **J2EE Module Dependencies** and select **RAD7EJBClient**, then click **OK** (Figure 17-5).



J2EE Module Dependencies

This property page lets you setup the module dependencies in an Enterprise Application.

Available J2EE modules

JAR/Module	Project
☐ RAD7Ant	RAD7Ant
☑ RAD7AppClient.jar	RAD7AppClient
☐ RAD7BankBasic...	RAD7BankBasicWeb
☐ RAD7EJB	RAD7EJB
☑ RAD7EJBClient	RAD7EJBClient
☐ RAD7EJBJava	RAD7EJBJava
☐ RAD7EJBWeb	RAD7EJBWeb

Figure 17-5 J2EE Module Dependencies

- Right-click **RAD7AppClient**, and select **Properties**.
- Select **J2EE Module Dependencies** and select **RAD7EJBClient**, then click **OK**.

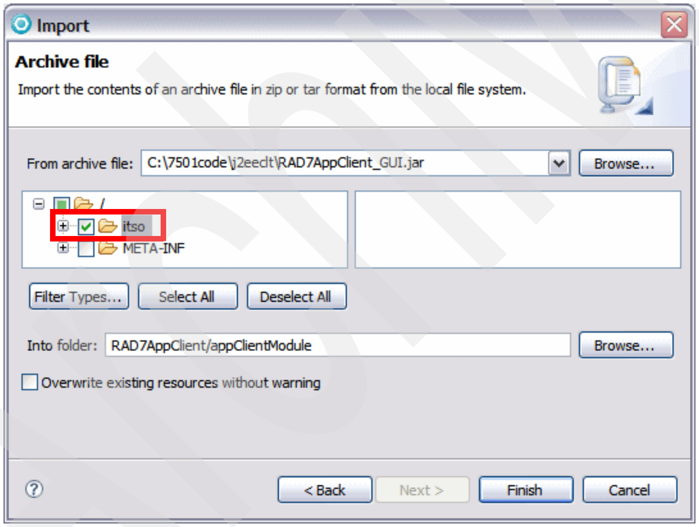
Importing the graphical user interface and control classes

In this section, we complete the graphical user interface (GUI) for the J2EE application client.

Because this chapter focuses on the aspects relating to development of J2EE application clients, we import the finished user interface and focus on implementing the code for accessing the EJBs.

To import the framework classes for the J2EE application client, do these steps:

- In the Project Explorer expand **RAD7AppClient**, right-click **appClientModule**, and select **Import**.
- In the Import dialog, expand **General** → **Archive File** and click **Next**.
- In the Archive file page click **Browse** to locate `c:\7501code\j2eeclient\RAD7AppClient_GUI.jar`.
- Expand **/** in the left panel and ensure that only **itso** is selected, and click **Finish** (Figure 17-6).



From archive file: C:\7501code\j2eeclt\RAD7AppClient_GUI.jar

☑ itso

□ META-INF

Into folder: RAD7AppClient/appClientModule

□ Overwrite existing resources without warning

Figure 17-6 Import existing GUI class

Two classes have been imported to the `RAD7AppClient` project:

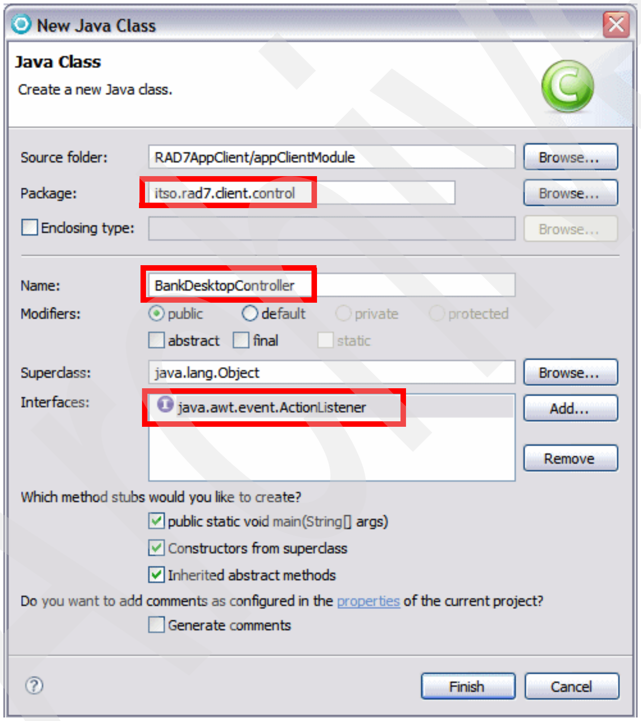
- `itso.rad7.client.ui.BankDesktop`—This is a visual class, containing the view for the Bank J2EE application client.
- `itso.bank.client.model.AccountTableModel`—This is an implementation of the interface `javax.swing.table.TableModel`. The class provides the relevant `TableModel` interface, given an array of `Account` instances.

Creating the BankDesktopController class

In this section, we create the controller class for the J2EE application client. This class is also the main class for the application and contains the EJB lookup code.

To create the `BankDesktopController` class, do these steps:

- In the Project Explorer expand **RAD7AppClient**, right-click **appClientModule** and select **New** → **Class**.
- In the New Java Class dialog (Figure 17-7), enter **`itso.rad7.client.control`** in the Package field, **`BankDesktopController`** in the Name field, select **public static void main(String[] args)** and **Constructors from superclass**.
- Click **Add** next to the Interface section. In the Implemented Interfaces Selection dialog, enter **`ActionListener`** in the Choose interfaces field, select **ActionListener - java.awt.event** in the Matching types list, and click **OK**.



New Java Class

Java Class

Create a new Java class.

Source folder: RAD7AppClient/appClientModule [Browse...]

Package: itso.rad7.client.control [Browse...]

□ Enclosing type: [Browse...]

Name: BankDesktopController

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object [Browse...]

Interfaces: java.awt.event.ActionListener [Add...] [Remove]

Which method stubs would you like to create?

☑ public static void main(String[] args)

☑ Constructors from superclass

☑ Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

□ Generate comments

[Finish] [Cancel]

Figure 17-7 Create class BankDesktopController

Completing the BankDesktopController class

In this section we add control logic to the `BankDesktopController` class, as well as the code to look up customer and account information from the EJB application.

> **Note:** The code found in this section can be copied from the complete `BankDesktopController` class that is supplied in the sample code:
>
> ```
> c:\7501code\j2eeclient\BankDesktopController.java
> ```
>
> We suggest that you copy the sections as noted in our procedure from the completed `BankDesktopController.java` (step by step). If you simply import the class, the `serviceLocatorMgr.jar` is not added as a Java JAR dependency to the `RAD7AppClient` project, and thus you will have *can not resolve* errors in the source code. The `serviceLocatorMgr.jar` is added by using the Insert EJB wizard.

To complete the `BankDesktopController` class, do these steps:

- The **BankDesktopController.java** is open in the Java editor.
- Add the import statements to the Java file:

```
import itso.rad7.client.ui.AccountTableModel;
import itso.rad7.client.ui.BankDesktop;
import itso.rad7.bank.exception.InvalidCustomerException;
import itso.rad7.bank.model.Account;
import itso.rad7.bank.model.Customer;
```

- Add two fields to the beginning of the class definition:

```
private BankDesktop desktop = null;
private AccountTableModel accountTableModel = null;
```

- Locate the **constructor** and modify it (the new code is in bold):

```
public BankDesktopController() {
   desktop = new BankDesktop();
   desktop.getBtnSearch().addActionListener(this);
   desktop.setVisible(true);
}
```

- Locate the **main** method stub and add one line:

```
public static void main(String[] args) {
   BankDesktopController controller = new BankDesktopController();
}
```

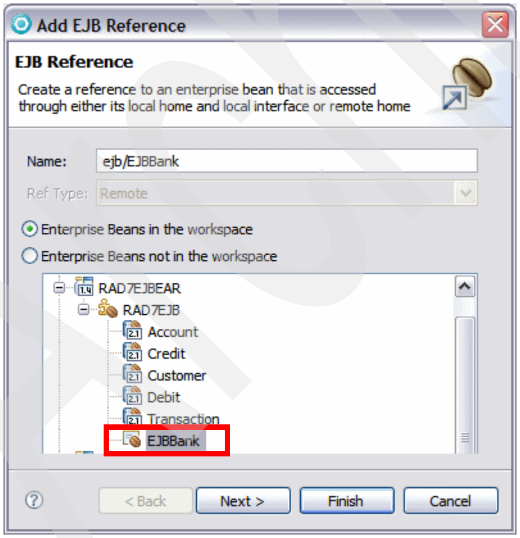
- Add the **setAccounts** method before the main method:

```
    private void setAccounts(Account[] accounts) {
        if (accountTableModel == null) {
            accountTableModel = new AccountTableModel();
            desktop.getTblAccounts().setModel(accountTableModel);
        }
        accountTableModel.setAccounts(accounts);
    }
```

- Locate the **actionPerformed** method stub and add one line:

```
public void actionPerformed(ActionEvent e) {
    String ssn = desktop.getTfSSN().getText();
}
```

- Place the cursor on the last line of the `actionPerformed` method (the line between the ssn variable declaration and the ending curly brace).
- In the Snippets view expand **EJB** and double-click **Call an EJB "create" method**.
- In the Insert EJB create dialog, click **New EJB Reference**.
- In the Add EJB Reference dialog, select **Enterprise Beans in the workspace**, expand **RAD7EJBEAR** → **RAD7EJB** and select **EJBBank**, and click **Finish** (Figure 17-8).



Add EJB Reference

EJB Reference

Create a reference to an enterprise bean that is accessed through either its local home and local interface or remote home

Name: ejb/EJBBank

Ref Type: Remote

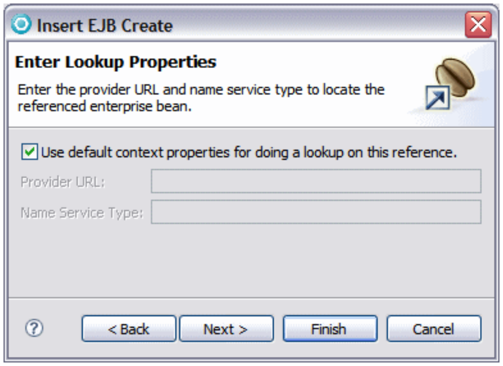
● Enterprise Beans in the workspace

○ Enterprise Beans not in the workspace

Figure 17-8 EJB reference

- When you return to the Insert EJB create dialog, click **Next**.

- In the Enter Lookup Properties dialog, select **Use default context properties for doing a lookup on this reference** and click **Finish** (Figure 17-9).



Insert EJB Create

Enter Lookup Properties

Enter the provider URL and name service type to locate the referenced enterprise bean.

☑ Use default context properties for doing a lookup on this reference.

Provider URL:

Name Service Type:

Figure 17-9 Enter Lookup Properties page

Note: By specifying to use the default context properties, we do not have to hard-code the server name in the code, or in other ways make the code location-aware.

To allow the application client to run on a node, separate from the application server, just specify the server name when starting the J2EE client container.

In the Application Client for WebSphere Application Server, this can be done by using the `-CCBootstrapHost` parameter in the **`launchClient`** command. Refer to the WebSphere Application Server InfoCenter for more information about using the Application Client for WebSphere Application Server.

The Insert EJB Create wizard does the following tasks:

- Adds the following line at the cursor location in the `actionPerformed` method:

```
EJBBank anEJBBank = createEJBBank();
```

- Adds private fields `STATIC_EJBBankHome_REF_NAME` and `STATIC_EJBBankHome_CLASS` to the class.
- Adds a private method, `createEJBBank`, to the class.
- Adds the `serviceLocatorMgr.jar` as a Utility JAR to the `RAD7AppClientEAR` enterprise application project.
- Creates an EJB reference, `ejb/EJBBank`, in the deployment descriptor of the `RAD7AppClient` project. Open the deployment descriptor and verify that the reference has a WebSphere Bindings JNDI name of `ejb/itso/rad7/bank/facade/ejb/EJBBankHome`.

- Adds a number of import statements to the Java class file.

Complete the actionPerformed method as shown in Example 17-1. The new code is highlighted in bold.

Example 17-1 Complete actionPerformed method

```
public void actionPerformed(ActionEvent e) {
   // we know that we are only listening to action events from
   // the search button, so...
   String ssn = desktop.getTfSSN().getText();
   Bank aBank = createBank();
   try {
      // look up the customer
      Customer customer = anEJBBank.getCustomer(ssn);
      // look up the accounts
      Account[] accounts = anEJBBank.getAccounts(ssn);

      // update the user interface
      desktop.getTfTitle().setText(customer.getTitle());
      desktop.getTfFirstName().setText(customer.getFirstName());
      desktop.getTfLastName().setText(customer.getLastName());
      setAccounts(accounts);
   }
   catch (UnknownCustomerException x) {
      // unknown customer. Report this using the output fields...
      desktop.getTfTitle().setText("(not found)");
      desktop.getTfFirstName().setText("(not found)");
      desktop.getTfLastName().setText("(not found)");
      setAccounts(new Account[0]);
   }
   catch (RemoteException x) {
      // unexpected RMI exception. Print it to the console and report it...
      x.printStackTrace();
      desktop.getTfTitle().setText("(internal error)");
      desktop.getTfFirstName().setText("(internal error)");
      desktop.getTfLastName().setText("(internal error)");
      setAccounts(new Account[0]);
   }
}
```

- Save and close the BankDesktopController.

The code for the ITSO Bank J2EE application client is now complete. Now we just need to register the BankDesktopController class as the main class for the application client.

Registering the BankDesktopController class as the main class

The `BankDesktopController` class contains the logic for the J2EE application client. We have to register that this is the main class for the application client, such that J2EE application client containers know how to launch the application:

- In the Project Explorer, right-click the **RAD7AppClient** → **Deployment Descriptor** and select **Open With** → **JAR Dependency Editor**.
- In the JAR Dependency editor, click **Browse** next to the Main-Class entry field (at the bottom).
- In the Type Selection dialog, enter **`BankDesk`** for the Select a class using field, then select the **BankDesktopController** in the Matching types list, and click **OK**.
- Save and close the JAR Dependency editor.

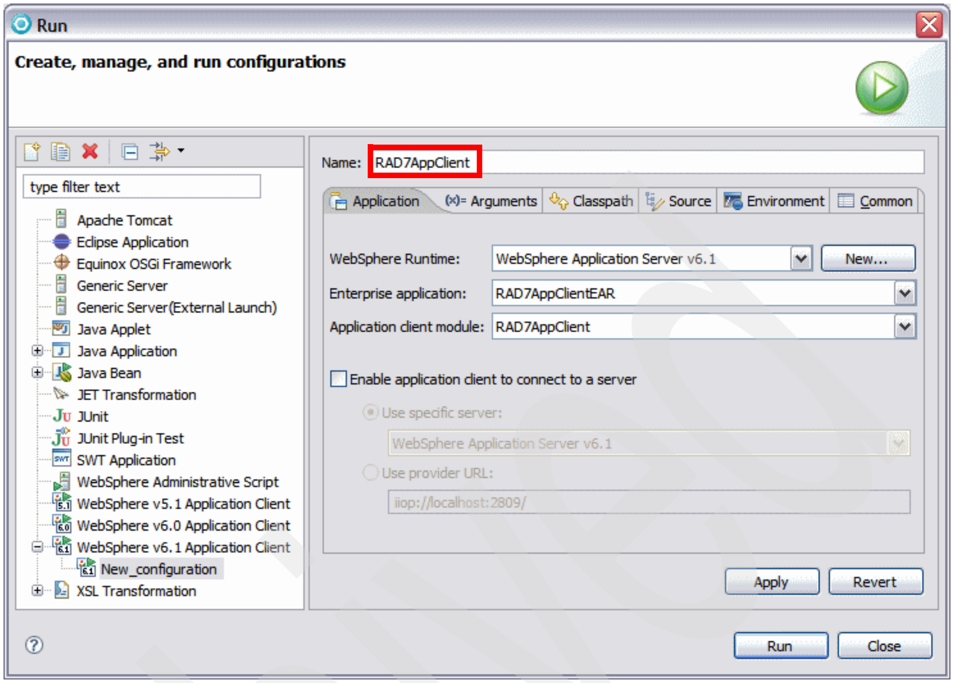
The error regarding the missing main class disappears from the Problems view when the file is saved.

Tip: If you get errors regarding classes not found (`Account`), rebuild each project (select the project and **Project** → **Build Project**).

Testing the J2EE application client

Now that the code has been updated, we can test the J2EE application client as follows:

- Make sure that the server is started.
- Ensure that the `RAD7EJBEAR` enterprise application is deployed on the server:
  - In the **Servers** vie, right-click **WebSphere Application Server v6.1** and select **Add and remove projects**. Add the `RAD7EJBEAR` project if it is not deployed to the server already.
  - Do not try to add the `RAD7AppClientEAR` to the server. We run the application client outside of the server.
- In the Project Explorer, right-click **RAD7AppClient** and select **Run As** → **Run**.
- In the Run dialog, double-click **WebSphere v6.1 Application Client** in the left pane. A `New_configuration` is added and displayed in the right pane.
- Enter **`RAD7AppClient`** in the Name field, click **Apply**, and then click **Run** (Figure 17-10).


Run

Create, manage, and run configurations

Name: RAD7AppClient

WebSphere Runtime: WebSphere Application Server v6.1

Enterprise application: RAD7AppClientEAR

Application client module: RAD7AppClient

☐ Enable application client to connect to a server

● Use specific server:

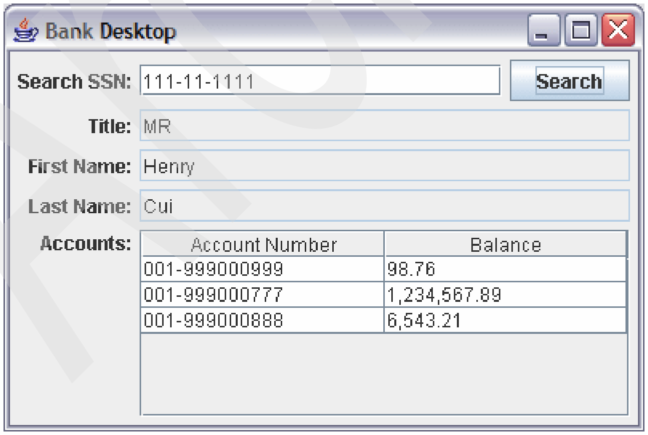
WebSphere Application Server v6.1

○ Use provider URL:

iiop://localhost:2809/

Figure 17-10 Create a new run configuration for the application client

- A number of messages are displayed in the Console.
- The Bank Desktop window opens. Enter `111-11-1111` in the Search SSN field and click **Search**. The result is displayed in Figure 17-11.



Search SSN: 111-11-1111

Title: MR

First Name: Henry

Last Name: Cui

Accounts:

Account Number	Balance
001-999000999	98.76
001-999000777	1,234,567.89
001-999000888	6,543.21

Figure 17-11 Running the application client

- Enter an invalid value for the Search SSN and observe the output in the GUI.
- When you have finished testing the J2EE application client, close the window.

We have now successfully built and tested a J2EE application client.

Packaging the J2EE application client

To run the application client outside Application Developer, we have to package the application.

> **Note:** Although the J2EE specification names the JAR format as the principle means for distributing J2EE application clients, the WebSphere Application Server application client container expects an Enterprise Application Archive (EAR) file.

To package the application client for deployment, do these steps:

- In the Project Explorer, right-click **RAD7AppClientEAR** and select **Export** → **EAR file**.
- In the EAR Export dialog, enter the location and the name of the EAR file (for example, `c:\7501code\deployment\RAD7AppClientEAR.ear`) in the Destination field, and click **Finish**.

The exported EAR file can now be deployed to a client node and executed using the Application Client for WebSphere Application Server.

Running the deployed application client

You can use the **`launchClient`** command to run the application client outside of Application Developer. Do these steps:

- Open a command window at `<RAD_HOME>\runtimes\base_v61\bin`.
- Execute the command:

```
launchclient c:\7501code\deployment\RAD7AppClientEAR.ear
```

- The Bank Desktop window opens and you can run the application client.

18

Develop Web services applications

This chapter introduces the concepts of a service-oriented architecture (SOA) and explains how such an architecture can be realized using the Java 2 Platform Enterprise Edition (J2EE) Web services implementation.

We explore the features provided by Application Developer for Web services development and look at two Web services development approaches: top-down and bottom-up. We also demonstrate how Application Developer can help with testing Web services and developing Web services client applications.

The chapter is organized into the following sections:

- Introduction to Web services and tooling in Application Developer
- Creating Web services from a JavaBean
- Creating Web services JSP and JSF clients
- Creating Web services from an EJB
- Creating a top-down Web service from a WSDL and using an Ant task
- Creating SDO facades for Web services
- Web services security
- Developing Web services with the Web Services Feature Pack

Note: For more detailed information, refer to the Redbooks publication, *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257.

Introduction to Web services

This section introduces architecture and concepts of the service-oriented architecture (SOA) and Web services.

Service-oriented architecture (SOA)

In a service-oriented architecture, applications are made up from loosely coupled software services, which interact to provide all the functionality needed by the application. Each service is generally designed to be very self-contained and stateless to simplify the communication that takes place between them.

There are three main roles involved in a service-oriented architecture:

- Service provider
- Service broker
- Service requester

The interactions between these roles are shown in Figure 18-1.

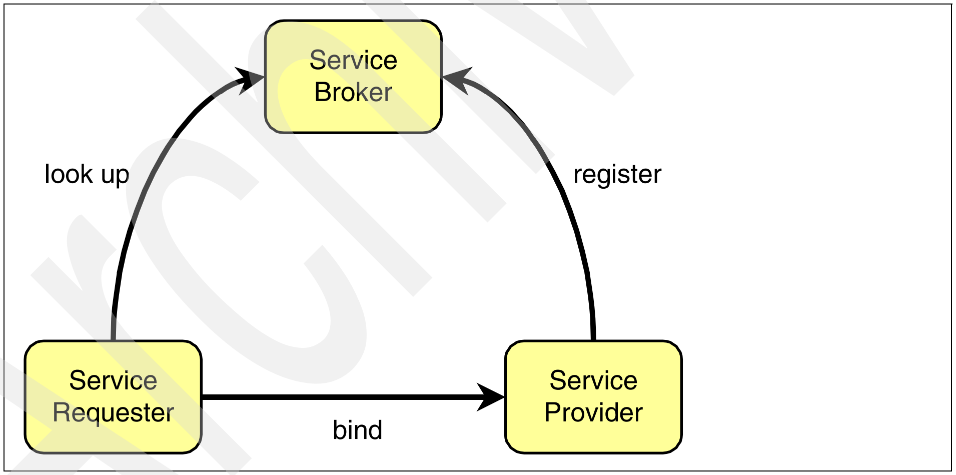


Figure 18-1 Service-oriented architecture

Service provider

The *service provider* creates a service and can publish its interface and access information to a *service broker*.

A service provider must decide which services to expose and how to expose them. There is often a trade-off between security and interoperability; the service provider must make technology decisions based on this trade-off. If the service provider is using a service broker, decisions must be made on how to categorize the service, and the service must be registered with the service broker using agreed-upon protocols.

Service broker

The *service broker,* also known as the *service registry*, is responsible for making the service interface and implementation access information available to any potential service requester.

The service broker will provide mechanisms for registering and finding services. A particular broker might be public (for example, available on the Internet) or private—only available to a limited audience (for example, on an intranet). The type and format of the information stored by a broker and the access mechanisms used will be implementation-dependent.

Service requester

The *service requester*, also know as a *service client*, discovers services and then uses them as part of its operation.

A service requester uses services provided by service providers. Using an agreed-upon protocol, the requester can find the required information about services using a broker (or this information can be obtained in some other way). Once the service requester has the necessary details of the service, it can bind or connect to the service and invoke operations on it. The binding is usually static, but the possibility of dynamically discovering the service details from a service broker and configuring the client accordingly makes dynamic binding possible.

Web services as an SOA implementation

Web services provides a technology foundation for implementing a service-oriented architecture. A major focus during the development of this technology is to make the functional building blocks accessible over standard Internet protocols which are independent of platforms and programming languages to ensure that very high levels of interoperability are possible.

Web services are self-contained software services that can be accessed using simple protocols over a network. They can also be described using standard mechanisms, and these descriptions can be published and located using standard registries. Web services can perform a wide variety of tasks, ranging from simple request-reply to full business process interactions.

Using tools like Application Developer, existing resources can be exposed as Web services very easily.

The following core technologies are used for Web services:

- XML
- SOAP
- WSDL

XML

Extensible Markup Language (XML) is the markup language that underlies Web services. XML is a generic language that can be used to describe any kind of content in a structured way, separated from its presentation to a specific device. All elements of Web services use XML extensively, including XML namespaces and XML schemas.

The specification for XML is available at:

http://www.w3.org/XML/

SOAP

Simple Object Access Protocol (SOAP) is a network, transport, and programming language neutral protocol that allows a client to call a remote service. The message format is XML. SOAP is used for all communication between the service requester and the service provider. The format of the individual SOAP messages depends on the specific details of the service being used.

The specification for SOAP is available at:

http://www.w3.org/TR/soap/

WSDL

Web Services Description Language (WSDL) is an XML-based interface and implementation description language. The service provider uses a WSDL document in order to specify:

- The operations a Web service provides
- The parameters and data types of these operations
- The service access information

WSDL is one way to make service interface and implementation information available in a UDDI registry. A server can use a WSDL document to deploy a Web Service. A service requester can use a WSDL document to work out how to access a Web Service (or a tool can be used for this purpose).

The specification for WSDL is available at:

http://www.w3.org/TR/wsdl/

Related Web services standards

The basic technologies of XML, SOAP, and WSDL are fundamental to Web services, but many other standards have been developed to help with developing and using them.

An excellent resource for information on standards related to Web services can be found at:

http://www.ibm.com/developerworks/views/webservices/standards.jsp

Web services in J2EE V1.4

One of the main changes in moving from J2EE V1.3 to V1.4 is the incorporation of Web services into the platform standard. J2EE V1.4 provides support for Web services clients and also allows Web services to be published. The main technologies in J2EE V1.4 that provide this support are as follows:

- Java API for XML-based Remote Procedure Calls (JAX-RPC): JAX-RPC provides an API for Web services clients to invoke services using SOAP over HTTP. It also defines standard mappings between Java classes and XML types.
- SOAP with Attachments API for Java (SAAJ): Allows SOAP messages to be manipulated from within Java code. The API includes classes to represent such concepts as SOAP envelopes (the basic packaging mechanism within SOAP), SOAP faults (the SOAP equivalent of Java exceptions), SOAP connections, and attachments to SOAP messages.
- Web Services for J2EE: This specification deals with the deployment of Web Service clients and Web services themselves. Under this specification, Web services can be implemented using JavaBeans or stateless session EJBs.
- Java API for XML Registries (JAXR): This API deals with accessing XML registry servers, such as servers providing UDDI functionality.

The specifications for Web services support in J2EE V1.4 are available at:

http://java.sun.com/j2ee/

Web services interoperability

In an effort to improve the interoperability of Web services, the Web Services Interoperability Organization (known as WS-I) was formed. WS-I produces a specification known as the *WS-I Basic Profile*, which describes the technology choices that maximize interoperability between Web services and clients running on different platforms, using different runtime systems, and written in different languages.

The WS-I Basic Profile is available at:

```
http://ws-i.org/deliverables/workinggroup.aspx?wg=basicprofile
```

Web services security

The WS-Security specification describes extensions to SOAP that allow for quality of protection of SOAP messages. This includes, but is not limited to, message authentication, message integrity, and message confidentiality. The specified mechanisms can be used to accommodate a wide variety of security models and encryption technologies. It also provides a general-purpose mechanism for associating security tokens with message content.

The WS-Security 1.0 standard was published in March 2004. The standard also includes the UsernameToken profile and X.509 Certificate Token profile.

Additional token profiles for REL and SAML are currently published as Committee Drafts. For additional information, refer to:

```
http://www.oasis-open.org/specs/index.php#wssv1.0
http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wss
```

Web services development approaches

There are two general approaches to Web service development: top-down and bottom-up.

In the top-down approach, a Web service is based on the Web service interface and XML types, defined in Web Services Description Language (WSDL) and XML Schema Definition (XSD) files. The developer first designs the implementation of the Web service by creating a WSDL file using the WSDL editor. The developer can then use the Web services wizard to create the Web service and skeleton Java classes to which the developer can add the required code.The developer then modifies the skeleton implementation to interface with the business logic.

In the bottom-up approach, a Web service is created based on the existing business logic in Java beans or EJBs. A WSDL file is generated to describe the resulting Web service interface.

The bottom-up pattern is often used for exposing existing function as a Web service. It might be faster, and no XSD or WSDL design skills are needed. However, if complex objects (for example, Java collection types) are used, then the resulting WSDL might be hard to understand and less interoperable.

The top-down approach allows for more control over the Web service interface and the XML types used, and is the recommended approach for developing new Web services.

Web services tools in Application Developer

Application Developer provides tools to create Web services from existing Java and other resources or from WSDL files, as well as tools for Web services client development and for testing Web services.

Creating a Web service from existing resources

Application Developer provides wizards for exposing a variety of resources as Web services. The following resources can be used to build a Web Service:

- **JavaBean**: The Web Service wizard assists you in creating a new Web Service from a simple Java class, configures it for deployment, and deploys the Web Service to a server. The server can be the WebSphere Application Server V6.1 Test Environment included with Rational Application Developer or another application server.
- **EJB**: The Web Service wizard assists you in creating a new Web Service from a stateless session EJB, configures it for deployment, and deploys the Web Service to a server.
- **DADX**: Document access definition extension (DADX) is an XML document format that specifies how to create a Web Service using a set of operations that are defined by DAD documents and SQL statements. A DADX Web Service enables you to expose DB2 XML Extender or regular SQL statements as a Web Service. The DADX file defines the operations available to the DADX run-time environment and the input and output parameters for the SQL operation.

Creating a skeleton Web service

Application Developer provides the functionality to create Web services from a description in a WSDL (or WSIL) file:

- **JavaBean from WSDL**: The Web Service wizard assists you in creating a skeleton JavaBean from an existing WSDL document. The skeleton bean contains a set of methods that correspond to the operations described in the WSDL document. When the bean is created, each method has a trivial implementation that you replace by editing the bean.

- **Enterprise JavaBean from WSDL**: The Web services tools support the generation of a skeleton EJB from an existing WSDL file. Apart from the type of component produced, the process is similar to that for JavaBeans.

Client development

To assist in development of Web service clients, Application Developer provides these features:

- **Java client proxy from WSDL**: The Web Service client wizard assists you in generating a proxy JavaBean. This proxy can be used within a client application to greatly simplify the client programming required to access a Web Service.
- **Sample Web application from WSDL**: Rational Application Developer can generate a sample Web application, which includes the proxy classes described above, and sample JSPs that use the proxy classes.
- **Web Service Discovery Dialog**: This dialog allows you to discover a Web service that exists online or in your workspace, create a proxy to the Web service, and then place the methods of the proxy on a Faces JSP file.

Testing tools for Web services

To allow developers to test Web services, Application Developer provides a range of features:

- **WebSphere Application Server** V6.1, V6.0 and V5.1Test Environment: These Test Environments are included with Rational Application Developer as a test server and can be used to host Web services. It provides a range of Web services runtimes, including an implementation of the J2EE specification standards.
- **Sample JSP application**: The Web application mentioned above can be used to test Web services and the generated proxy it uses.
- **Web Services Explorer**: This is a simple test environment that can be used to test any Web Service, based only on the WSDL file for the service. The service can be running on a local test server or anywhere else on the network.
- **Universal Test Client**: The Universal Test Client (UTC) is a very powerful and flexible test application that is normally used for testing EJBs. Its flexibility makes it possible to test ordinary Java classes, so it can be used to test the generated proxy classes created to simplify client development.
- **TCP/IP Monitor**: The TCP/IP Monitor works like a proxy server, passing TCP/IP requests on to another server and directing the returned responses

back to the originating client. In the process of doing this, it records the TCP/IP messages that are exchanged, and can display these in a special view within Rational Application Developer.

Preparing for the samples

To prepare for this sample, we import some sample code. This is a simple Web application that includes Java classes and an Enterprise JavaBean.

Import the sample

This section describes the steps required for preparing the environment for the Web services application samples:

- Switch to the J2EE perspective Project Explorer view.
- Select **File** → **Import** → **Other** → **Project Interchange**.
- In the Import Projects dialog, click **Browse** to navigate to and select the **RAD7WebServicesStart.zip** from the `c:\7501code\webservices` folder, and click **Open**.
- Click **Select All** and click **Finish**.
- Notice that there are some warnings in the workspace. If you do not want to see these warnings, select **Window** → **Preferences** → **Java** → **Compiler** → **Errors/Warnings** → expand **Potential programming problems** and change the option for **Serializable class without serialVersionUID** from **Warning** to **Ignore**.
- Expand the **Generic Types** section, and change the option for **Unchecked generic type operation** from **Warning** to **Ignore**. Click **OK**.
- At the resulting Error/Warnings Settings Changed pop-up, click **Yes**.

After the build, all the warnings in the workspace should disappear.

Sample projects

The sample application that we use for creating Web service consists of the following projects:

- **RAD7WebServiceUtility** project: Simple banking model with `BankMemory`, `Customer`, and `Account` beans. This is a simplified version of the `RAD7Java` project used in Chapter 7, “Develop Java applications” on page 227.
- **RAD7WebServiceWeb** project: Contains the `SimpleBankBean`, a JavaBean with a few methods that retrieve data from the `MemoryBank`, a search HTML page, and a result JSP.

- **RAD7WebServiceEJB** project: Contains the `SimpleBankFacade` session EJB with a few methods that retrieve data from the `MemoryBank`.
- **RAD7WebServiceEAR** project: Enterprise application that contains the other three projects.

Test the application

To start and test the basic application, do these steps:

- In the Servers view, right-click the server (for example, WebSphere Application Server v6.1), and select **Add and remove projects**.
- In the Add and Remove Projects dialog, select **RAD7WebServiceEAR** under and click **Add**.
- The **RAD7WebServiceEAR** is now under the Configured projects column. Click **Finish**.
- Expand **RAD7WebServiceWeb** → **WebContent**, right-click **search.html**, and select **Run As** → **Run on Server**.
- Select **Choose an existing server** and the V6.1 server to run the application and then click **Finish**.
- The search page opens in a Web browser. Enter an appropriate value in the Social Security Number field, for example, 111-11-1111, and click **Search**. If everything is working correctly, you should see customer's full name and customer's first account, which have been read from the hash table.
- The stateless session EJB, `SimpleBankFacade`, can be tested using the Universal Test Client (UTC). See "Testing EJBs with the Universal Test Client" on page 781, for more information on using the UTC. The methods are:
  - `getCustomerFullName(ssn)`—Retrieves the full name (use 111-11-1111)
  - `getNumAccounts(ssn)`—Retrieves the number of accounts
  - `getAccountId(ssn, int)`—Retrieves the account ID by index (0,1,2,...)
  - `getAccountBalance(accountId)`—Retrieves the balance

We now have some resources in preparation for the Web services sample, including a JavaBean in the `RAD7WebServiceWeb` project and a session EJB in the `RAD7WebServiceEJB` project. We use these as a base for developing and testing the Web services examples.

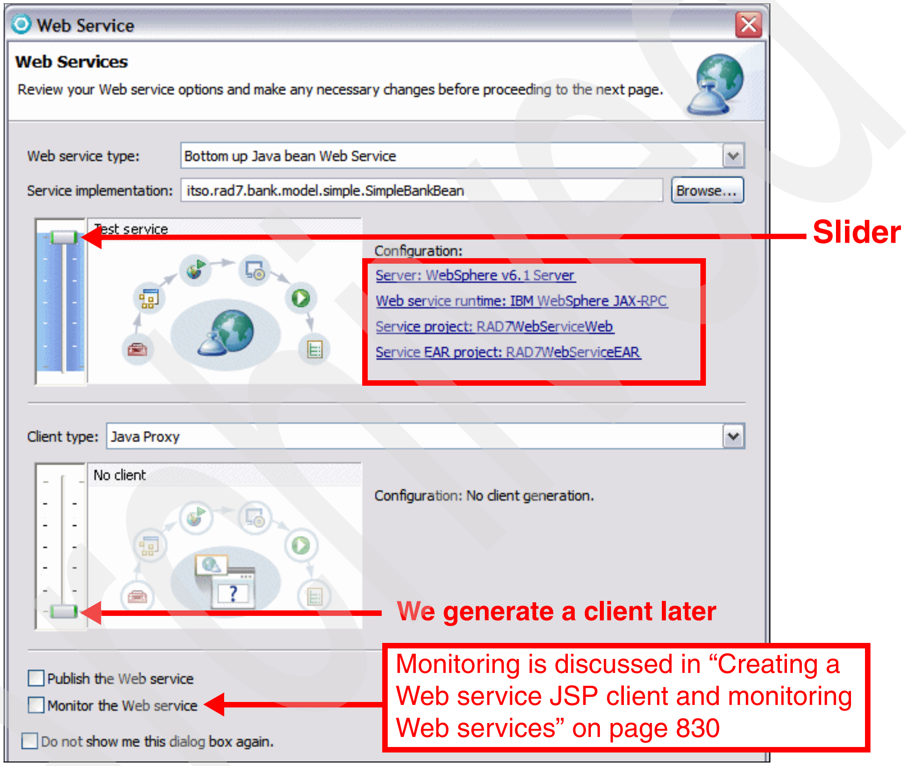
Creating Web services from a JavaBean

In the first example, we create a Web service from an existing Java class. The imported application contains a Java class called `SimpleBankBean`, which has various methods to get customer and account information from the bank.

Creating Web services using the Web Service wizard

To create a Web service from a JavaBean, do these steps:

- Start the Web Service wizard:
  - In the J2EE/Web perspective expand **RAD7WebServiceWeb** → **Java Resources: src** → **itso.rad7.bank.model.simple**.
  - Right-click **SimpleBankBean.java** and select **Web services** → **Create Web service**. The Web Service wizard starts (Figure 18-2).



Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Web service type: Bottom up Java bean Web Service

Service implementation: itso.rad7.bank.model.simple.SimpleBankBean

Configuration:
Server: WebSphere v6.1 Server
Web service runtime: IBM WebSphere JAX-RPC
Service project: RAD7WebServiceWeb
Service EAR project: RAD7WebServiceEAR

Client type: Java Proxy

Configuration: No client generation.

□ Publish the Web service
□ Monitor the Web service
□ Do not show me this dialog box again.

Slider

We generate a client later

Monitoring is discussed in "Creating a Web service JSP client and monitoring Web services" on page 830

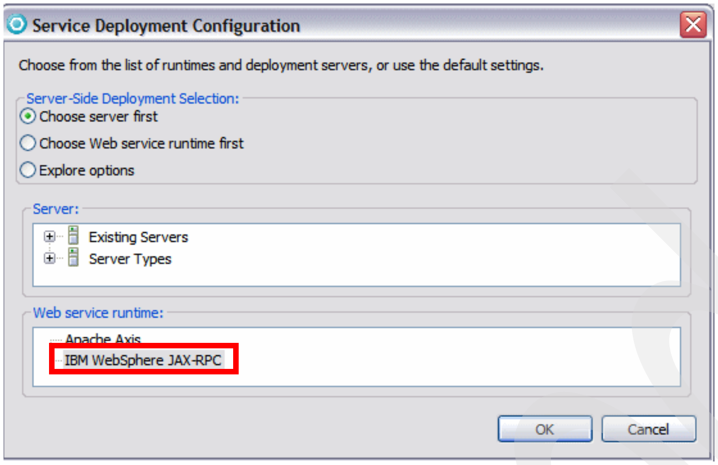
Figure 18-2 Web Service wizard: Web Services page

> **Tip:** Alternatively, you can start the Web Services wizard using **File** → **New** → **Other** → **Web Services** → **Web Service**.

- Select the Web Services options in the Web Services page:
  - Select **Bottom up Java bean Web service** as your Web service type. This should be selected by default.
  - Move the slider for the service to the **Test** position (top). This provides options for testing the service in subsequent pages of the wizard.

> **Behind the scenes:**
>
> The slider allows you to select the stages of Web services development. It allows more granular division of Web services development:
>
> - **Develop**: Develops the WSDL definition and implementation of the Web service. This includes such tasks as creating the modules which will contain the generated code, WSDL files, deployment descriptors, and Java files when appropriate.
> - **Assemble**: Ensures that the project which will host the Web service or client will get associated to an EAR when required by the target application server.
> - **Deploy**: Creates the deployment code for the service.
> - **Install**: Installs and configures the Web module and EARs on the target server. If any changes to the endpoints of the WSDL file are required, they are made in this stage.
> - **Start**: Starts the Web service once the service has been installed on the server.
> - **Test**: Provides various options for testing the service, such as using the Web Service Explorer or sample JSPs.
>
> In Application Developer V6, the user has to create a server and start the server in order to generate Web services code. In Application Developer v7, the server does not have to be created if you move the slider to **Deploy** or below. If the user does not have WebSphere Application Server 6.0 or 6.1 installed, the wizard uses the Web services emitters in the `base_v6_stub` or `base_61_stub` folder to generate the code.

  - Ensure that the following server-side configurations are selected:
    - Server: WebSphere v6.1 Server
    - Web service runtime: IBM WebSphere JAX-RPC
    - Service project: `RAD7WebServiceWeb`
    - Service EAR project: `RAD7WebServiceEAR`
  - If you click the hyperlink **Server: WebSphere v6.1 Server**, the Service Deployment Configuration dialog is displayed (Figure 18-3).



Service Deployment Configuration

Choose from the list of runtimes and deployment servers, or use the default settings.

Server-Side Deployment Selection:

- ● Choose server first
- ○ Choose Web service runtime first
- ○ Explore options

Server:

- Existing Servers
- Server Types

Web service runtime:

- Apache Axis
- IBM WebSphere JAX-RPC

Figure 18-3 Web Service wizard: Service Deployment Configuration

– This page allows you to select the server and runtime. We will leave this page as default and click **Cancel** to exit this page.

Question: There are two Web service runtimes available in Application Developer: Apache Axis and IBM WebSphere JAX-RPC. Which one is recommended for use by IBM?

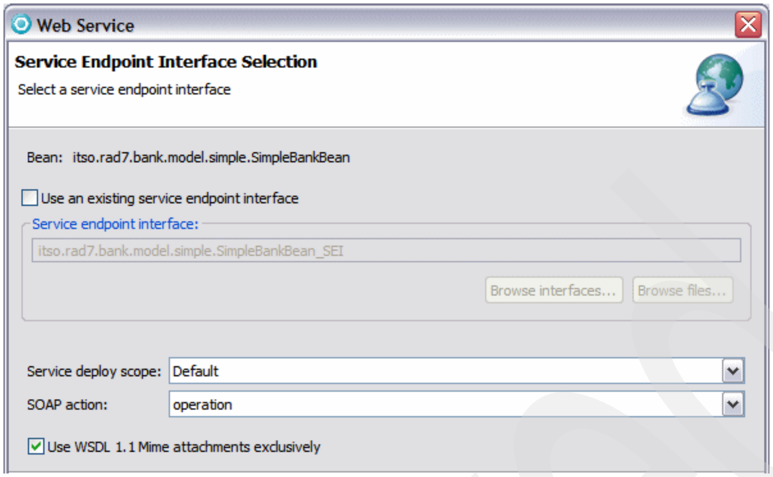
Answer:

- The recommended Web service runtime to use within Application Developer is the WebSphere JAX-RPC Web services runtime.
- Axis runtime shipped with Application Developer v7 (Axis v1.3) is a J2SE SOAP engine and it supports the JAX-RPC standard. It does not require a J2EE container and does not support the JSR 109.
- The WebSphere JAX-RPC Web service runtime leverages the work that IBM contributed to the Apache Axis code base. It is derived from Apache Axis, but has diverged and contains many enhancements, such as improved performance, WS-Security, multi-protocol support, J2EE compliance, in-process optimization, compression, smart parsing, and other enhancements.

– Clear **Publish the Web service** (we do not publish to a UDDI registry) and clear **Monitor the Web service** (we will do that later).

– Click **Next** in the Web Services page.

- In the Service Endpoint Interface Selection dialog, accept the default settings and click **Next** (Figure 18-4).



Web Service

Service Endpoint Interface Selection

Select a service endpoint interface

Bean: itso.rad7.bank.model.simple.SimpleBankBean

☐ Use an existing service endpoint interface

Service endpoint interface: itso.rad7.bank.model.simple.SimpleBankBean_SEI

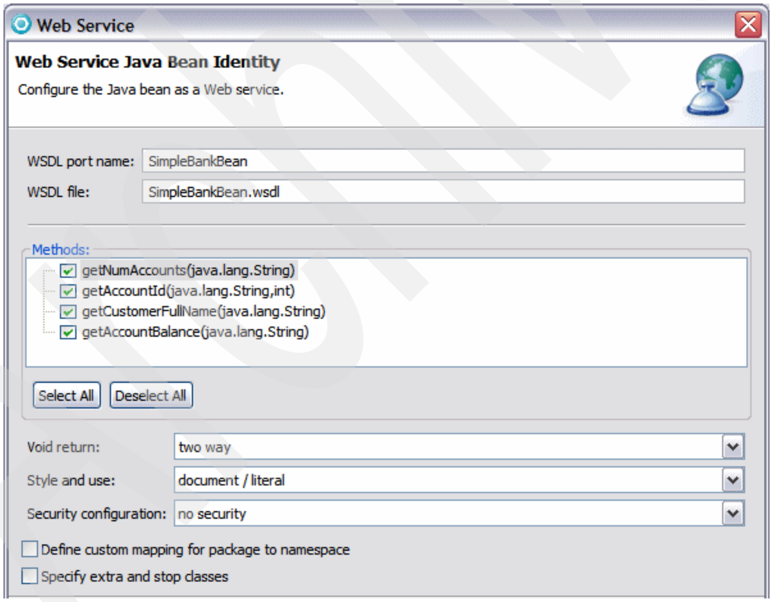
Service deploy scope: Default

SOAP action: operation

☑ Use WSDL 1.1 Mime attachments exclusively

Figure 18-4 Web Service wizard: Service Endpoint Interface Selection

- In the Web Service dialog, accept the default options and click **Next** (Figure 18-5).



Web Service

Web Service Java Bean Identity

Configure the Java bean as a Web service.

WSDL port name: SimpleBankBean

WSDL file: SimpleBankBean.wsdl

Methods:

☑ getNumAccounts(java.lang.String)
☑ getAccountId(java.lang.String,int)
☑ getCustomerFullName(java.lang.String)
☑ getAccountBalance(java.lang.String)

Void return: two way

Style and use: document / literal

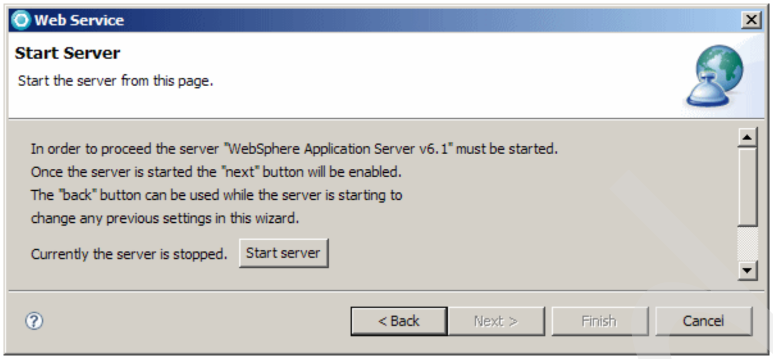
Security configuration: no security

☐ Define custom mapping for package to namespace

☐ Specify extra and stop classes

Figure 18-5 Web Service wizard: Web Services Java Bean Identity

- If you do not start the server before you run the wizard, you are directed to the dialog shown in Figure 18-6. Click **Start Server**. It takes a while to start the server. After the server is started, click **Next**.



Web Service

Start Server

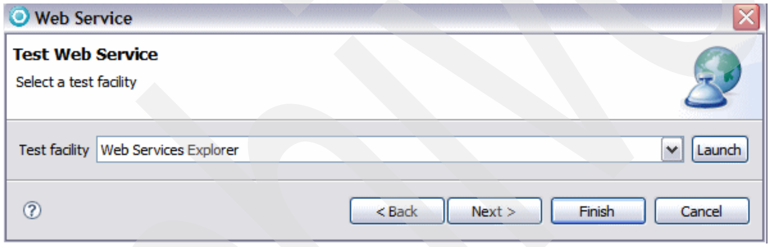
Start the server from this page.

In order to proceed the server "WebSphere Application Server v6.1" must be started.
Once the server is started the "next" button will be enabled.
The "back" button can be used while the server is starting to
change any previous settings in this wizard.

Currently the server is stopped.

Figure 18-6 Web Service wizard: Start server dialog

- The Test Web Services dialog (Figure 18-8) shows because we moved the slider for the service to the Test position. Click **Launch** to launch the Web Services Explorer.



Web Service

Test Web Service

Select a test facility

Test facility: Web Services Explorer

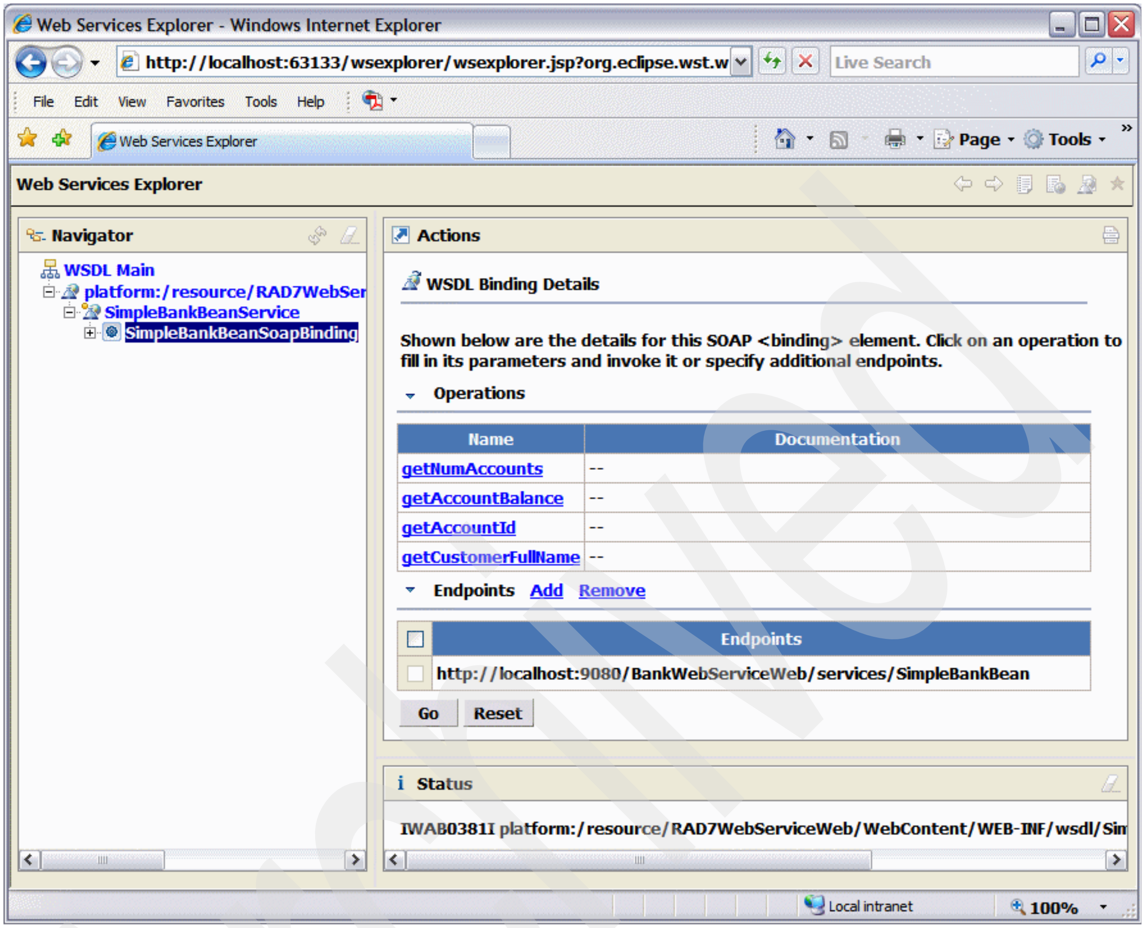
Figure 18-7 Web Service wizard: Test Web Service

- The Web Services Explorer opens in an external Web browser (Figure 18-8).

> **Behind the scenes:** The Web Services Explorer is a JSP Web application hosted on the Apache Tomcat servlet engine contained within Eclipse. The Web Services Explorer uses the WSDL to render a SOAP request. It does not involve data marshalling and unmarshalling. The return parameter is stripped out and the values are displayed in a pre-defined format.
>
> **Note**: The Web Services Explorer can also be started by selecting a WSDL file in Project Explorer and **Web Services** → **Test with Web Services Explorer**. In this case, an internal browser in the Workbench is opened.

  – Notice the endpoint of the Web service:

```
http://localhost:9080/BankWebServiceWeb/services/SimpleBankBean
```


Web Services Explorer - Windows Internet Explorer
File Edit View Favorites Tools Help
Live Search
Web Services Explorer
Page
Tools
Navigator
Actions
Local intranet
100%

http://localhost:63133/wsexplorer/wsexplorer.jsp?org.eclipse.wst.w

Web Services Explorer

- WSDL Main
  - platform:/resource/RAD7WebSer
    - SimpleBankBeanService
      - SimpleBankBeanSoapBinding

WSDL Binding Details

Shown below are the details for this SOAP <binding> element. Click on an operation to fill in its parameters and invoke it or specify additional endpoints.

Operations

Name	Documentation
getNumAccounts	--
getAccountBalance	--
getAccountId	--
getCustomerFullName	--

Endpoints Add Remove

□	Endpoints
□	http://localhost:9080/BankWebServiceWeb/services/SimpleBankBean

Go Reset

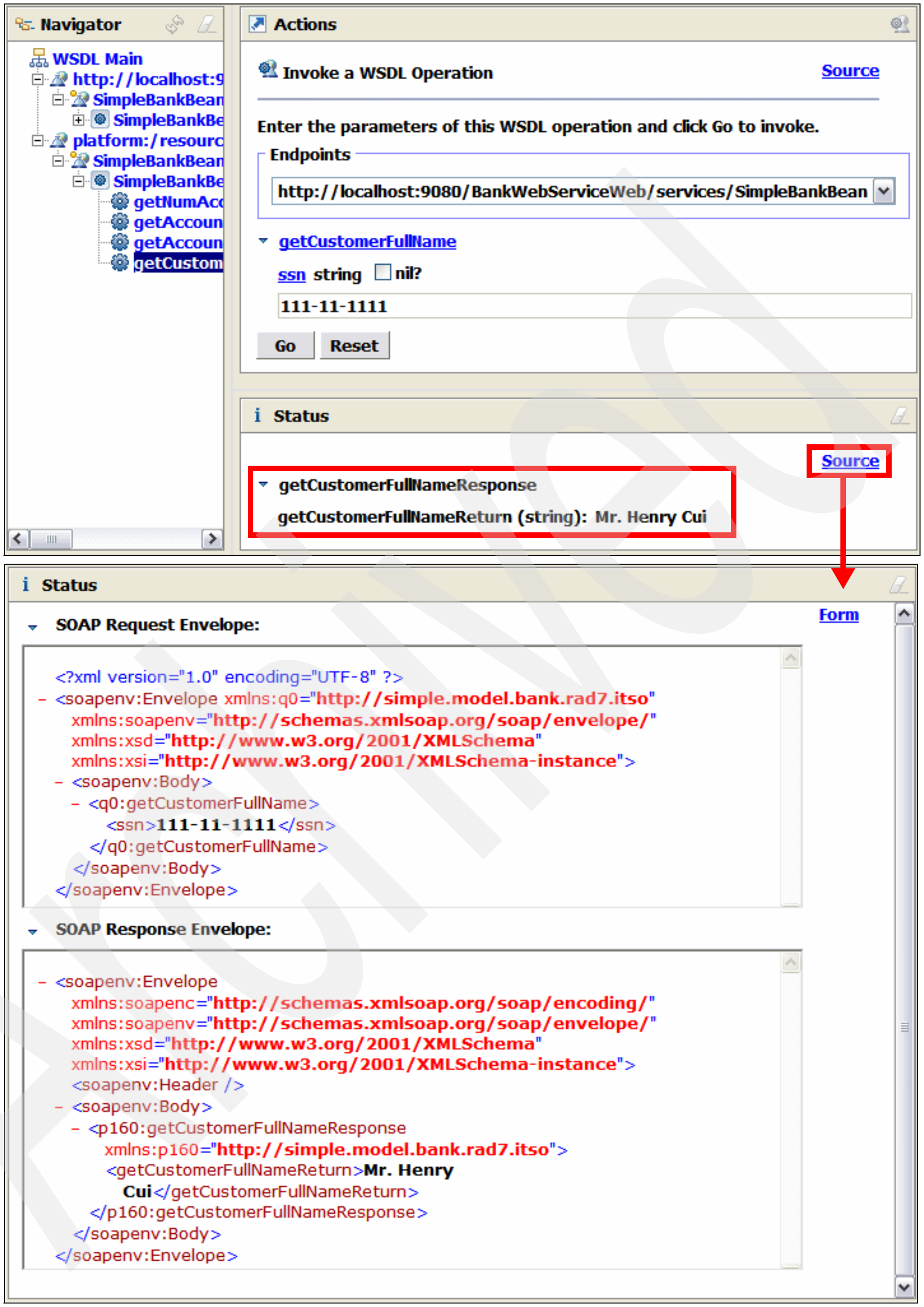
Status

IWAB0381I platform:/resource/RAD7WebServiceWeb/WebContent/WEB-INF/wsdl/Sim

Figure 18-8 Web Services Explorer

- – Select one of the methods (for example, `getCustomerFullName`).
- – The Actions pane on the right displays a simple interface allowing you to enter values for the method parameters. Enter a value for the customer ID, such as 111-11-1111, and click **Go**.
- – The results of the Web service invocation appear in the Status pane at the bottom right of the Web Services Explorer (Figure 18-9).
- – Double-click the **Status** pane bar to maximize it. Then click **Source** to view the SOAP messages as a raw XML.

- ► Close the Web Services Explorer.



Invoke a WSDL Operation

Enter the parameters of this WSDL operation and click Go to invoke.

Endpoints

http://localhost:9080/BankWebServiceWeb/services/SimpleBankBean

getCustomerFullName

ssn string □ nil?

111-11-1111

getCustomerFullNameResponse

getCustomerFullNameReturn (string): Mr. Henry Cui

SOAP Request Envelope:

```
  <?xml version="1.0" encoding="UTF-8" ?>
- <soapenv:Envelope xmlns:q0="http://simple.model.bank.rad7.itso"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  - <soapenv:Body>
    - <q0:getCustomerFullName>
        <ssn>111-11-1111</ssn>
      </q0:getCustomerFullName>
    </soapenv:Body>
  </soapenv:Envelope>
```

SOAP Response Envelope:

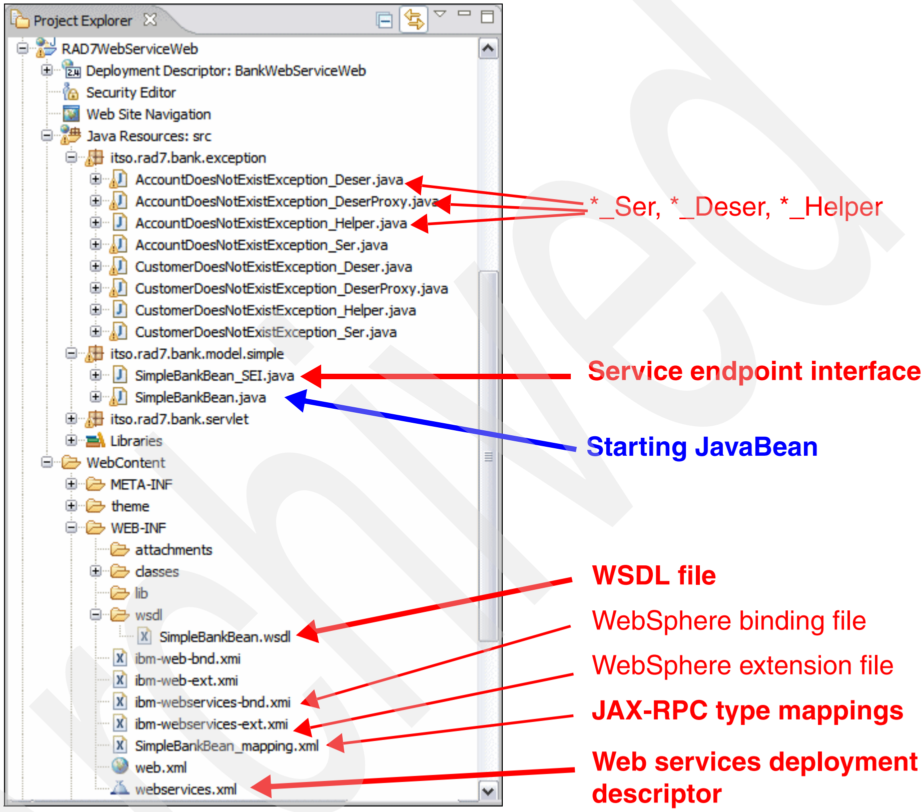
```
- <soapenv:Envelope
    xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header />
  - <soapenv:Body>
    - <p160:getCustomerFullNameResponse
        xmlns:p160="http://simple.model.bank.rad7.itso">
        <getCustomerFullNameReturn>Mr. Henry
          Cui</getCustomerFullNameReturn>
      </p160:getCustomerFullNameResponse>
    </soapenv:Body>
  </soapenv:Envelope>
```

Figure 18-9 Web Services Explorer: SOAP Request and Response

- Click **Finish** to exit the Web Services wizard.

Resources generated by the Web Service wizard

The wizard generates a number of files based on the choices made. Since the original Java classes are located in the RAD7WebServiceWeb project, all of the generated code is located in the same project. The generated files are visible in two different views, as seen in Figure 18-10 and Figure 18-11.


Project Explorer
RAD7WebServiceWeb
Deployment Descriptor: BankWebServiceWeb
Security Editor
Web Site Navigation
Java Resources: src
itso.rad7.bank.exception
AccountDoesNotExistException_Deser.java
AccountDoesNotExistException_DeserProxy.java
AccountDoesNotExistException_Helper.java
AccountDoesNotExistException_Ser.java
CustomerDoesNotExistException_Deser.java
CustomerDoesNotExistException_DeserProxy.java
CustomerDoesNotExistException_Helper.java
CustomerDoesNotExistException_Ser.java
itso.rad7.bank.model.simple
SimpleBankBean_SEI.java
SimpleBankBean.java
itso.rad7.bank.servlet
Libraries
WebContent
META-INF
theme
WEB-INF
attachments
classes
lib
wsdl
SimpleBankBean.wsdl
ibm-web-bnd.xmi
ibm-web-ext.xmi
ibm-webservices-bnd.xmi
ibm-webservices-ext.xmi
SimpleBankBean_mapping.xml
web.xml
webservices.xml

*_Ser, *_Deser, *_Helper

Service endpoint interface

Starting JavaBean

WSDL file

WebSphere binding file

WebSphere extension file

JAX-RPC type mappings

Web services deployment descriptor

Figure 18-10 Web services generated resources: Web project view

- The **service endpoint interface** is the Java interface that is implemented by the Web Service. This will include a subset of the public methods on the class that haves been exposed in the Web service.
- The **WSDL file** is in the WEB-INF folder: This WSDL is used by the server for deployment purposes, and is accessible to external clients through HTTP. For example, if the server is running on port 9080, you can retrieve the WSDL using this URL:

```
http://localhost:9080/BankWebServiceWeb/services/SimpleBankBean?wsdl
```

- The **Web services deployment descriptor** contains information used by the server to deploy the Web services in the project. The format of this file is defined by the Web Services for J2EE specification.
- The **WebSphere bindings file** is used to map local names to global names (for example, to map EJB references to real names used to register EJBs in JNDI).
- The **WebSphere extensions file** stores information relating to WebSphere extensions to the J2EE specification. This acts as an extension to the information contained within the deployment descriptor.
- The **WebSphere bindings file** and **WebSphere extensions file** are also used to configure the WS-Security. The WebSphere extensions file defines the settings that are going to be used and the extensions file is used to specify how the declared settings are implemented.
- The **JAX-RPC type mapping file** contains information on the relationships between types used in Java code and their equivalents in XML.
- ***_Ser** provides the bean serializer for converting the Java object to an XML complex type. ***_Deser** provides the bean deserializer for converting the XML complex type to Java object. ***_Helper** helps the Web services engine locate the proper serializer and deserializer for the Java object.

JSR 109 Web Services view

Note: In some cases, you might not be able to see the Web services or clients under the JSR-109 Web Services category. Restarting the Workbench should solve the problem.

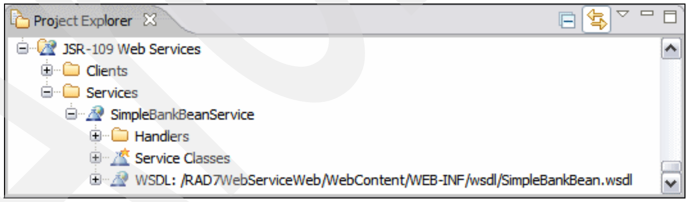


Figure 18-11 Web services generated files: Web Services view

The Web Services view only shows information about Web services defined in projects within the workspace.

- The **Service Classes** section shows how the Web Service is registered in a Web project as a Servlet (this can also be seen in the Web project's deployment descriptor).

- The **WSDL** section shows the location of the internally visible copy of the WSDL file for the Web Service.
- The **Handlers** section (empty in this case) lists the JAX-RPC handlers that are configured for this Web Service. Handlers provide a mechanism for intercepting the SOAP message. The JAX-RPC handler can examine and potentially modify the content of a SOAP message. Handlers can be used for logging, SOAP header processing, or caching. A limited form of encryption is also possible.

Creating a Web service JSP client and monitoring Web services

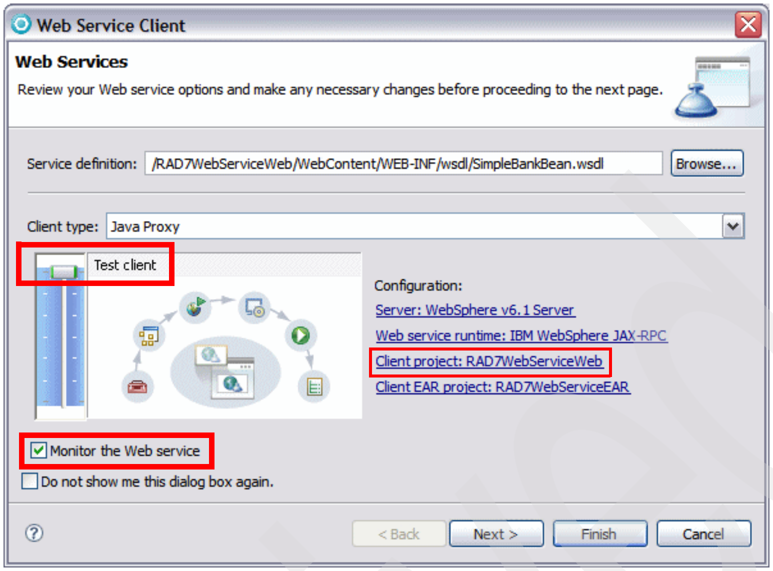
The Web Service Client wizard assists you in generating a Java bean proxy and a sample application. The sample Web application demonstrates how to code a proxy file.

To generate a client and test the client proxy, do these steps:

- Locate the WSDL file under **RAD7WebServiceWeb** → **WebContent** → **WEB-INF** → **wsdl**.

> **Note:** Alternatively, expand **JSR-109 Web Services** → **Services** → **SimpleBankBeanService**.

- Right-click **SimpleBankBean.wsdl** and select **Web Services** → **Generate Client**.
- In the Web Services Client dialog (Figure 18-12):
  - Move the slider up to the **Test** position. This provides options for testing the service using a JSP-based sample application.
  - Select **Monitor the Web service**.


Web Service Client

Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Service definition: /RAD7WebServiceWeb/WebContent/WEB-INF/wsdl/SimpleBankBean.wsdl

Client type: Java Proxy

Test client

Configuration:

Server: WebSphere v6.1 Server

Web service runtime: IBM WebSphere JAX-RPC

Client project: RAD7WebServiceWeb

Client EAR project: RAD7WebServiceEAR

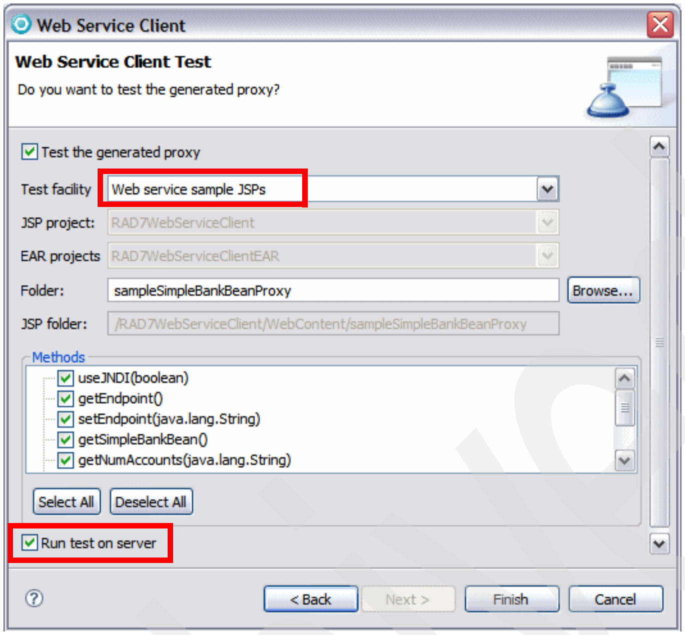
☑ Monitor the Web service

□ Do not show me this dialog box again.

Figure 18-12 Generate Web Service client

- We recommend that the Web service and Web service client are in separate Web and EAR projects:
  - Click **Client project: RAD7WebService Web**. The Specify Client Project Settings dialog comes up.
  - Change the client project name to **RAD7WebServiceClient**.
  - Select **Dynamic Web project** as the project type.
  - Change the client EAR project name to **RAD7WebServiceClientEAR**.
  - Click **OK**. The wizard creates the Web and EAR projects for you.
  - Click **Next**.
- In the Web Service Proxy Page dialog, accept the default (**no security**) and click **Next**.
- When prompted, enable automatic file overwrite.

- Next we consider the Web Service Client Test dialog (Figure 18-13).



Web Service Client Test

Do you want to test the generated proxy?

☑ Test the generated proxy

Test facility: Web service sample JSPs

JSP project: RAD7WebServiceClient

EAR projects: RAD7WebServiceClientEAR

Folder: sampleSimpleBankBeanProxy

JSP folder: /RAD7WebServiceClient/WebContent/sampleSimpleBankBeanProxy

Methods

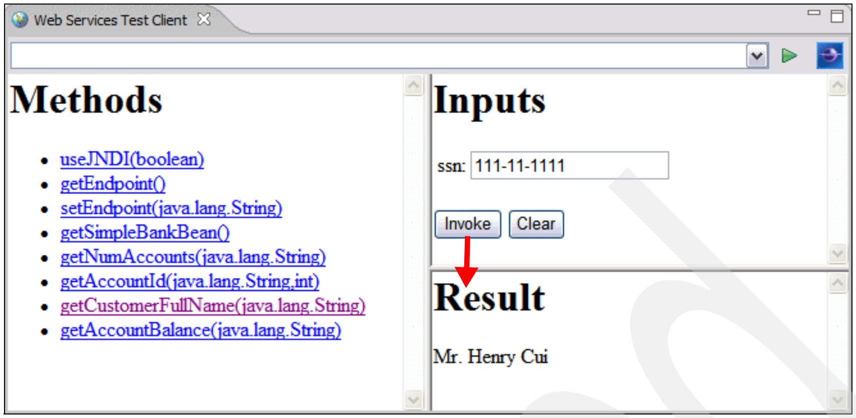
- ☑ useJNDI(boolean)
- ☑ getEndpoint()
- ☑ setEndpoint(java.lang.String)
- ☑ getSimpleBankBean()
- ☑ getNumAccounts(java.lang.String)

☑ Run test on server

Figure 18-13 Web Service Client Test

- In the Web Service Client Test dialog (Figure 18-13), use these settings:
  - Select **Test the generated proxy**.
  - Test facility: Select **Web Service sample JSPs** (default).
  - Folder: `sampleSimpleBankBeanProxy` (default). You can specify a different folder for the generated application if you want.
  - Methods: Leave all methods selected.
  - Select **Run test on server**.
  - Click **Finish**.
- The sample application starts and is displayed in a Web browser.
- Select the **getCustomerFullName** method, enter a valid value in the customer ID field (such as 111-11-1111), and then click **Invoke**.

  The results are displayed in the result pane (Figure 18-14).



Methods

- useJNDI(boolean)
- getEndpoint()
- setEndpoint(java.lang.String)
- getSimpleBankBean()
- getNumAccounts(java.lang.String)
- getAccountId(java.lang.String,int)
- getCustomerFullName(java.lang.String)
- getAccountBalance(java.lang.String)

Inputs

ssn: 111-11-1111

Result

Mr. Henry Cui

Figure 18-14 Sample JSP results

- The TCP/IP Monitor is also started. The TCP/IP Monitor lets you intercept and examine the SOAP traffic coming in and out of a Web service.
- If you invoke the **getEndpoint** method in the Sample JSP page, you will get a URL as follows:

```
http://localhost:8940/BankWebServiceEARWeb/services/SimpleBankBean
```

  You might see a port number other than 8940. It depends on what port number the wizard generated for the Monitor.

- If you select **Window** → **Preferences** → **Run/Debug** → **TCP/IP Monitor**, you can see the new Monitor is there to listen to the same port number. The TCP/IP Monitor is started and ready to listen to the SOAP request and direct it to the Web service provider.

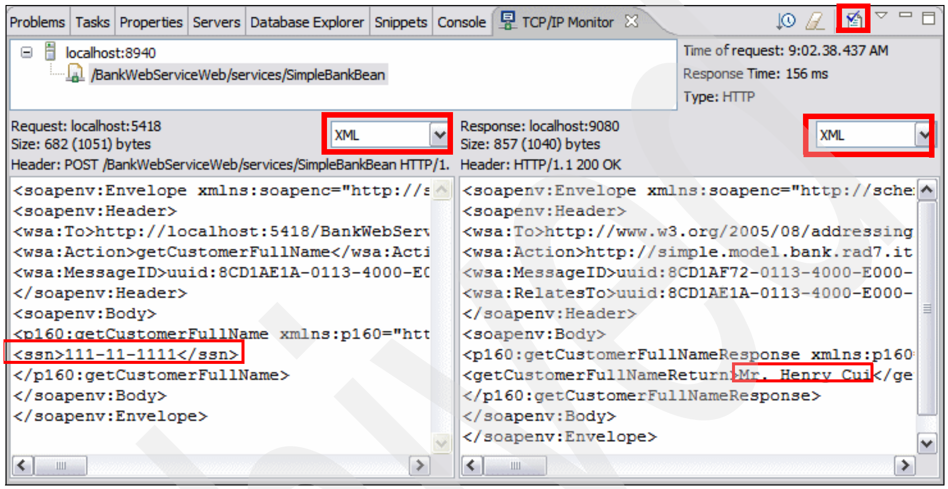
Behind the scenes:

- When you select **Monitor the Web service** in the Web Service wizard page, the Web Service Client wizard dynamically creates the TCP/IP Monitor for you. It uses certain algorithm to find an available listening port for the Monitor, and the sample JSP client page uses the URL to dynamically set the Web service endpoint to match the Monitor port.
- Using the wizard to create the TCP/IP Monitor is very handy as the user does not have to spend time to figure out how to redirect the SOAP request to the TCP/IP Monitor, especially in the case of monitoring remote Web services.

- All requests and responses are routed through the TCP/IP Monitor and appear in the TCP/IP Monitor view.

The TCP/IP Monitor view shows all the intercepted requests in the top pane, and when a request is selected, the messages passed in each direction are shown in the bottom panes (request in the left pane, response in the right). This can be a very useful tool in debugging Web services and clients.

Select the **XML** view to display the SOAP request and response in XML format (Figure 18-15).



localhost:8940
/BankWebServiceWeb/services/SimpleBankBean

Time of request: 9:02.38.437 AM
Response Time: 156 ms
Type: HTTP

Request: localhost:5418
Size: 682 (1051) bytes
Header: POST /BankWebServiceWeb/services/SimpleBankBean HTTP/1.
XML

```
<soapenv:Envelope xmlns:soapenc="http://s
<soapenv:Header>
<wsa:To>http://localhost:5418/BankWebServ
<wsa:Action>getCustomerFullName</wsa:Acti
<wsa:MessageID>uuid:8CD1AE1A-0113-4000-E0
</soapenv:Header>
<soapenv:Body>
<p160:getCustomerFullName xmlns:p160="htt
<ssn>111-11-1111</ssn>
</p160:getCustomerFullName>
</soapenv:Body>
</soapenv:Envelope>
```

Response: localhost:9080
Size: 857 (1040) bytes
Header: HTTP/1.1 200 OK
XML

```
<soapenv:Envelope xmlns:soapenc="http://sche
<soapenv:Header>
<wsa:To>http://www.w3.org/2005/08/addressing
<wsa:Action>http://simple.model.bank.rad7.it
<wsa:MessageID>uuid:8CD1AF72-0113-4000-E000-
<wsa:RelatesTo>uuid:8CD1AE1A-0113-4000-E000-
</soapenv:Header>
<soapenv:Body>
<p160:getCustomerFullNameResponse xmlns:p160
<getCustomerFullNameReturn>Mr. Henry Cui</ge
</p160:getCustomerFullNameResponse>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 18-15 TCP/IP Monitor

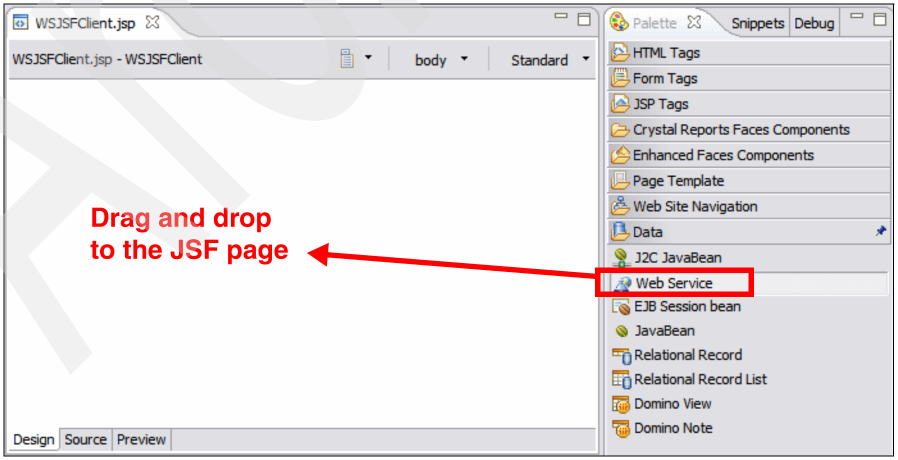
- ► To ensure that the Web service SOAP traffic is WS-I compliant, you can generate a log file by clicking the icon ☑ at the top right corner. In the dialog box that opens, select a name for the log file and specify where you want it to be stored (for example in the client project).
- ► The log file is validated for WS-I compliance. You will see a confirmation dialog stating *The WS-I Message Log file is valid*. You can open the log file in an XML editor to examine its contents.
- ► Stop the TCP/IP Monitor by selecting **Window** → **Preferences** → **Run/Debug** → **TCP/IP Monitor** and **stop** the TCP/IP Monitor from the list.

Note: You might notice that there are some warnings in the workspace. If you do not want to see these warnings, select **Window** → **Preferences** → **Java** → **Compiler** → **Errors/Warnings** → expand **Deprecated and restricted API** section and change the option for **Deprecated API** from **Warning** to **Ignore**. Expand the **Unnecessary code** section, and change the option for **Local variable is never read** from **Warning** to **Ignore**. Click **OK**.

Creating a Web service JSF client

The Web Service Discovery Dialog allows you to discover a Web service that exists online or in the workspace, create a proxy to the Web service, and then place the methods of the proxy on a Faces JSP file:

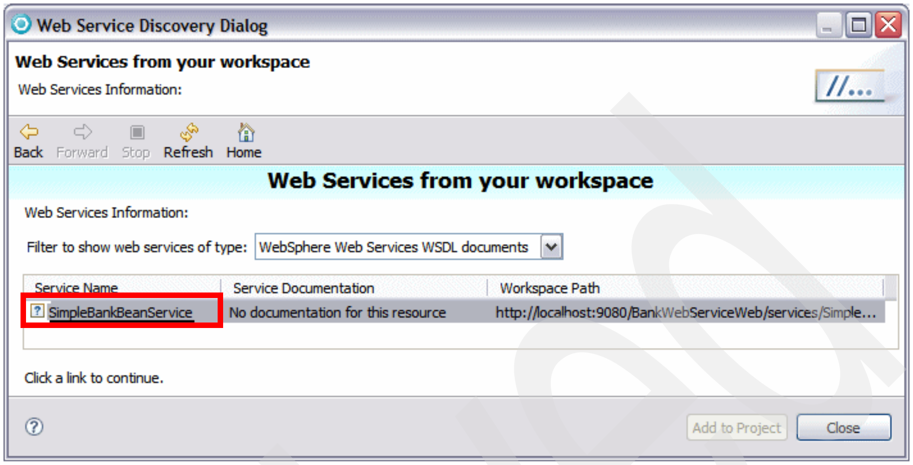
- Remove the `RAD7WebServiceClientEAR` from the server using **Add and Remove Projects** (we will add a project to the EAR and automatic publishing gets into the way), or expand the server, right-click the project, and select **Remove**.
- Create a dynamic Web project In the Web perspective select **File** → **New** → **Dynamic Web Project**.
  - Enter **RAD7WebServiceJSFClient** as the project name.
  - In the Configurations section, select **Faces Project** to add the required JSF facets to the project facets list.
  - Select **RAD7WebServiceClientEAR** as the EAR project name.
  - Click **Finish**. If you are prompted to open the Web perspective, click **Yes**.
- In the **RAD7WebServiceJSFClient** project, right-click the **WebContent** and select **New** → **Web Page**.
- Enter **WSJSFClient** as the file name. Select **JSP** as the template and click **Finish**.
- The `WSJSFClient.jsp` opens in an editor. Select the **Design** tab.
- In the Palette, select the **Data** tab. Select **Web Service** and click into the JSF page (Figure 18-16).



Drag and drop to the JSF page

Figure 18-16 Drag and drop Web Service to JSF design view

- The **Web Services Discovery Dialog** opens. Select **Web services from your workspace** and the dialog shown in Figure 18-17 opens.



Web Services from your workspace

Web Services Information:

Web Services from your workspace

Web Services Information:

Filter to show web services of type: WebSphere Web Services WSDL documents

Service Name	Service Documentation	Workspace Path
SimpleBankBeanService	No documentation for this resource	http://localhost:9080/BankWebServiceWeb/services/Simple...

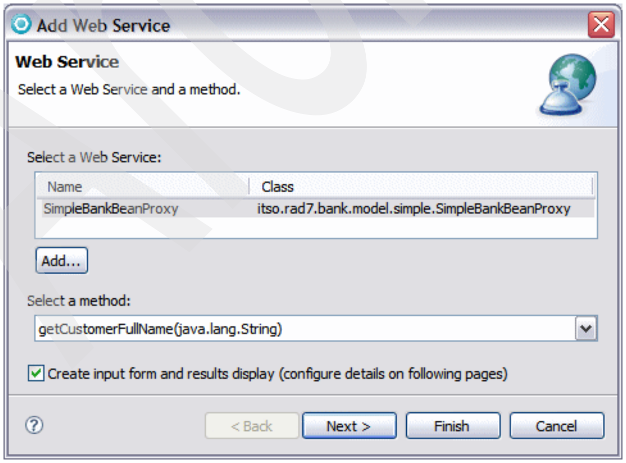
Click a link to continue.

Figure 18-17 Web Services Discovery Dialog

- Click **SimpleBankBeanService** to display the service with its port. Notice that the URL is in ?WSDL format to dynamically retrieve the WSDL file:

```
http://localhost:9080/BankWebServiceWeb/services/SimpleBankBean?WSDL
```

- Select **Port: SimpleBankBean** and click **Add to Project**.
- The Web service you selected is now listed in the list of Web services. Select **getCustomerFullName(java.lang.String)** from the drop-down menu and click **Next** (Figure 18-18).



Web Service

Select a Web Service and a method.

Select a Web Service:

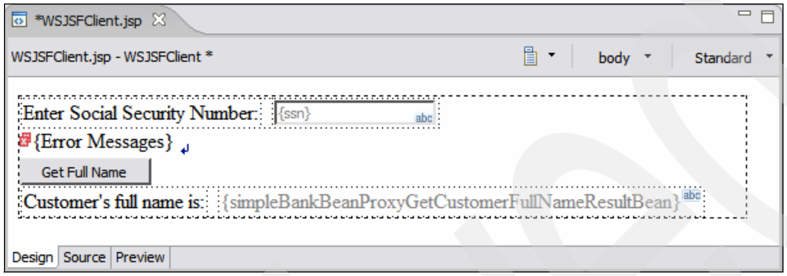
Name	Class
SimpleBankBeanProxy	itso.rad7.bank.model.simple.SimpleBankBeanProxy

Select a method: getCustomerFullName(java.lang.String)

☑ Create input form and results display (configure details on following pages)

Figure 18-18 Add Web Service

- In the Input form page, change the label to **Enter Social Security Number:**.
- Click **Options** and change the label from Submit to **Get Full Name**. Click **OK**.
- Click **Next**.
- In the Results form page, change the **Label** to **Customer's full name is:**.
- Click **Finish** to generate the input and output parts into the JSF page (Figure 18-19). Save the file.



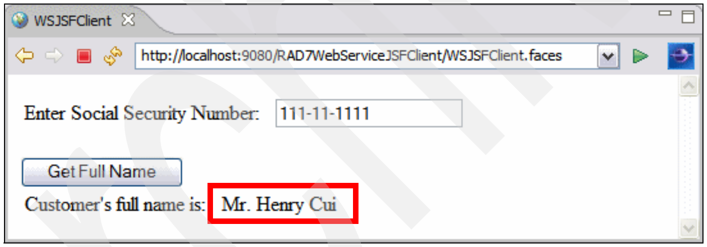
Enter Social Security Number: {ssn}

{Error Messages}

Customer's full name is: {simpleBankBeanProxyGetCustomerFullNameResultBean}

Figure 18-19 JSF page with Web service invocation

- Right-click `WSJSFClient.jsp` and Select **Run As** → **Run on Server**. The client application is deployed to the server for testing (Figure 18-20).



http://localhost:9080/RAD7WebServiceJSFClient/WSJSFClient.faces

Enter Social Security Number: 111-11-1111

Customer's full name is: Mr. Henry Cui

Figure 18-20 JSF client run

> **Tip:** You can add a Web service to a JSF page in the Web Diagram by clicking **Data** in the pop-up toolbar of a page (the same discovery dialog opens).

Handling exceptions in Web services

When the Web service throws an exception to indicate that an error has occurred, the WebSphere Web services engine converts the exception to a SOAP fault and returns the fault in the SOAP response. On the client side, the WebSphere Web services engine converts the SOAP fault to a Java exception.

There are four types of exceptions that can be thrown from the server:

- java.rmi.RemoteException
- java.lang.RuntimeException
- javax.xml.rpc.soap.SOAPFaultException (subclass of RuntimeException)
- A user-defined exception (mapped from the wsdl:fault construct)

The client side will receive one of the following types of exceptions:

- java.rmi.RemoteException
- javax.xml.rpc.soap.SOAPFaultException
- A user-defined exception

> **Note:** For detailed information about exception handling in JAX-RPC Web services, refer to the developerWorks article at:
>
> http://www.ibm.com/developerworks/webservices/library/ws-tip-jaxrpc.html

In the last section we created a Web services JSF client to invoke the getCustomerFullName operation. If the customer ID does not exist in the bank, the Web service throws a CustomerDoesNotExistException, which is a user-defined exception. On the client side, the generated Web service conveys the exception to the client so that the client is able to catch the user defined exception and display a user friendly message in the JSF page.

In this section we show you how to handle use-defined exception in the JSF client code.

Adding exception handling code

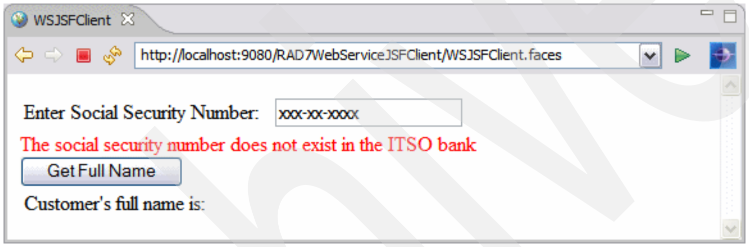
To add exception handling code to the JSF client, do these steps:

- Open the **WSJSFClient.jsp**.
- In the Design tab, select the **Get Full Name** button.
- In the Quick Edit view, you can see the generated source code.
- Add a new catch statement before the catch(Throwable e) statement (Example 18-1).

Example 18-1 Exception handling in Web services

```
try {
   SimpleBankBeanProxyGetCustomerFullNameResultBean = getSimpleBankBeanProxy()
         .getCustomerFullName(
               getSimpleBankBeanProxyGetCustomerFullNameParamBean()
                     .getSsn());
} catch (CustomerDoesNotExistException e) {
   this.getFacesContext().addMessage(this.getTextSsn1().getId(),
   new FacesMessage
         ("The social security number does not exist in the ITSO bank"));
} catch (Throwable e) {
   logException(e);
}
return null;
```

- To resolve the compilation errors, right-click in the JSP page and select **Edit Page Code**.
- In the WSJSFClient.java code, select **Source** → **Organize Imports**. Save the code.
- In the page designer, select **{Error Messages}**. In the Properties view, for **h:messages**, type **color: red** in the Styles: Props field.
- Rerun the JSF application in the server and enter a bad social security number to test the error handling (Figure 18-21).


WSJSFClient
http://localhost:9080/RAD7WebServiceJSFClient/WSJSFClient.faces
Get Full Name

Enter Social Security Number: xxx-xx-xxxx

The social security number does not exist in the ITSO bank

Customer's full name is:

Figure 18-21 Exception handing in JSF client

Watching the SOAP fault using the TCP/IP Monitor

To monitor the SOAP traffic, do these steps:

- Start the TCP/IP Monitor (**Window** → **Preferences** → **Run/Debug** → **TCP/IP Monitor**) and notice the port.
- In the Design tab, select the Get Full Name button, and in the Quick Edit view add one line to the code (where xxxx is the Monitor port):

```
try {
   getSimpleBankBeanProxy().setEndpoint
      ("http://localhost:xxxx/BankWebServiceWeb/services/SimpleBankBean");
```

- Save the JSP and run the client once more with a bad social security number.
- Notice the response in the TCP/IP view:

```
<soapenv:Envelope ..........">
<soapenv:Header>
......
```

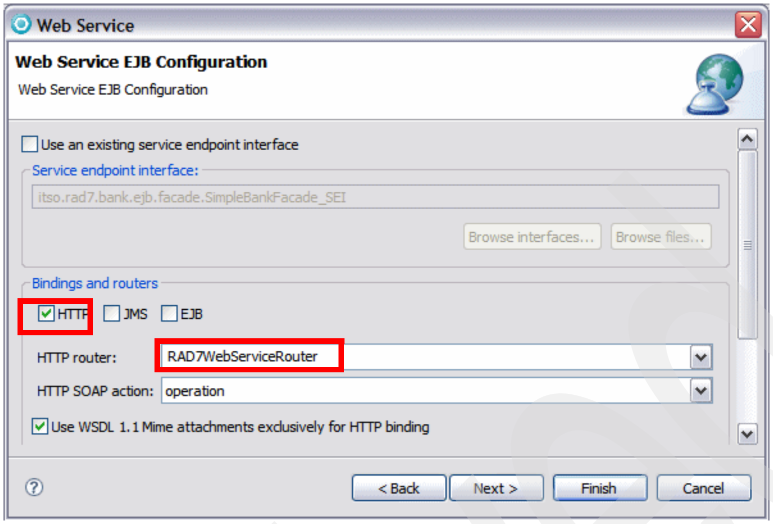
```
</soapenv:Header>
<soapenv:Body>
<soapenv:Fault>
<faultcode xmlns:p790="http://exception.bank.rad7.itso">
                p790:CustomerDoesNotExistException</faultcode>
<faultstring>
   <![CDATA[itso.rad7.bank.exception.CustomerDoesNotExistException:
                Bank get customer failed: xxx-xx-xxxx]]></faultstring>
<detail encodingStyle="">
<p790:CustomerDoesNotExistException
xmlns:p790="http://exception.bank.rad7.itso">
<message>Bank get customer failed: xxx-xx-xxxx</message>
</p790:CustomerDoesNotExistException>
</detail>
</soapenv:Fault>
</soapenv:Body>
</soapenv:Envelope>.
```

The SOAP fault contains a fault code, a fault string, and detail element. The fault string provides a human readable explanation of the fault. The detail element carries application specific information about the error.

Creating Web services from an EJB

The process of creating a Web Service from an EJB session bean is very similar to the process for a JavaBean:

- Expand the EJB project **RAD7WebServiceEJB** → **Deployment Descriptor** → **Session Beans**.
- Right-click **SimpleBankFacade**, and select **Web Services** → **Create Web Service**.
- The **Web Service type** should be **Bottom Up EJB Web Service**. Leave all the settings at default and click **Next**.
- In the Web Service EJB Configuration page (Figure 18-22), do these steps:
  - Select **HTTP** as the binding.
  - With **HTTP** as the binding protocol, we have to specify which Web project will contain the routing servlet to forward requests to the EJB. Change the HTTP router to **RAD7WebServiceRouter**.



Web Service EJB Configuration

Web Service EJB Configuration

☐ Use an existing service endpoint interface

Service endpoint interface: itso.rad7.bank.ejb.facade.SimpleBankFacade_SEI

Bindings and routers

☑ HTTP ☐ JMS ☐ EJB

HTTP router: RAD7WebServiceRouter

HTTP SOAP action: operation

☑ Use WSDL 1.1 Mime attachments exclusively for HTTP binding

Figure 18-22 Web Services EJB Configuration

- You can go through the rest of the Web Service wizard pages (and accept the defaults), or click **Finish**.

Behind the scenes:

- The multiprotocol binding support is available starting with WebSphere Application Server v5.1.1.
- The tooling support for multiprotocol binding is a new feature of Application Developer v7. The multiprotocol Java API for JAX-RPC is an extension of the JAX-RPC programming model, and extends the existing JAX-RPC capabilities to support binding types like HTTP (SOAP over HTTP), EJB (RMI over IIOP) and JMS (SOAP over JMS).
- SOAP over HTTP is the most commonly used Web services binding type. However, if you are looking for a more reliable and scalable messaging mechanism, you might consider using SOAP over JMS to guarantee message delivery. Using the RMI/IIOP protocol instead of a SOAP-based protocol yields better performance because it avoids the extraneous marshalling and unmarshalling. It also enables you to get support for client transactions in EJB Web services. Only SOAP over HTTP is WS-I compliant. You will get a non-compliant warning during Web services code generation if you use EJB or JMS binding types.
- For more detailed information, refer to the Redbooks publication, *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257.

- Expand the EJB Project **RAD7WebServiceEJB → ejbModule → META-INF → WSDL**.
- Expand the RAD7WebServiceEJB project and explore the generated code, which is similar to the code in the RAD7WebServiceWebB project, but under ejbModule and META-INF.
- Right-click **SimpleBankFacade.wsdl** and select **Web Services → Test with Web Services Explorer**.
- Test the EJB Web service using the Web Services Explorer to make sure it works.
- Open the deployment descriptor of the RAD7WebServiceRouter project and notice the generated servlet.
- We can also test the EJB Web service by running the TestClient.jsp (in RAD7WebServiceClient/WebContent/sampleSimpleBanBeanProxy):
  - Invoke the setEndpoint method with the URL of the EJB Web service:

    ```
    http://localhost:9080/RAD7WebServiceRouter/services/SimpleBankFacade
    ```
  - Run some operations.
  - This works because the EJB and the JavaBean Web service have the same operations.

Creating a top-down Web service from a WSDL

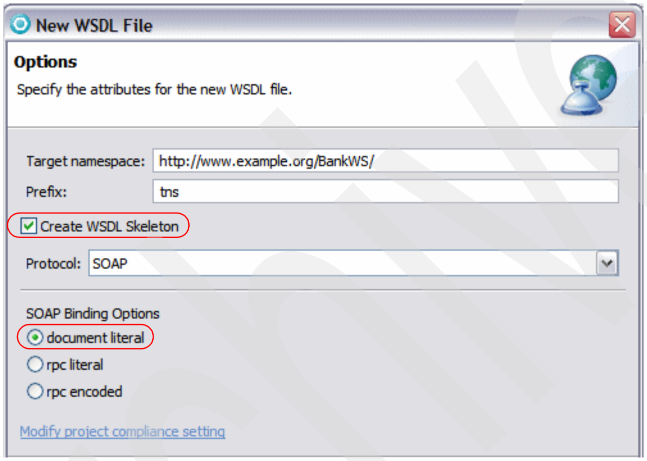
When creating a Web service using a top-down approach, first you design the implementation of the Web service by creating a WSDL file. You can do this using the WSDL editor. You can then use the Web Services wizard to create the Web service and skeleton Java classes to which you can add the required code. The top-down approach is the recommended way of creating a Web service.

Designing the WSDL using the WSDL editor

In this section, we create a WSDL with two operations: getAccount (using an account ID to retrieve an account) and getCustomer (using a customer ID to retrieve a customer).

The WSDL editor allows you to easily and graphically create, modify, view, and validate WSDL files. To create a WSDL, do these steps:

- Create a new dynamic Web project to host the new Web service:
  - Web project: `RAD7TopDownBankWS`
  - EAR project: `RAD7TopDownBankEAR`
- Create a WSDL file:
  - Right-click the `RAD7TopDownBankWS/WebContent` folder and **New** → **Other** → **Web services** folder → **WSDL**, and click **Next**.
  - Change the File name to **BankWS.wsdl** and click **Next**.
  - In the Options page, leave the default and click **Finish** (Figure 18-23).


New WSDL File

Options

Specify the attributes for the new WSDL file.

Target namespace: http://www.example.org/BankWS/

Prefix: tns

☑ Create WSDL Skeleton

Protocol: SOAP

SOAP Binding Options

● document literal

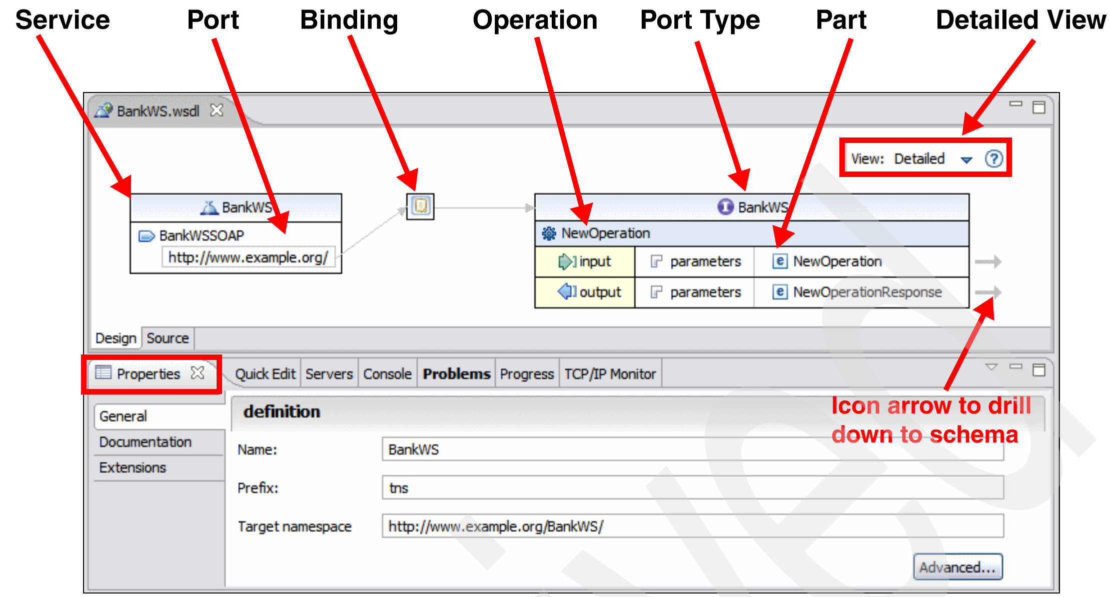
○ rpc literal

○ rpc encoded

Modify project compliance setting

Figure 18-23 New WSDL File wizard

- The WSDL editor is now opened with the new WSDL file. Select the **Design** tab in the WSDL editor.
  - In the WSDL editor, switch to **Detailed** View by clicking the **down arrow** at the top right corner.
  - Select the **Properties** view. Now you are ready to edit the WSDL file, as shown in Figure 18-24.



definition

Name: BankWS

Prefix: tns

Target namespace: http://www.example.org/BankWS/

Figure 18-24 WSDL editor

- Editing the WSDL file:
  - Change the operation name by double-clicking **NewOperation** and overtyping the name with **getAccount**. (You can also select the operation and change the name in the Properties view.)
  - Add a new operation. In the Design view, right-click the port type **BankWS**, select **Add Operation**, and name the operation **getCustomer**.
  - To change the input type of the WSDL operation **getAccount**, click the **indicator icon (**right arrow**)** to the right of the input operation to drill down into the schema.
- The Inline Schema Editor opens. Switch to the **Detailed** view (Figure 18-25 shows what we want to accomplish).

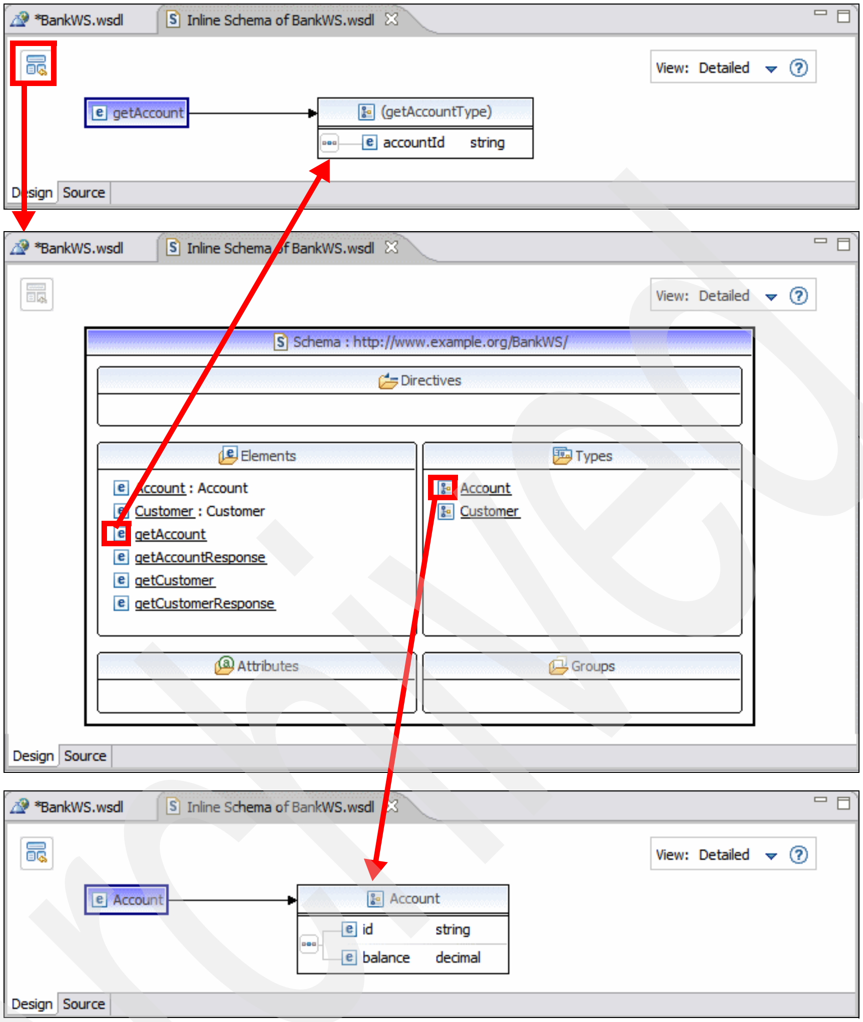


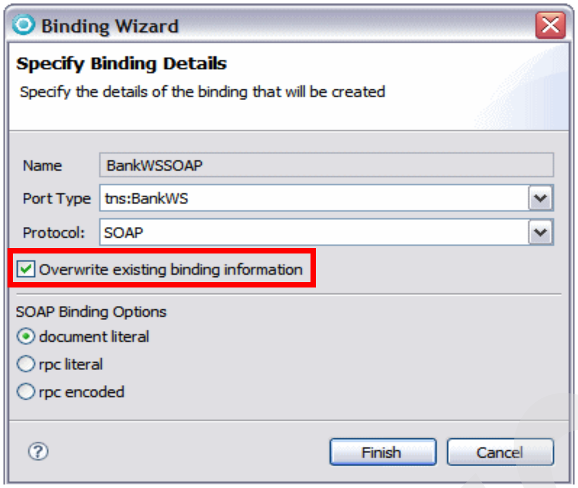
Figure 18-25 Inline Schema editor

- Change the element name from in to **accountId**, and select the type as xsd:string.
- Clicking the icon at the top left shows all the directives, elements, types, attributes, and groups in the WSDL.
- In the Types category, right-click and select **Add Complex Type**. Change the name to **Account**.

- Right-click **Account** → **Add Sequence**.
- Right-click **Account** again → **Add Element** → Change the name to **id**.
- Right-click the content model object → **Add Element**. Change the name to **balance**, and select the type as **decimal** using the **Browse** option.
- Click the icon at the top left corner. In the Types section, right-click → **Add Complex Type** → change the name to **Customer.**
- Right-click **Customer** → **Add Sequence.**
- Right-click **Customer** again → **Add Element** → Change the name to **ssn.**
- Right-click **Customer** → **Add Element** → Change the element name to **firstName**.
- Right-click **Customer** → **Add Element** → Change the name to **lastName**.
- Right-click **Customer** → **Add Element** → Change the name to **title**.
- Click the icon at the top left corner. We have to add global element for the complex types. Right-click the **Elements** category and click **Add Element**. Change the element name to **Account** → right-click **Account** → **Set Type** → **Browse** → select **Account**.
- Add global element **Customer**, set type to **Customer**.
- In the WSDL editor, click the **arrow icon** to the right of the element **getAccountResponse** to drill down into the schema.
- Right-click element **out** → **Set Type** → **Browse** → select **Account**.
- In the WSDL editor, click the **arrow icon** to the right of the element **getCustomer** to drill down into the schema. Change the element name from **in** to **customerId**.
- In the WSDL editor, click the **arrow icon** to the right of the element **getCustomerResponse** to drill down into the schema. Right-click element **out** → **Set Type** → **Browse** → select **Customer**.
- Save and close the Schema editor.

► In the WSDL editor, right-click the **binding** icon (as shown in Figure 18-24 on page 844) → **Generate Binding Content**.

In the Binding wizard, select **Overwrite existing binding information** and click **Finish** (Figure 18-26).



Binding Wizard

Specify Binding Details

Specify the details of the binding that will be created

Name: BankWSSOAP

Port Type: tns:BankWS

Protocol: SOAP

☑ Overwrite existing binding information

SOAP Binding Options

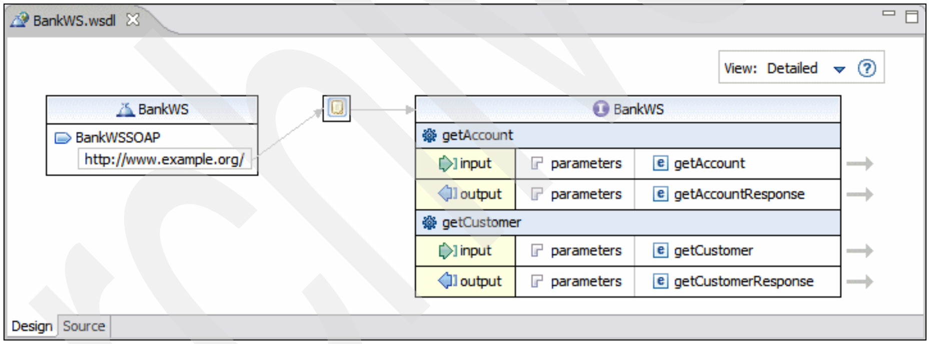
● document literal

○ rpc literal

○ rpc encoded

Figure 18-26 Specify Binding Details wizard

- Save the WSDL file (Figure 18-27).



BankWS.wsdl

View: Detailed

BankWS

BankWSSOAP

http://www.example.org/

BankWS

getAccount		
input	parameters	getAccount
output	parameters	getAccountResponse
getCustomer		
input	parameters	getCustomer
output	parameters	getCustomerResponse

Figure 18-27 BankWS.wsdl

- Right-click **BankWS.wsdl** → **Validate**. You should not receive any WSDL validation errors or warnings in the Problems view.

Note: In Application Developer V6, you received a dialog box to display the validation result. At the time of writing, the behavior is that the dialog box is not displayed. The user has to check the Problems view to see if the WSDL is valid or not.

- If you have a problem when creating the WSDL file, you can import the `BankWS.wsdl` from the `C:\7501code\webservices` folder.

Generating the skeleton JavaBean Web service

To generate a skeleton JavaBean Web service from a WSDL, do these steps:

- Right-click **BankWS.wsdl** → **Web Services** → **Generate Java bean skeleton**.
- Keep the slider at **Start service** level. Notice that code is generated into the `RAD7TopDownBankWS` project.
- Click **Finish**.
- After the code generation, the skeleton class **BankWSSOAPImpl.java** opens in the Java editor.

Implement the generated JavaBean skeleton

You have to provide the business logic for the generated JavaBean skeleton:

- We use the simple implementation shown in Example 18-2.

Example 18-2 Implementation of the Generated JavaBean skeleton

```
package org.example.www;
import java.math.BigDecimal;
public class BankWSSOAPImpl implements org.example.www.BankWS_PortType {
   public org.example.www.Account getAccount(java.lang.String accountId)
         throws java.rmi.RemoteException {
      Account account = new Account();
      account.setId(accountId);
      account.setBalance(new BigDecimal(1000.00));
      return account;
   }
   public org.example.www.Customer getCustomer(java.lang.String customerId)
         throws java.rmi.RemoteException {
      Customer customer = new Customer();
      customer.setSsn(customerId);
      customer.setFirstName("Henry");
      customer.setLastName("Cui");
      customer.setTitle("Mr.");
      return customer;
   }
}
```

Test the generated Web service

Notice that a **wsdl** folder has been generated under `WebContent/WEB-INF`. This WSDL is the final service WSDL and includes the target address.

Note: You cannot use the `BankWS.wsdl` in the `WebContent` folder to test the Web service. The Web service endpoint is set to `http://www.example.org/` when we created this WSDL. The Web service code generation takes the existing WSDL and generate a new WSDL in the `WebContent/WEB-INF/wsdl` folder with the correct endpoint.

- To test the Web service, expand Web Project **RAD7TopDownBankWS** → **WebContent** → **WEB-INF** → **wsdl** → **BankWS.wsdl** and select **Web Services** → **Test with Web Services Explorer**.

  Test the **getCustomer** and **getAccount** operation. You should see that the correct result is displayed in the Web Services Explorer.

Explore!

Once you have created a Web service, you might want to make changes to it:

- For example, you might want to add a new WSDL operation `getBalanceByAccountId` to the WSDL. Once the WSDL is changed, you have to regenerate the Web service code and the existing business logic will be wiped out.
- To retain your changes while updating the Web service, you can use the **skeleton merge** feature. This allows you to regenerate the Web service while keeping your changes intact.
- This option can be enabled by selecting **Window** → **Preferences** → **Web Services** → **Resource Management** → **Merge generated skeleton file**.

Creating Web services with Ant tasks

If you prefer not to use the Web Service wizard, you can use Ant tasks to create Web service using the IBM WebSphere JAX-RPC runtime environment. The Ant tasks support creating Web services using both top-down and bottom-up approaches. Once you have created your Web service, you can then deploy it to a server, test it, and publish it as a business entity or business service.

In this section, we will use Ant tasks to automate the top-down code generation process as we did in the last section.

- Create a new dynamic Web project to host the Web service generated by Ant tasks: **RAD7WebServiceAnt** in **RAD7WebServiceEAR.**
- Copy the **BankWS.wsdl** from `RAD7TopDownBankWS/WebContent` folder to the **RAD7WebServiceAnt** folder (not under `WebContent`).

- Right-click **RAD7WebServiceAnt** and select **New** → **Other** → **Web Services** → **Ant Files** and click **Next**.
  - A file named **wsgen.xml** is created.
  - Two extra files are imported into the project: **axisclient.properties** and **axisservice.properties**. **Delete** these files. These two files are used to create Axis Web services and are not required for our example.
- Right-click **RAD7WebServiceAnt** and select **Import** → **Import**:
  - Select **General** → **File System** and then click **Next**.
  - Click **Browse** and locate the directory:

    ```
    %RAD_HOME%\runtimes\wsdk\wsgenTemplates
    C:\IBM\SDP70\runtimes\wsdk\wsgenTemplates
    ```

  - Select **was_jaxrpc_tdjava.properties** file. This sample properties file is referred to by `wsgen.xml`. Do not select other files. Click **Finish**.
- Open **wsgen.xml**, find the `<property>` tag, and change it to:

  ```
  <property file="was_jaxrpc_tdjava.properties"/>
  ```

  Save and close this file.
- Open **was_jaxrpc_tdjava.properties**, find the line for `InitialSelection`, and change it to:

  ```
  InitialSelection=/RAD7WebServiceAnt/BankWS.wsdl
  ```

  Save and close this file.

Running the Web service Ant task

To run the Ant task, right-click **wsgen.xml** and select **Run As** → **3 Ant Build**:

- In the Ant dialog, select the **JRE** tab and select **Run in the same JRE as the workspace**.
- Click **Apply** and then click **Run**.

The Web service is generated:

- The WSDL file for the service is created and placed in `WEB-INF\wsdl`.
- Several deployment descriptors are placed in the `WEB-INF` folder.
- Other Web service artifacts are generated into Java Resources.
- You can implement the generated skeleton (`BankWSSOAPImpl`) and test the Web service, as we did in the last section.

Generating SDO facades for Web services

JAX-RPC 1.1 (JSR 101) defines the standard mapping of WSDL and schema to Java in J2EE 1.4. The schema portion of the mapping does not support several common features of schema such as choices, substitution groups, simple type restrictions, and wildcards. JAX-RPC 1.1 requires that the unsupported schema types map to the Java `SOAPElement` type. Depending on the schema of the WSDL file, this might result in a Java service endpoint interface programming model that requires extensive programming effort to deal with the `SOAPElement` types.

You have the option of disabling the JAX-RPC schema to Java mappings that the wizards use by default and instead using the Java Eclipse EMF 2.2 SDO representation of XML schema. This can be useful if the XML schema contains content that the JAX-RPC mappings do not handle gracefully.

The SDO facade pattern is illustrated in Figure 18-28.

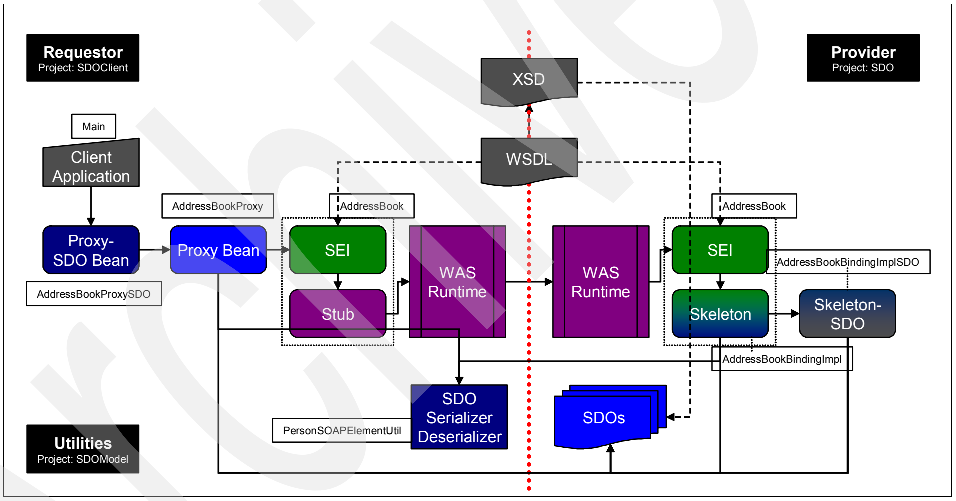


Figure 18-28 SDO facade for Web services

Note: For more detailed information about the SDO facade for Web services, refer to the developerWorks article at:

`http://www.ibm.com/developerworks/webservices/library/ws-emfsdo/`

Application Developer v7 automates the generation and adaptation of SDOs to Java Web services and clients, which automates the techniques in the foregoing article.

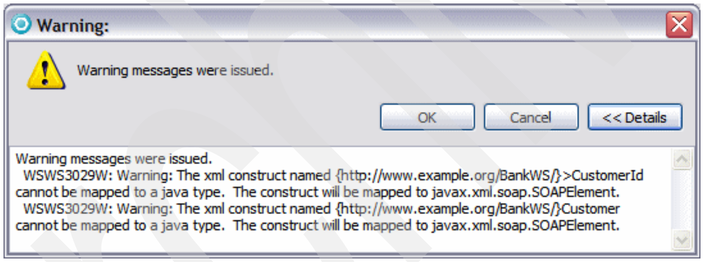
In this section, we demonstrate the problem in JAX-RPC 1.1 XML to Java mapping and show how to use the SDO facade to work around the problem:

- Create a new dynamic Web project to host the new Web service: **RAD7WebServiceSDOFacade** in **RAD7WebServiceEAR**.
- Import the **BankSDOFacadeWS.wsdl** from `C:\7501code\webservices` into the **WebContent** folder of the Web project `RAD7WebServiceSDOFacade`.
- Open the WSDL file and in the Source tab you can see that both WSDL input and output operations use the `<xsd:choice>` type:

```
<xsd:choice>
   <xsd:element name="ssn" type="xsd:string"></xsd:element>
   <xsd:element name="sin" type="xsd:string"></xsd:element>
</xsd:choice>
```

  Social security number (ssn) is used in the United States and social insurance number (sin) is used in Canada.

- Right-click **BankSDOFacadeWS.wsdl** and select **Web Services** → **Generate Java bean skeleton**.
- Keep the slider at **Start service** and click **Finish**. You receive the warning dialog, shown in Figure 18-29.



Warning messages were issued.

```
Warning messages were issued.
  WSWS3029W: Warning: The xml construct named {http://www.example.org/BankWS/}>CustomerId
cannot be mapped to a java type.  The construct will be mapped to javax.xml.soap.SOAPElement.
  WSWS3029W: Warning: The xml construct named {http://www.example.org/BankWS/}Customer
cannot be mapped to a java type.  The construct will be mapped to javax.xml.soap.SOAPElement.
```

Figure 18-29 Warning message for unsupported JAX-RPC data types

- Click **OK** to continue the code generation. The generated skeleton contains this code:

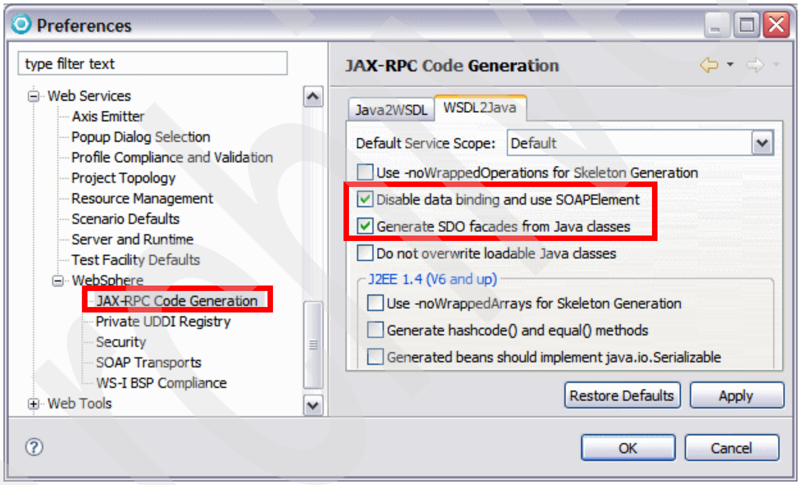
```
package org.example.www;
public class BankWSSOAPImpl implements org.example.www.BankWS_PortType{
   public javax.xml.soap.SOAPElement getCustomer
               (javax.xml.soap.SOAPElement parameters)
            throws java.rmi.RemoteException {
       return null;
   }
}
```

In the method signature, `javax.xml.soap.SOAPElement` is listed as the method input and output. To use the `SOAPElement` class, the user has to deal with the low-level SOAP with attachments APIs for Java (SAAJ) to access XML instance data. To work around this problem, you can use the SDO facade for Web services tooling.

Generating an SDO facade

To have the Web Service wizard generate SDOs when creating a top-down Web service or a Web service client, change some Web services preferences:

- Select **Window** → **Preferences** → **Web Services** → **WebSphere** → **JAX-RPC Code Generation**.
- On the **WSDL2Java** tab, select **Disable data binding and use SOAPElement** and **Generate SDO facades from Java classes** (Figure 18-30). Click **OK**.



Default Service Scope: Default

☐ Use -noWrappedOperations for Skeleton Generation
☑ Disable data binding and use SOAPElement
☑ Generate SDO facades from Java classes
☐ Do not overwrite loadable Java classes

J2EE 1.4 (V6 and up)

☐ Use -noWrappedArrays for Skeleton Generation
☐ Generate hashcode() and equal() methods
☐ Generated beans should implement java.io.Serializable

Figure 18-30 Preference setting for SDO facade Web services

Tip: The Web services preferences page is very useful and has great impact on Web services development. For more detailed information, refer to the developerWorks article at:

`http://www.ibm.com/developerworks/rational/library/07/0508_cui/`

- Create Skeleton Web services and Web services client using SDO facade tooling:

- Right-click **BankSDOFacadeWS.wsdl** and select **Web Services** → **Generate JavaBean skeleton**:
- Move the slider for the **service** to the **Start service** position**.**
- Move the slider for the **client** to the **Start client** position**.**
- Click the link for the client project to create the Web services client in a new project:
    - Client project: **RAD7WebServiceSDOFacadeClient**
    - Client Project type: **Dynamic Web Project**
    - Client EAR project: **RAD7WebServiceSDOFacadeClientEAR**
    - Click **OK**.
- Click **Finish**.

Generated code

A number of packages are generated into both server and client projects. Some packages hold the SDO objects (`org.example.bank.ws.impl`) and the interfaces (`org.example.bank.ws`), and other packages hold helper classes.

You can now use the navigatable and programmable SDO classes instead of traversing the `SOAPElement`:

```
public class BankWSSOAPImplSDO {
   public Customer getCustomer(CustomerIdType parameters) {
      return null;
   }
}
```

Implementing the SDO Web service server and client

Implement the generated server side skeleton. Copy/Paste the code as listed in Example 18-3 to the generated **BankWSSOAPImplSDO.java**.

Example 18-3 Sever side implementation

```
package org.example.www;
import org.example.bank.ws.Customer;
import org.example.bank.ws.CustomerIdType;
import org.example.bank.ws.WsFactory;

public class BankWSSOAPImplSDO {
   public Customer getCustomer(CustomerIdType parameters) {
      Customer customer = WsFactory.eINSTANCE.createCustomer();
      customer.setFirstName("Henry");
      customer.setLastName("Cui");
      customer.setTitle("Mr.");
```

```
        if (parameters.getSsn() == null)
           customer.setSin(parameters.getSin());
        else
           customer.setSsn(parameters.getSsn());
        return customer;
    }
}
```

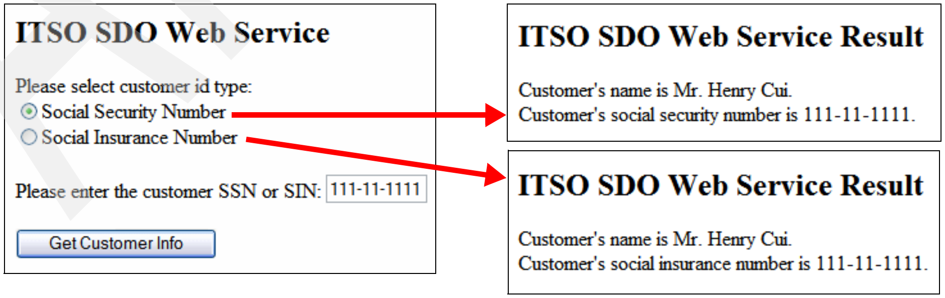
Create the Web services Web client to invoke the generated SDO facade proxy:

- Import **CustomerInquiry.html** from `C:\7501code\webservices` into the `RAD7SDOFacadeBankWSClient/WebContent` folder.
- Right-click Web project **RAD7SDOFacadeBankWSClient** and **New** → **Servlet**.
  - Java Package: **org.example.bank.servlet**
  - Class name: **CustomerInfo**
  - Click **Finish**.
- Replace **CustomerInfo.java** from `C:\7501code\webservices` into the servlet.
- Study the servlet code. The `customerId` ssn or sin can be set in the parameter of the Web service call. The resulting `Customer` bean has either the ssn or sin attribute filled:

```
if (idType.equals("SSN"))
        customerId.setSsn(request.getParameter("id"));
else    customerId.setSin(request.getParameter("id"));
Customer customer = BankProxy.getCustomer(customerId);
```

Running the SDO Web service

To run the SDO Web service, right-click **CustomerInquiry.html** and select **Run As** → **Run on Server**. Select the customer ID type and click **Get Customer Info** (Figure 18-31).



ITSO SDO Web Service

Please select customer id type:

● Social Security Number

○ Social Insurance Number

Please enter the customer SSN or SIN: 111-11-1111

ITSO SDO Web Service Result

Customer's name is Mr. Henry Cui.
Customer's social security number is 111-11-1111.

ITSO SDO Web Service Result

Customer's name is Mr. Henry Cui.
Customer's social insurance number is 111-11-1111.

Figure 18-31 Running the SDO Web service

Tip: WSDL fault handling in SDO facade tooling

In the normal JAX-RPC case, a WSDL fault gets mapped to an application specific exception class listed in the throws clause of the business method. With **no data binding selected**, the WSDL fault gets mapped to nothing.

At runtime, the Web service engine converts the SOAP fault into a `SOAPFaultException` and throws it through the business method as an unchecked exception, meaning that there is no representation of the exception in the throws clause.

If you have to throw an application exception from the business method of a Java SDO facade skeleton generated with no data binding enabled, you have to write some code to assemble and throw a `SOAPFaultException` consisting of a `Detail` object with zero or more `DetailEntry` objects, each of which acts as the root of a document fragment corresponding to the global element referenced by the fault in the WSDL.

Looking ahead, JAX-WS 2.0, the successor to JAX-RPC, uses JAXB as the preferred binding between XML schema and Java. This replaces the relatively poor JAX-RPC data mapping. JAX-B promises mappings for all XML schemas. The Web Services Feature Pack supports JAX-WS 2.0. We will explore the rich features of Web Services Feature Pack at the end of this chapter.

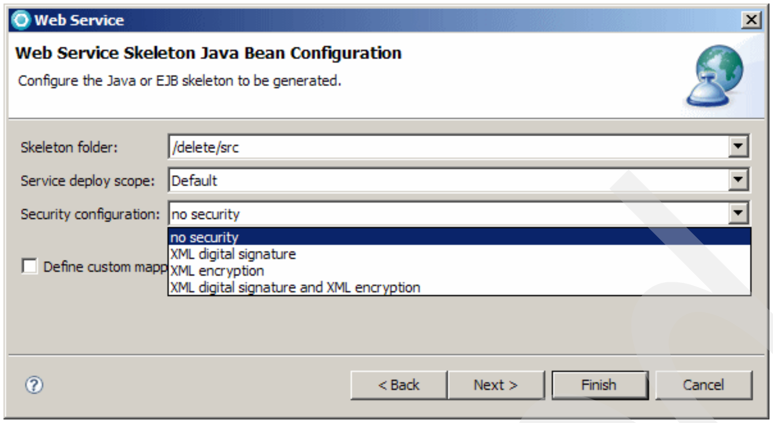
Web services security

Web services security for WebSphere Application Server v6.1 is based on standards included in the Organization for the Advancement of Structured Information Standards (OASIS) Web services security (WSS) Version 1.0 specification, the Username Token Version 1.0 profile, and the X.509 token Version 1.0 profile.

Application Developer provides three options to configure WS-Security:

1. Enabling Web services security in the **Web service creation wizards**.

   When creating a Web service with the Web Service wizard using the WebSphere run-time environments, you have four security options as shown in Figure 18-32.



Web Service Skeleton Java Bean Configuration

Configure the Java or EJB skeleton to be generated.

Skeleton folder: /delete/src

Service deploy scope: Default

Security configuration: no security

- no security
- XML digital signature
- XML encryption
- XML digital signature and XML encryption

□ Define custom mapp

Figure 18-32 Web Service security page in the Web Service wizard

- **None**: This will not enable security. This is the only WS-I compliant option.
- **XML Digital Signature**: This provides integrity against elements of a SOAP message. It allows you to digitally sign elements in the SOAP message protecting integrity.
- **XML Encryption**: This allows you to encrypt elements of a SOAP message and propagate the encrypted data.
- **XML Digital Signature and XML Encryption**: This combines the previous two options.

The security options available through the wizard are for **illustration** purposes only. They should **never** be used in a production environment. They point to the sample keystore and certificate files in WebSphere Application Server.

2. Enabling Web services security using the **WS Security wizard**.

   This is a new feature in Application developer v7. You can enable production-level security for your Web services using the WS Security wizard, which provides the following options:

   - Add **XML digital signature**
   - Add **XML encryption**
   - Add a stand alone **security token**

     A stand alone security token provides authentication for the Web service; it will validate the user, and determine whether a client is valid in a particular context.

- **Clone the security settings** of another Web service

  If you are creating multiple Web services and you want all of them to have the same security, you can choose to clone the security settings of a Web service and apply them to another Web service.

3. Enabling Web services security **manually**.

   If the WS Security wizard does not meet your needs, you have the option of manually securing Web services. This process gives you more control over the security settings, but is more time consuming and error prone than enabling security using the wizards. You can use the Web Services editor to configure server side security and the Web Services Client editor to configure the client side security. Or, you can deal with the deployment descriptor files directly, such as `ibm-webservices-bnd.xmi` and `ibm-webservices-ext.xmi` for the server side configuration, and `ibm-webservicesclient-bnd.xmi` and `ibm-webservicesclient-ext.xmi` for the client side configuration.

Using the WS Security wizard

In this section, we will use the new Web services security wizard to configure **XML encryption**.

Two key stores that contain the following keys are provided:

- **Server key store**—Key store used by the Web service provider, and holds:
  - Server public and private keys
  - Client certificate with only a public key

Key store path	C:\7501code\webservices\ServerKeyStore.jks
Key store password	henry
Key store type	JKS
Key alias	servercert
Key password	henry

- **Client key store**—Key store used by the Web service client, and holds:
  - Client public and private keys
  - Server certificate with only a public key

Key store path	C:\7501code\WebServices\ClientKeyStore.jks
Key store password	cui
Key store type	JKS
Key alias	clientcert
Key password	cui

When the Web service consumer sends a SOAP request to the Web service provider, it encrypts the message with the Web service provider's public key. When the Web service provider receives the encrypted message, it uses its own private key to decrypt the message.

When the Web service provider sends a SOAP response to the Web service consumer, it encrypts the message with the Web service consumer's public key. When the Web service consumer receives the encrypted message, it uses its own private key to decrypt the message.

Configure the server

To configure XML encryption for the server, do these steps:

- In the Project Explorer expand **JSR-109 Web Services** → **Services** → right-click **SimpleBankBeanService** → **Secure Web Service** → **Add XML Encryption** (Figure 18-33).

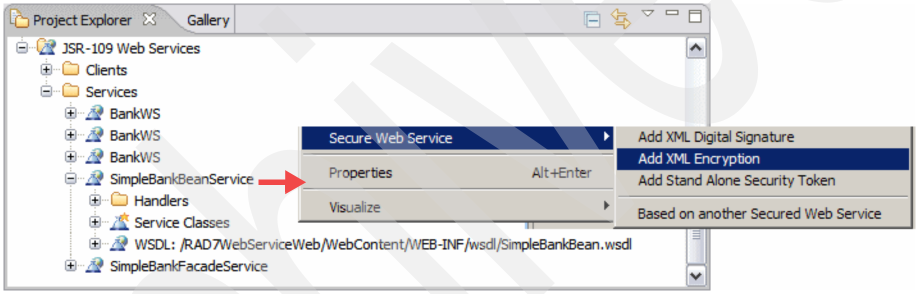
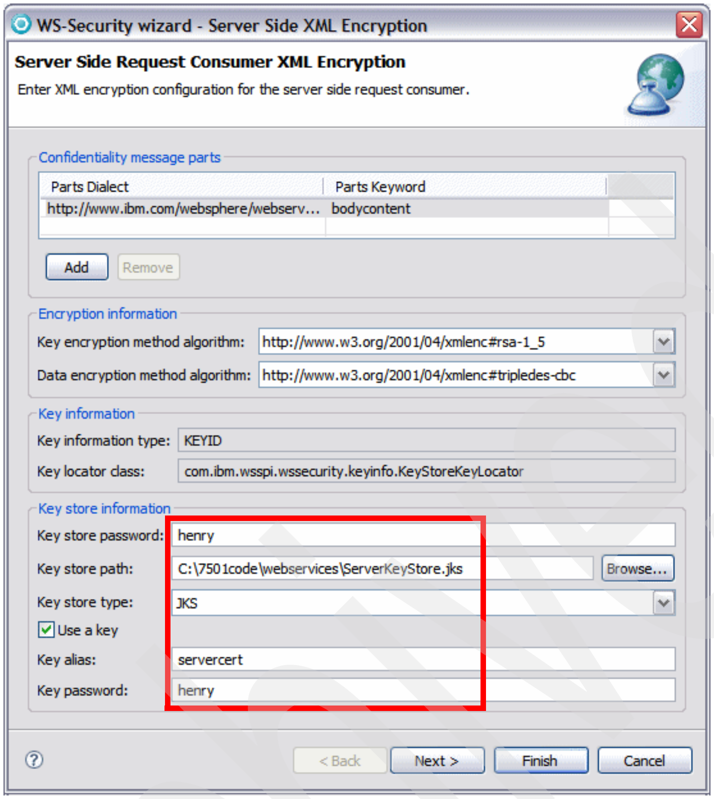


Figure 18-33 Start the WS Security wizard

- In the Server Side Request Consumer XML Encryption page (Figure 18-34), enter the following settings:
  - Key store password: **henry**
  - Key store path: `C:\7501code\webservices\ServerKeyStore.jks`
  - Key store type: **JKS**
  - Select **Use a key**
  - Key alias: **servercert**
  - Key password: **henry**
  - Click **Next**.



WS-Security wizard - Server Side XML Encryption

Server Side Request Consumer XML Encryption

Enter XML encryption configuration for the server side request consumer.

Confidentiality message parts

Parts Dialect	Parts Keyword
http://www.ibm.com/websphere/webserv...	bodycontent

Encryption information

Key encryption method algorithm: http://www.w3.org/2001/04/xmlenc#rsa-1_5

Data encryption method algorithm: http://www.w3.org/2001/04/xmlenc#tripledes-cbc

Key information

Key information type: KEYID

Key locator class: com.ibm.wsspi.wssecurity.keyinfo.KeyStoreKeyLocator

Key store information

Key store password: henry

Key store path: C:\7501code\webservices\ServerKeyStore.jks

Key store type: JKS

☑ Use a key

Key alias: servercert

Key password: henry

Figure 18-34 Server Side Request Consumer XML Encryption

- In the Server Side Response Generator XML Encryption page, enter the following settings:
  - Key store password: **henry**
  - Key store path: `C:\7501code\webservices\ServerKeyStore.jks`
  - Key store type: select **JKS** from the drop-down menu
  - Select **Use a key**
  - Key alias: **clientcert**
  - Key password: **cui**
- Click **Finish**.

Configure the client

To configure XML encryption for the client, we can create corresponding XML encryption based on the server configuration:

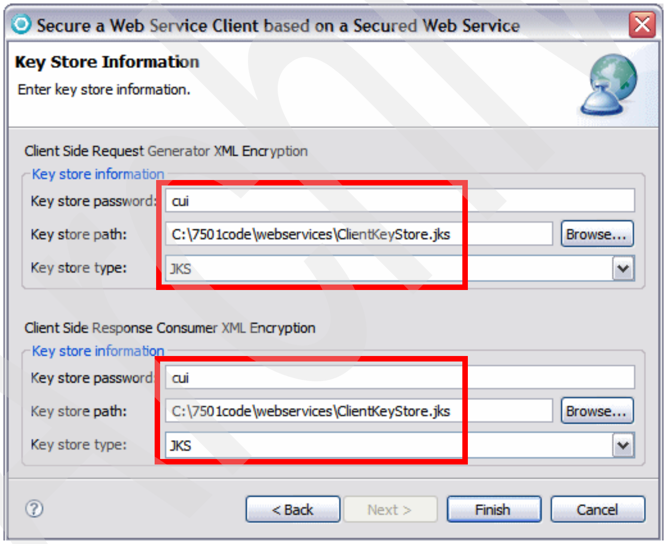
- Select **JSR-109 Web Services** → **Clients** → **RAD7WebServiceClient: service/SimpleBankBeanService** and **Secure Web Service Client** → **Based on a Secured Web Service**
- Select **RAD7WebServiceWeb: service/SimpleBankBeanService** (pre-selected) and click **Next**.
- In the Key Store Information dialog (Figure 18-35), type these values:

  Client Side Request Generator Encryption:

  – key store password: **cui**
  – key store path: `C:\7501code\webservices\ClientKeyStore.jks`
  – key store type: **JKS**

  Client Side Response Consumer XML Encryption:

  – key store password: **cui**
  – key store path: `C:\7501code\webservices\ClientKeyStore.jks`
  – key store type: **JKS**



Secure a Web Service Client based on a Secured Web Service

Key Store Information

Enter key store information.

Client Side Request Generator XML Encryption

Key store information

Key store password: cui

Key store path: C:\7501code\webservices\ClientKeyStore.jks

Key store type: JKS

Client Side Response Consumer XML Encryption

Key store information

Key store password: cui

Key store path: C:\7501code\webservices\ClientKeyStore.jks

Key store type: JKS

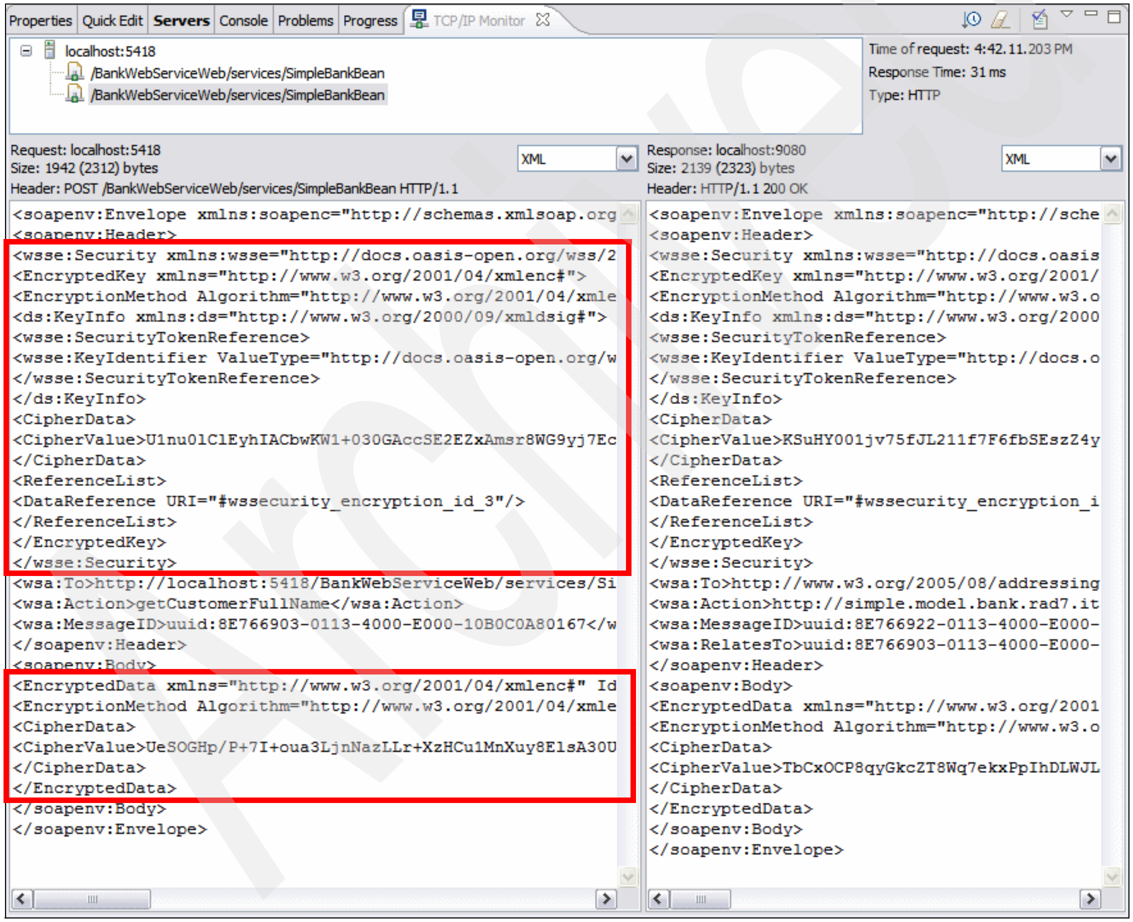
Figure 18-35 Client Side Key Store Information

- Click **Finish**.

Running the Web service with security

To run and monitor the Web service with security, do these steps:

- Select **RAD7WebServiceClient** → **WebContent** → **sampleSimpleBankBeanProxy** → **TestClient.jsp** and **Run As** → **Run on Server**.
- The JSP page opens in a browser. Select the **getCustomerFullName** method and then enter **111-11-1111**. Click **Invoke**. The correct result is displayed.
- Use the TCP/IP Monitor to see the encrypted message (Figure 18-36):



localhost:5418
/BankWebServiceWeb/services/SimpleBankBean
/BankWebServiceWeb/services/SimpleBankBean

Time of request: 4:42.11.203 PM
Response Time: 31 ms
Type: HTTP

Request: localhost:5418
Size: 1942 (2312) bytes
Header: POST /BankWebServiceWeb/services/SimpleBankBean HTTP/1.1

XML

```
<soapenv:Envelope xmlns:soapenc="http://schemas.xmlsoap.org
<soapenv:Header>
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2
<EncryptedKey xmlns="http://www.w3.org/2001/04/xmlenc#">
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmle
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000/09/xmldsig#">
<wsse:SecurityTokenReference>
<wsse:KeyIdentifier ValueType="http://docs.oasis-open.org/w
</wsse:SecurityTokenReference>
</ds:KeyInfo>
<CipherData>
<CipherValue>U1nu01C1EyhIACbwKW1+030GAccSE2EZxAmsr8WG9yj7Ec
</CipherData>
<ReferenceList>
<DataReference URI="#wssecurity_encryption_id_3"/>
</ReferenceList>
</EncryptedKey>
</wsse:Security>
<wsa:To>http://localhost:5418/BankWebServiceWeb/services/Si
<wsa:Action>getCustomerFullName</wsa:Action>
<wsa:MessageID>uuid:8E766903-0113-4000-E000-10B0C0A80167</w
</soapenv:Header>
<soapenv:Body>
<EncryptedData xmlns="http://www.w3.org/2001/04/xmlenc#" Id
<EncryptionMethod Algorithm="http://www.w3.org/2001/04/xmle
<CipherData>
<CipherValue>UeSOGHp/P+7I+oua3LjnNazLLr+XzHCu1MnXuy8E1sA30U
</CipherData>
</EncryptedData>
</soapenv:Body>
</soapenv:Envelope>
```

Response: localhost:9080
Size: 2139 (2323) bytes
Header: HTTP/1.1 200 OK

XML

```
<soapenv:Envelope xmlns:soapenc="http://sche
<soapenv:Header>
<wsse:Security xmlns:wsse="http://docs.oasis
<EncryptedKey xmlns="http://www.w3.org/2001/
<EncryptionMethod Algorithm="http://www.w3.o
<ds:KeyInfo xmlns:ds="http://www.w3.org/2000
<wsse:SecurityTokenReference>
<wsse:KeyIdentifier ValueType="http://docs.o
</wsse:SecurityTokenReference>
</ds:KeyInfo>
<CipherData>
<CipherValue>KSuHY001jv75fJL211f7F6fbSEszZ4y
</CipherData>
<ReferenceList>
<DataReference URI="#wssecurity_encryption_i
</ReferenceList>
</EncryptedKey>
</wsse:Security>
<wsa:To>http://www.w3.org/2005/08/addressing
<wsa:Action>http://simple.model.bank.rad7.it
<wsa:MessageID>uuid:8E766922-0113-4000-E000-
<wsa:RelatesTo>uuid:8E766903-0113-4000-E000-
</soapenv:Header>
<soapenv:Body>
<EncryptedData xmlns="http://www.w3.org/2001
<EncryptionMethod Algorithm="http://www.w3.o
<CipherData>
<CipherValue>TbCxOCP8qyGkcZT8Wq7ekxPpIhDLWJL
</CipherData>
</EncryptedData>
</soapenv:Body>
</soapenv:Envelope>
```

Figure 18-36 Using the TCP/IP Monitor to see the encrypted message

- Notice the `<wsse:Security>` and `<EncryptedData>` sections in the request and response messages.
- Remember how to activate the TCP/IP Monitor:
  - Start the TCP/IP Monitor (**Window** → **Preferences** → **Run/Debug**) and remember the port.
  - In the sample JSP client page, click **getEndpoint** method to see the current endpoint of the SOAP request:

    ```
    http://localhost:9080/BankWebServiceWeb/services/SimpleBankBean
    ```

  - We have to direct the SOAP request to the TCP/IP Monitor port. Select **setEndpoint** and set the endpoint to:

    ```
    http://localhost:xxxx/BankWebServiceWeb/services/SimpleBankBean
    ```

Explore the WS-Security configuration

You can see what has been generated by opening the WS-Security configuration:

- Open the **webservices.xml** file (`RAD7WebServiceWeb/WebContent/WEB-INF`) in the Web Services editor:
  - Select the **Extensions** page and expand Request Consumer Service Configuration Details → Required Confidentiality, and Response Generator Service Configuration Details → Confidentiality.
  - Select the **Binding Configurations** page and expand Request Consumer Binding Configuration Details → Key Locators, Key Information, Encryption Information, and Response Generator Binding Configuration Details → Key Locators, Key Information, Encryption Information.
- Open the **web.xml** file (`RAD7WebServiceClient/WebContent/WEB-INF`) in the Web Deployment Descriptor editor:
  - Select the **WS Extension** page and expand Request Generator Configuration → Confidentiality, and Response Consumer Configuration → Required Confidentiality.
  - Select the **WS Binding** page and expand Security Request Generator Binding Configuration → Key Locators, Key Information, Encryption Information, and Security Response Consumer Binding Configuration → Key Locators, Key Information, Encryption Information.

Developing Web services with the Web Services Feature Pack

On June 29, 2007, IBM officially released the IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services and Rational Application Developer v7.0.0.3. Rational Application Developer v7.0.0.3 contains new installable product features including **Web Services Feature Pack** and **IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services**.

- The Web Services Feature Pack provides features for assembling and deploying reliable, asynchronous, secure, and interoperable Java Web services for application components. The Web Services Feature Pack is designed for IBM WebSphere Application Server V6.1, extended with the WebSphere Application Server Feature Pack for Web Services. This feature supports the development of Web services from annotated Java or from Web Services Description Language (WSDL) using any XML schema.
- The IBM WebSphere Application Server, Version 6.1 Feature Pack for Web Services upgrades IBM WebSphere Application Server V6.1 to V6.1.0.9 and installs the runtime required to support the Feature Pack for Web Services.

  The WebSphere Application Server Version 6.1 Feature Pack for Web Services extends the capabilities of WebSphere Application Server V6.1 to enable Web services messages to be sent asynchronously, reliably, and securely, focusing on interoperability with other vendors and to provide support for the Java API for XML Web Services (JAX-WS) 2.0 programming model.
- The Feature Pack for Web Services introduces a set of Web services standards that support interoperable and reliable Web service applications. You can send messages asynchronously, which means that messages can communicate reliably even if one of the parties is temporarily offline, busy, or not available. You can send messages securely and rest assured that the messages are not vulnerable to attack. You can be confident that the communication is reliable and reaches its destination while interoperating with other vendors.

JAX-WS simplifies application development through a standard, annotation-based model to develop Web services applications and clients. A common set of binding rules for XML and Java objects make it easy to incorporate XML data and processing functions in Java applications; and a further set of enhancements help you send binary attachments, such as images or files, with the Web service request in an optimal way.

Simplified management using *Web services profiles* makes it easy to configure and reuse configurations, so that you can introduce new Web services profiles seamlessly in the future. These configurations are captured in a grouping called *policy sets*, which allows you to select and associate different qualities of service with an application. You can configure policy sets to allow only those capabilities within a given WS-Interoperability (WS-I) profile, thereby limiting and ensuring that the only configurable portions are those that are desired.

Supported standards and programming model

The Feature Pack for Web Services includes support for:

- New Web services standards, including:
  - Web Services Reliable Messaging (WS-RM)
  - Web Services Addressing (WS-Addressing)
  - SOAP Message Transmission Optimization Mechanism (MTOM)
- New standards-based programming model support:
  - Java API for XML Web Services (JAX-WS 2.0)
  - Java Architecture for XML Binding (JAXB 2.0)
  - SOAP 1.2
  - SOAP with Attachments API for Java (SAAJ 1.3)

In this section, we use the Web Services Feature Pack to create Web services and a Web services client using the same WSDL we created in “Creating a skeleton Web service” on page 817. We use the JAX-WS Web services runtime to generate Web services and secure the Web service client with the RAMP policy.

This section includes the following tasks:

- Installing the Web Services Feature Pack
- Enabling the WebSphere Application Server v6.1 test environment with the Feature Pack for Web Services
- Creating a Web service and a client using the JAX-WS runtime
- Securing the Web service and client with the RAMP policy set
- Sending binary data using MTOM
- Creating Web services using annotations

Installing the Web Services Feature Pack

If you are installing IBM Rational Application Developer, Version 7.0 for the first time, then you can install 7.0.0.3 update at the same time that you install IBM Rational Application Developer, Version 7.0 by clicking **Check for updates** on the Install page of the Installing Packages wizard in IBM Installation Manager.

You also have to select **Web Services Feature Pack** and **IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services** option in the Select the features you want to install page. Detailed steps can be found in Appendix A, “Product installation” on page 1281.

If IBM Rational Application Developer, Version 7.0, 7.0.0.1 or 7.0.0.2 is already installed on your system, then you can install the 7.0.0.3 update by using the **Update Packages wizard** in IBM Installation Manager.

Note that new installable product features, such as the Web Services Feature Pack, are not displayed and cannot be added while running the Update Packages wizard in Installation Manager. To install Web Services Feature Pack, run the **Modify Packages wizard** in Installation Manager after you have installed the fix pack and select **Web Services Feature Pack** and **IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services** option in the wizard to install these new features.

Enabling the WebSphere Application Server v6.1 test environment with the Feature Pack for Web Services

After installing the IBM WebSphere Application Server V6.1 Feature Pack for Web Services, the updated WebSphere Application Server v6.1 test environment is not visible unless you manually adopt a server profile that is enabled with the IBM WebSphere Application Server V6.1 Feature Pack for Web Services.

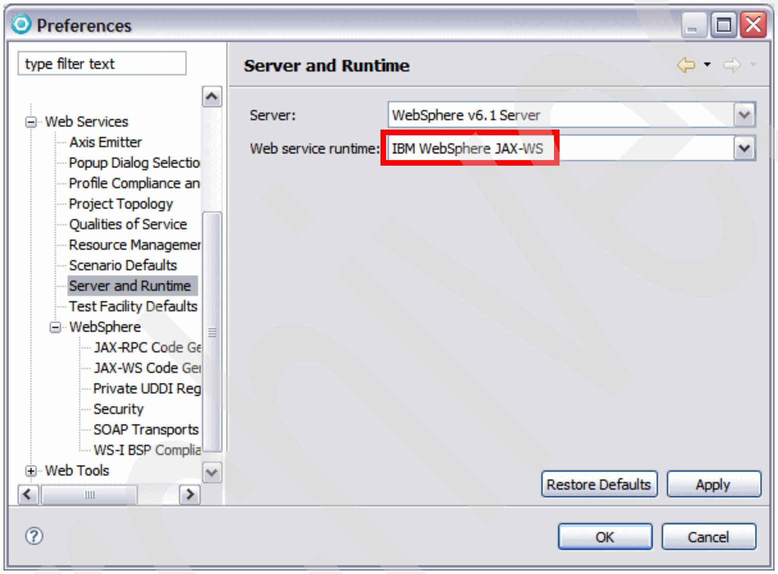
The installation of IBM WebSphere Application Server V6.1 Feature Pack for Web Services creates a new profile named **AppSrvWSFP01**, which is enabled for the feature pack. To use this profile, when you launch the Workbench for the first time after installing IBM WebSphere Application Server V6.1 Feature Pack for Web Services, do these steps:

- Create a new server definition that uses the AppSrvWSFP01 server profile. Right-click in the Servers view and select **New** → **Server.** Click **Next** twice, select **AppSrvWSFP01** for the WebSphere profile name and click **Finish**. We renamed the new server to WebSphere v6.1 Server WSFP for clarity.
- You could modify the existing default server to point to the AppSrvWSFP01 server profile, but this is not suggested (we do have the new profile anyway).

Creating a Web service and a client using the JAX-WS runtime

To create a Web service and Web service client using the JAX-WS Web services runtime, do these steps:

- Set JAX-WS as the default Web services runtime in the workspace:
  - Select **Window** → **Preferences** → **Web services** → **Server and Runtime**.
  - Select **IBM WebSphere JAX-WS** as the Web service runtime and click **OK** (Figure 18-37).

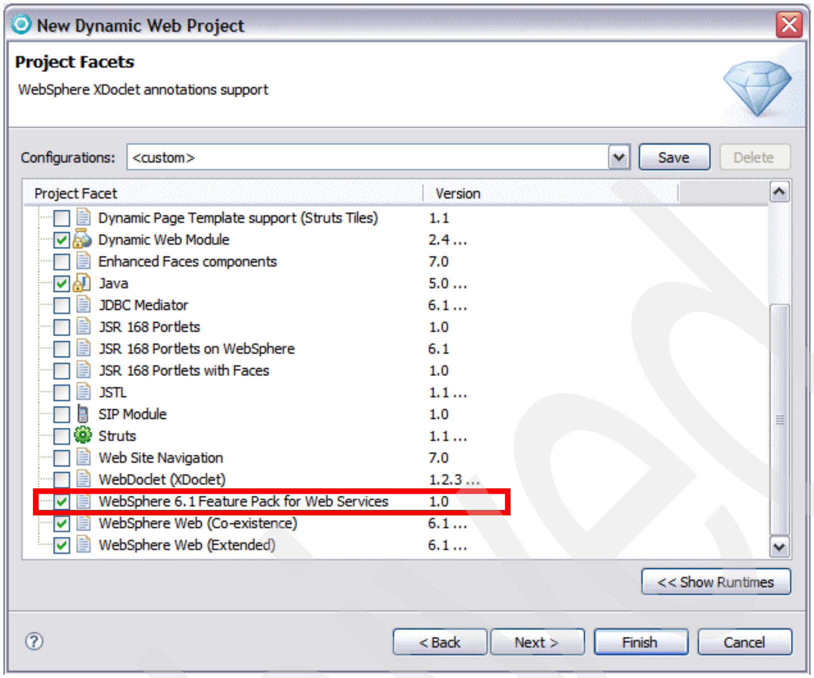


Server: WebSphere v6.1 Server

Web service runtime: IBM WebSphere JAX-WS

Figure 18-37 Set JAX-WS as the default Web service runtime

- Create a dynamic Web project for the Web service:
  - Project Name: **RAD7WSFP**
  - Target runtime: WebSphere Application Server 6.1 (default)
  - EAR project: **RAD7WSFPEAR**
  - Click **Next**.
  - On the Project Facets page, select **WebSphere 6.1 Feature Pack for Web Services 1.0** and click **Finish** (Figure 18-38).



Project Facets

WebSphere XDoclet annotations support

Configurations: <custom>

Project Facet	Version
☐ Dynamic Page Template support (Struts Tiles)	1.1
☑ Dynamic Web Module	2.4 ...
☐ Enhanced Faces components	7.0
☑ Java	5.0 ...
☐ JDBC Mediator	6.1 ...
☐ JSR 168 Portlets	1.0
☐ JSR 168 Portlets on WebSphere	6.1
☐ JSR 168 Portlets with Faces	1.0
☐ JSTL	1.1 ...
☐ SIP Module	1.0
☐ Struts	1.1 ...
☐ Web Site Navigation	7.0
☐ WebDoclet (XDoclet)	1.2.3 ...
☑ WebSphere 6.1 Feature Pack for Web Services	1.0
☑ WebSphere Web (Co-existence)	6.1 ...
☑ WebSphere Web (Extended)	6.1 ...

Figure 18-38 Add Web Services Feature Pack facets to the project

- Create a dynamic Web project for the Web service client:
  - Project Name: **RAD7WSFPClient**
  - EAR project: **RAD7WSFPEAR** as the EAR project name.
  - Click **Next**.

> **Note:** Note that in JAX-RPC Web services it is suggested that you generate the service and client code into different EARs due to file name conflicts. JAX-WS Web services do not have this problem, so the service and client can safely share the same EAR.

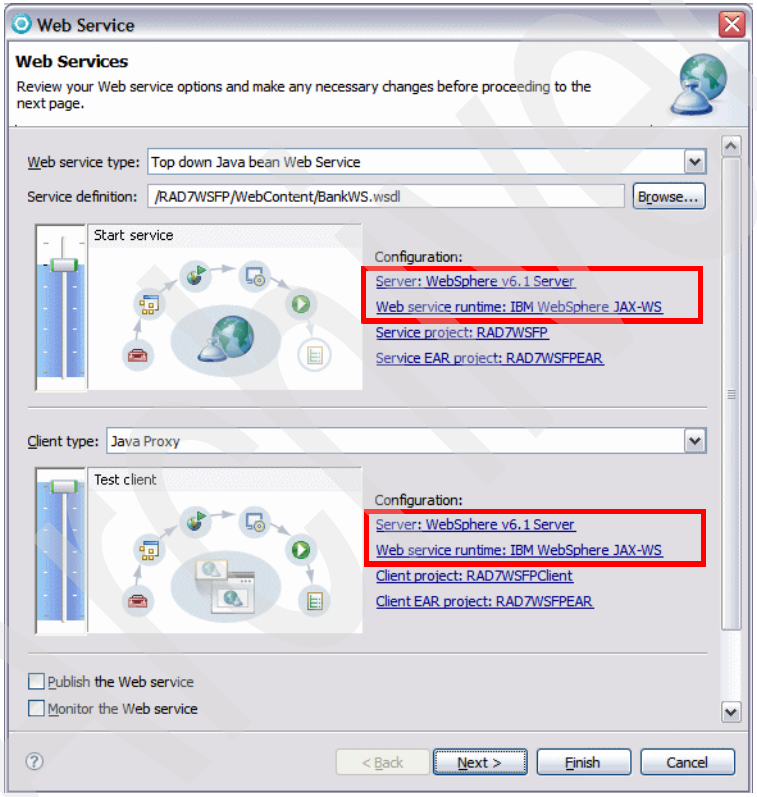
  - On the Project Facets page, select **WebSphere 6.1 Feature Pack for Web Services 1.0** and click **Finish**.
- Import the **`BankWS.wsdl`** from `C:\7501code\webservices` into the `RAD7WSFP/WebContent` folder.
- Start WebSphere Application Server v6.1. Make sure the server uses the Web Services Feature Pack enabled profile.

Note: If the server profile is not associated with the Web Services Feature Pack, you will get a **Publishing failed** error when you run the Web services wizard with the JAX-WS runtime.

Creating the service and client

Create the Web service and client using the Web Service wizard:

- Right-click **BankWS.wsdl** → **Web Services** → **Generate Java bean skeleton**.
- The Web Service wizard starts with the Web Services page (Figure 18-39).



Web Service

Web Services

Review your Web service options and make any necessary changes before proceeding to the next page.

Web service type: Top down Java bean Web Service

Service definition: /RAD7WSFP/WebContent/BankWS.wsdl

Start service

Configuration:
Server: WebSphere v6.1 Server
Web service runtime: IBM WebSphere JAX-WS
Service project: RAD7WSFP
Service EAR project: RAD7WSFPEAR

Client type: Java Proxy

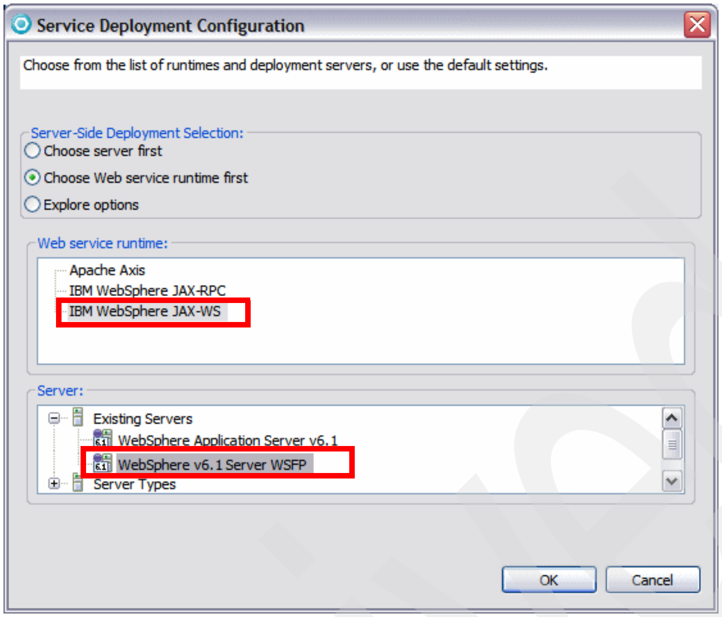
Test client

Configuration:
Server: WebSphere v6.1 Server
Web service runtime: IBM WebSphere JAX-WS
Client project: RAD7WSFPClient
Client EAR project: RAD7WSFPEAR

□ Publish the Web service
□ Monitor the Web service

Figure 18-39 Web Services page

- Select the following options for the Web service:
  - Keep the slider at **Start service**.
  - Click **Server: WebSphere v6.1 Server** to open the Service Deployment Configuration (Figure 18-40).



Service Deployment Configuration

Choose from the list of runtimes and deployment servers, or use the default settings.

Server-Side Deployment Selection:

○ Choose server first

● Choose Web service runtime first

○ Explore options

Web service runtime:

- Apache Axis
- IBM WebSphere JAX-RPC
- IBM WebSphere JAX-WS

Server:

- Existing Servers
  - WebSphere Application Server v6.1
  - WebSphere v6.1 Server WSFP
- Server Types

Figure 18-40 Service Deployment Configuration

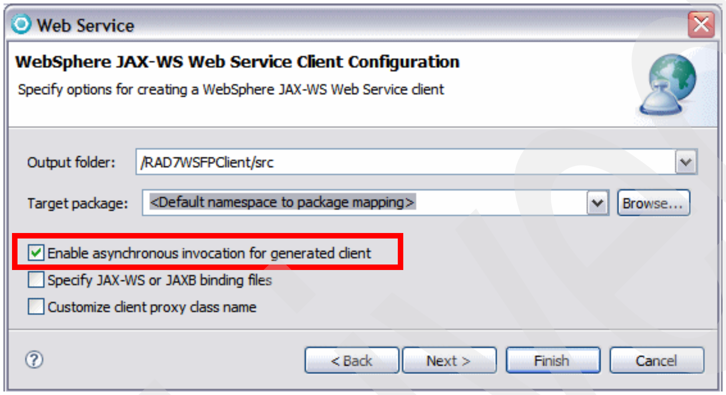
- Select **Choose Web service runtime first**.
- Select **IBM WebSphere JAX-WS** as Web service runtime.
- Expand **Existing Servers** and select the server with the profile associated with Web Services Feature Pack. In Figure 18-40 there are two servers:
  - WebSphere Application Server v6.1 is associated with a profile `AppSrv01`, which is not enabled with Web Services Feature Pack.
  - WebSphere v6.1 Server WSFP is associated with `AppSrvWSFP01`, which is the correct selection in this scenario.
  - Click **OK**.
- Service project: **RAD7WSFP**
- Service EAR project: **RAD7WSFPEAR**

► Select the following options for the Web service client:

- Move the slider to **Test client**.
- Server: **WebSphere v6.1 Server**.
- Web service runtime: **IBM WebSphere JAX-WS**
- Client project: **RAD7WSFPClient**
- Client EAR project: **RAD7WSFPEAR**

► Click **Next**.

- In the WebSphere JAX-WS Top Down Web Service Configuration page, click **Next**. If the server is not started yet, you will be directed to a **Start server** page. Click **Start server**. Once the server is started, click **Next**.
- In the Test Web Service page, do not launch the Web Services Explorer, and click **Next**.
- In the WebSphere JAX-WS Web Service Client Configuration page, select **Enable asynchronous invocation for generated client** (Figure 18-41).



WebSphere JAX-WS Web Service Client Configuration

Specify options for creating a WebSphere JAX-WS Web Service client

Output folder: /RAD7WSFPClient/src

Target package: <Default namespace to package mapping>

☑ Enable asynchronous invocation for generated client

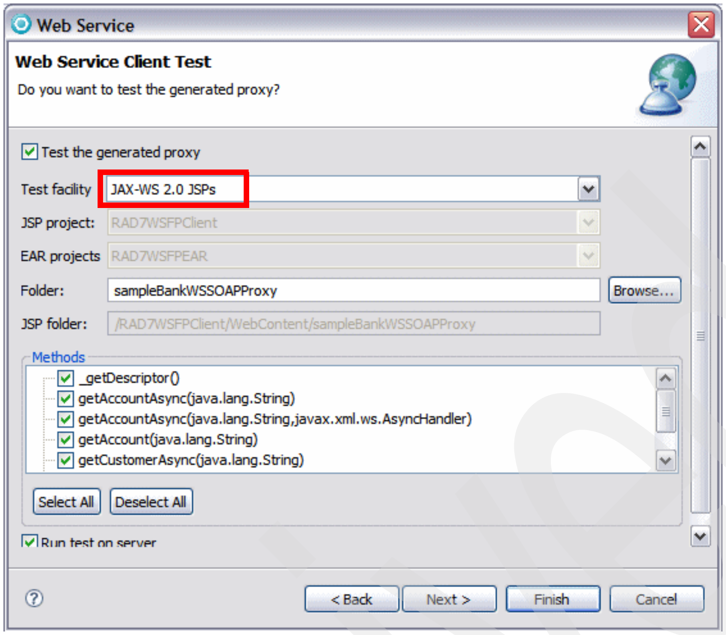
□ Specify JAX-WS or JAXB binding files

□ Customize client proxy class name

Figure 18-41 WebSphere JAX-WS Web Service Client Configuration

When you select to enable an asynchronous client, for each method in the Web service, two additional methods are created. These are polling and callback methods, which allow the client to function asynchronously. Click **Next**.

- In the Web Service Client Test page, select **JAX-WS 2.0 JSPs** as the Test Facility. Leave all the methods selected. Click **Finish** (Figure 18-42).



Web Service

Web Service Client Test

Do you want to test the generated proxy?

☑ Test the generated proxy

Test facility: JAX-WS 2.0 JSPs

JSP project: RAD7WSFPClient

EAR projects: RAD7WSFPEAR

Folder: sampleBankWSSOAPProxy

JSP folder: /RAD7WSFPClient/WebContent/sampleBankWSSOAPProxy

Methods

☑ _getDescriptor()
☑ getAccountAsync(java.lang.String)
☑ getAccountAsync(java.lang.String,javax.xml.ws.AsyncHandler)
☑ getAccount(java.lang.String)
☑ getCustomerAsync(java.lang.String)

☑ Run test on server

Figure 18-42 Select the test facility for the JAX-WS client

Implement the JavaBean skeleton

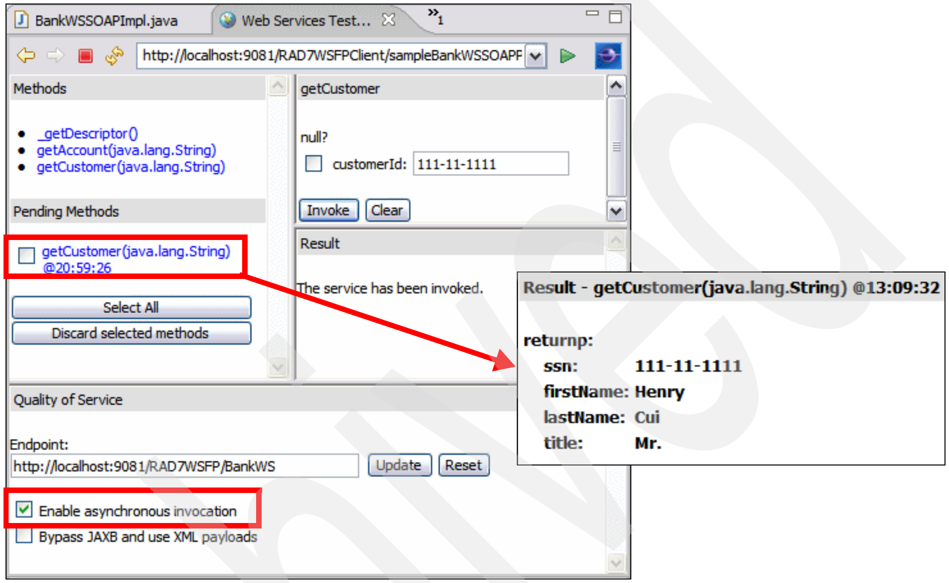
After code generation, the generated skeleton **BankWSSOAPImpl.java** is open in the editor. You might notice the Java annotations in this class (Example 18-4). The annotation **@javax.jws.WebService** tells the server runtime environment to expose all public methods on that bean as a Web service.

Example 18-4 Java annotations in JAX-WS Web services

```
@javax.jws.WebService (endpointInterface="org.example.bankws.BankWS",
                       targetNamespace="http://www.example.org/BankWS/",
                       serviceName="BankWS", portName="BankWSSOAP")
```

- Copy/paste the code shown in Example 18-2 on page 848 as the implementation of the skeleton Java Bean. Only copy the lines in bold, do not add the exceptions.
- The sample JSP client is also open in a browser. To test the service *synchronously*, select **getCustomer** method and enter **111-11-1111** as the customer ID, and click **Invoke**.
- To test the Web service *asynchronously*, in the Quality of Service pane select **Enable asynchronous invocation**.

- Select the **getCustomer** method and enter **111-11-1111** as the customer ID, and click **Invoke**. The Result pane displays `The service has been invoked.`
- A new link is display under Pending Methods, indicating that the method is in progress (Figure 18-43). Click the link to display the method response in the Results pane.



- _getDescriptor()
- getAccount(java.lang.String)
- getCustomer(java.lang.String)

Pending Methods

□ getCustomer(java.lang.String) @20:59:26

getCustomer

null?

□ customerId: 111-11-1111

Result

The service has been invoked.

Result - getCustomer(java.lang.String) @13:09:32

returnp:

ssn:	111-11-1111
firstName:	Henry
lastName:	Cui
title:	Mr.

Quality of Service

Endpoint: http://localhost:9081/RAD7WSFP/BankWS

☑ Enable asynchronous invocation

□ Bypass JAXB and use XML payloads

Figure 18-43 JAX-WS client asynchronous invocation

Securing the Web service and client with the RAMP policy set

You can use policy sets to simplify configuring the qualities of service for Web services and clients. Policy sets are assertions about how Web services are defined. Using policy sets, you can combine configurations for different policies. You can use policy sets with JAX-WS applications, but not with JAX-RPC applications.

A policy set is identified by a unique name. An instance of a policy set consists of a collection of policy types. An empty policy set has no policy instance defined in it.

You can use the policy sets that are included with Application Developer to simplify configuring the qualities of service for your Web services and clients. For example, the **Reliable Asynchronous Messaging Profile** (RAMP) default policy set consists of instances of the WS-Security, WS-Addressing, and WS-ReliableMessaging policy types.

Policy sets omit application or user-specific information, such as keys for signing, key store information, or persistent store information. Instead, application and user-specific information is defined in the bindings. Typically, bindings are specific to the application or the user, and bindings are not normally shared. On the server side, if you do not specify a binding for a policy set, a default binding will be used for that policy set. On the client side, you must specify a binding for each policy set.

A policy set attachment defines which policy set is attached to service resources, and which bindings are used for the attachment. The bindings define how the policy set is attached to the resources. An attachment is defined outside of the policy set, as meta data associated with the application. To enable a policy set to work with an application, a binding is required. Use the policy set attachment wizards to configure bindings.

In this section, we attach the Reliable Asynchronous Messaging Profile (RAMP) default policy set to the Web service and client. This policy set provides the following features:

- **Reliable message delivery** to the intended receiver, by enabling WS-ReliableMessaging
- **Message integrity** by digital signature that includes signing the body, timestamp, WS-Addressing headers, and WS-ReliableMessaging headers using the WS-SecureConversation and WS-Security specifications
- **Confidentiality** by encryption that includes encrypting the body, signature and signature confirmation elements, using the WS-SecureConversation and WS-Security specifications

Attach the RAMP policy set to the Web service

To attach the RAMP policy set to the Web service, do these steps:

- In the Project Explorer expand **JAX-WS Web Services** → **Services** → **RAD7WSFP: {http://www.example.org/BankWS/}BankWS**, and select **Manage Policy Set Attachment** (Figure 18-44).

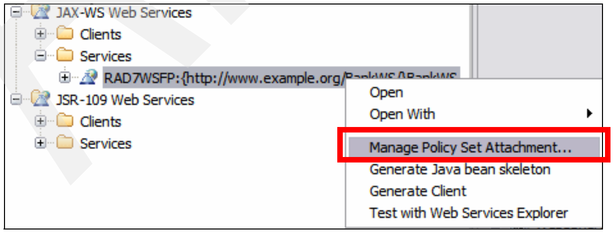
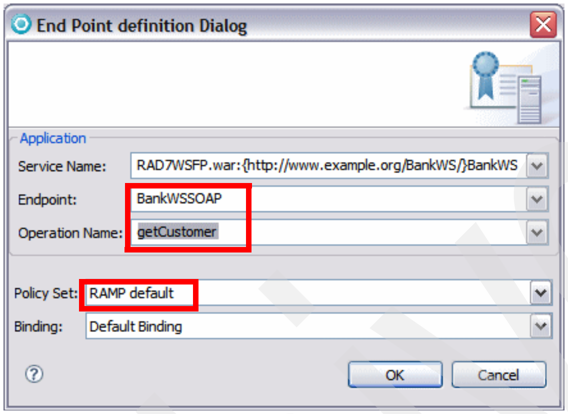

JAX-WS Web Services
Clients
Services
RAD7WSFP:{http://www.example.org/BankWS/}BankWS
JSR-109 Web Services
Clients
Services
Open
Open With
Manage Policy Set Attachment...
Generate Java bean skeleton
Generate Client
Test with Web Services Explorer

Figure 18-44 Start Manage Policy Set Attachment wizard

- Select the **RAD7WSFPEAR** as the service EAR project and click **Add**.
- You can apply a policy set at the service, port, or operation level. Different policy sets can be applied to various endpoints and operations within a single Web service. However, the service and client must have the same policy set settings. Select **BankWSSOAP** as the Endpoint and **getCustomer** as the Operation Name. From the Policy Set drop-down list, select **RAMP default**, Click **OK** (Figure 18-45).



Service Name: RAD7WSFP.war:{http://www.example.org/BankWS/}BankWS
Endpoint: BankWSSOAP
Operation Name: getCustomer
Policy Set: RAMP default
Binding: Default Binding

Figure 18-45 End point definition

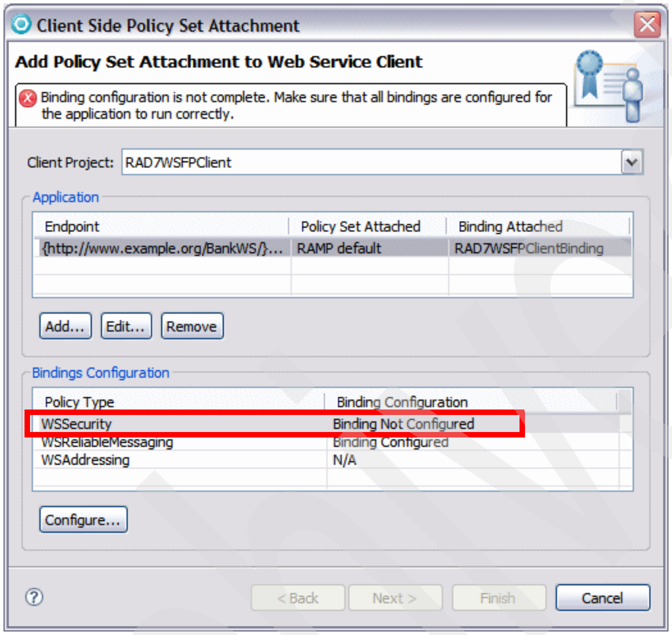
- The service is now listed in the Application table and the RAMP default policy. Click **Finish**.

Secure the Web service client with the RAMP policy set

To secure the Web service client with the RAMP policy set, do these steps:

- Select **JAX-WS Web Services** → **Clients** → **RAD7WSFPClient: {http://www.example.org/BankWS/}BankWS**, and select **Manage Policy Set Attachment**.
- In the Application section, click **Add** to attach a policy set to the endpoint and specify the bindings.
- In the Endpoint Definition dialog, do these steps:
  - Select **BankWSSOAP** as the Endpoint and **getCustomer** as the Operation Name.
  - From the Policy Set drop-down list, select **RAMP default**.
  - In the Binding field, enter **RAD7WSFPClientBinding** as the name for the binding that you want to associate with the attachment. Because the information that a binding contains is unique to a given environment or platform, you can use each binding with only one attachment. Click **OK**.

- The policy types contained by the policy set you selected are listed in the Bindings Configuration table. Any of these policies that require additional configuration information are marked with `Binding Not Configured`. For the RAMP default policy set, the WS-Addressing policy does not have to be configured, while the WS-Security policy does (Figure 18-46).



Client Side Policy Set Attachment

Add Policy Set Attachment to Web Service Client

Binding configuration is not complete. Make sure that all bindings are configured for the application to run correctly.

Client Project: RAD7WSFPClient

Application

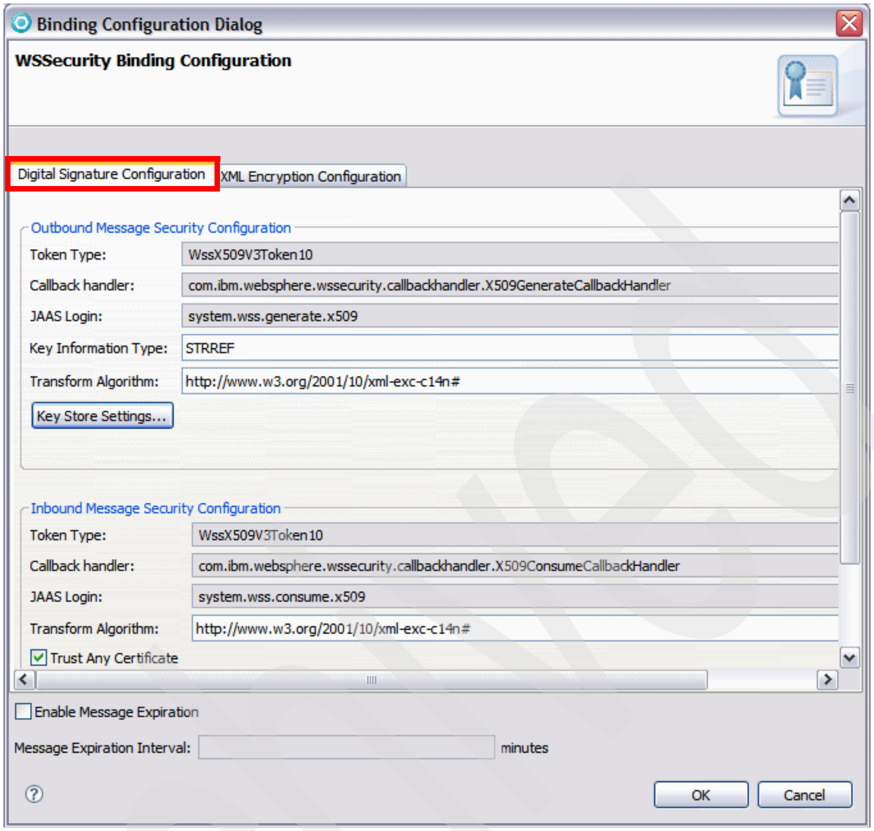
Endpoint	Policy Set Attached	Binding Attached
{http://www.example.org/BankWS/}...	RAMP default	RAD7WSFPClientBinding

Bindings Configuration

Policy Type	Binding Configuration
WSSecurity	Binding Not Configured
WSReliableMessaging	Binding Configured
WSAddressing	N/A

Figure 18-46 Client side policy set attachment

- Select the **WSSecurity policy** and click **Configure**. The WSSecurity Binding Configuration page is displayed (Figure 18-47).



Binding Configuration Dialog

WSSecurity Binding Configuration

Digital Signature Configuration | XML Encryption Configuration

Outbound Message Security Configuration

Token Type: WssX509V3Token10

Callback handler: com.ibm.websphere.wssecurity.callbackhandler.X509GenerateCallbackHandler

JAAS Login: system.wss.generate.x509

Key Information Type: STRREF

Transform Algorithm: http://www.w3.org/2001/10/xml-exc-c14n#

Key Store Settings...

Inbound Message Security Configuration

Token Type: WssX509V3Token10

Callback handler: com.ibm.websphere.wssecurity.callbackhandler.X509ConsumeCallbackHandler

JAAS Login: system.wss.consume.x509

Transform Algorithm: http://www.w3.org/2001/10/xml-exc-c14n#

☑ Trust Any Certificate

□ Enable Message Expiration

Message Expiration Interval: minutes

OK Cancel

Figure 18-47 WSSecurity Binding Configuration

- On the Digital Signature Configuration tab, the outbound message security configuration is for the Web service client request to the server. Click **Key Store Settings**:
  - Keystore Path: **<RAD_HOME>\runtimes\base_v61\etc\ws-security\samples\dsig-sender.ks**

    You have to replace <RAD_HOME> with the actual Application Developer installation root directory:
  - Keystore Password: **client**
  - Keystore Type: **JKS**
  - Key Alias: **soaprequester**
  - Key Password: **client**
  - Click **OK.**

- Select the **XML Encryption Configuration** tab. For Outbound Message Security Configuration, click **Key Store Settings**:
  - Keystore Path: **<RAD_HOME>\runtimes\base_v61\etc\ws-security\samples\enc-sender.jceks**
  - Keystore Password: **storepass**
  - Keystore Type: **JCEKS**
  - Key Alias: **bob**
  - Click **OK**.
- On the XML Encryption Configuration tab, for Inbound Message Security Configuration, click **Key Store Settings**:
  - Keystore Path: **<RAD_HOME>\runtimes\base_v61\etc\ws-security\samples\enc-sender.jceks**
  - Keystore Password: **storepass**
  - Keystore Type: **JCEKS**
  - Key Alias: **alice**
  - Key Password: **keypass**
  - Click **OK**.
- Once you have added the required key store settings XML encryption, click **OK**.
- Click **Finish** to complete the wizard.

Note: This example uses the sample keystore and certificate files in WebSphere Application Server v6.1. The policy set configuration wizard in Application Developer v7.0.0.3 Web Services Feature Pack does not support Web service side custom binding configuration. This could be done through the use of the WebSphere Administrative Console. The support for custom binding configuration from the tools will be available in future version of Application Developer.

Test the secured Web service

To test the secured Web service, do these steps:

- In the `RAD7WSFPClient` project, right-click **TestClient.jsp** → **Run As** → **Run on Server**. Select the WSFP server and click **Finish**.
- Select **Window** → **Preferences** → **Run/Debug** → **TCP/IP Monitor**. Make sure the TCP/IP Monitor listens to the correct port and direct the SOAP request to the correct endpoint. Because we have switched the profile to the Web Services Feature Pack enabled profile, we have to change the port to the correct port that the new profile is running on. For example, change the port from 9080 to 9081. Alternatively, you can click **Add** and create another TCP/IP Monitor with the correct port settings.

- Start the TCP/IP Monitor.
- In the sample JSP client, Quality of Service pane, change the endpoint to the correct TCP/IP port and click **Update**:

```
http://localhost:xxxx/RAD7WSFP/BankWS
```

- Invoke the **getAccount** method with an account number of `001-999000777` and verify that the message is in clear (unencrypted) text (we applied the RAMP policy to the `getCustomer` method only).
- Invoke the **getCustomer** method with a customer number of `111-11-1111` and verify that the message is signed and encrypted.
- Stop the server or continue with the MTOM example.

Sending binary data using MTOM

SOAP Message Transmission Optimization Mechanism (MTOM) is a standard that is developed by the World Wide Web Consortium (W3C). MTOM describes a mechanism for optimizing the transmission or wire format of a SOAP message by selectively re-encoding portions of the message while still presenting an XML Information Set (Infoset) to the SOAP application.

MTOM uses the XML-binary Optimized Packaging (XOP) in the context of SOAP and MIME over HTTP. XOP defines a serialization mechanism for the XML Infoset with binary content that is not only applicable to SOAP and MIME packaging, but to any XML Infoset and any packaging mechanism. It is an alternate serialization of XML that just happens to look like a MIME multipart or related package, with XML documents as the root part.

That root part is very similar to the XML serialization of the document, except that base64-encoded data is replaced by a reference to one of the MIME parts, which is not base64 encoded. This reference enables you to avoid the bulk and overhead in processing that is associated with encoding. Encoding is the only way binary data can work directly with XML.

In this section we use the top-down approach to create a JAX-WS Web service to send binary attachments along with SOAP request, and receive binary attachments along with SOAP response using MTOM.

The Web service client sends three types of document: Microsoft Word, image, and PDF file. We describe several ways to send the documents:

- The client uses **byte[]** to send the Word document.
- The client uses **java.awt.Image** to send the image file.
- The client uses **javax.activation.DataHandler** to send the PDF file.

Once the Web service receives the binary data from the client, it stores the received document on the local hard disk and then passes the same document back to the client. In a real world scenario, the provider or the consumer can just send an acknowledgement message, once it receives the binary data from the other side. For our example, we want to show how to enable the MTOM on both the client and the server side in a compact example.

Create a Web service project and import the WSDL

To create a Web service project, do these steps:

- Select **File** → **New** → **Dynamic Web Project**.
  - Project Name: **RAD7WSMTOM**
  - Target runtime: **WebSphere Application Server 6.1**.
  - EAR Project Name: **RAD7WSMTOMEAR**
  - Click **Next**.
- On the Project Facets page, select **WebSphere 6.1 Feature Pack for Web Services 1.0** and click **Finish**.
- Select **File** → **Import** → **General** → **File System**, click **Browse** to locate the `C:\7501code\webservices` folder, and select **`ProcessDocumentService.wsdl`**. Select the **`RAD7WSMTOM/WebContent`** folder as the Into folder and click **Finish**.

Open the `ProcessDocumentService.wsdl` and take a look at the source. You will see some interesting attributes, highlighted in Example 18-5.

Example 18-5 WSDL snippet in ProcessDocument.wsdl

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
            xmlns:xmime="http://www.w3.org/2005/05/xmlmime"
            xmlns:tns="http://mtom.rad7.ibm.com/"
            targetNamespace="http://mtom.rad7.ibm.com/" version="1.0">
<xs:complexType name="sendPDFFile">
   <xs:sequence>
      <xs:element minOccurs="0" name="arg0" type="xs:base64Binary"
                                          xmime:expectedContentTypes="*/*"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="sendWordFile">
   <xs:sequence>
      <xs:element minOccurs="0" name="arg0" type="xs:base64Binary"/>
   </xs:sequence>
</xs:complexType>
<xs:complexType name="sendImage">
   <xs:sequence>
      <xs:element minOccurs="0" name="arg0" type="xs:base64Binary"
                                    xmime:expectedContentTypes="image/jpeg"/>
   </xs:sequence>
</xs:complexType>
```

The default mapping for `xs:base64Binary` is `byte[]` in Java. If you want to use a different mapping, you can add the `xmime:expectedContentTypes` attribute to the element containing the binary data. This attribute is defined in the `http://www.w3.org/2005/05/xmlmime` namespace and specifies the MIME types that the element is expected to contain. The setting of this attribute changes how the code generators create the JAXB class for the data. Depending on the `expectedContentTypes` value contained in the WSDL file, the JAXB artifacts generated are in the Java type as described in Table 18-1.

Table 18-1 Mapping between MIME type and Java type

MIME type	Java type
`image/gif`	`java.awt.Image`
`image/jpeg`	`java.awt.Image`
`text/plain`	`java.lang.String`
`text/xml`	`javax.xml.transform.Source`
`application/xml`	`javax.xml.transform.Source`
`*/*`	`javax.activation.DataHandler`

Based on this table we can predict that:

- `sendWordFile` will be mapped to `byte[]` in Java
- `sendPDFFile` will be mapped to `javax.activation.DataHandler`
- `sendImage` will be mapped to `java.awt.Image`.

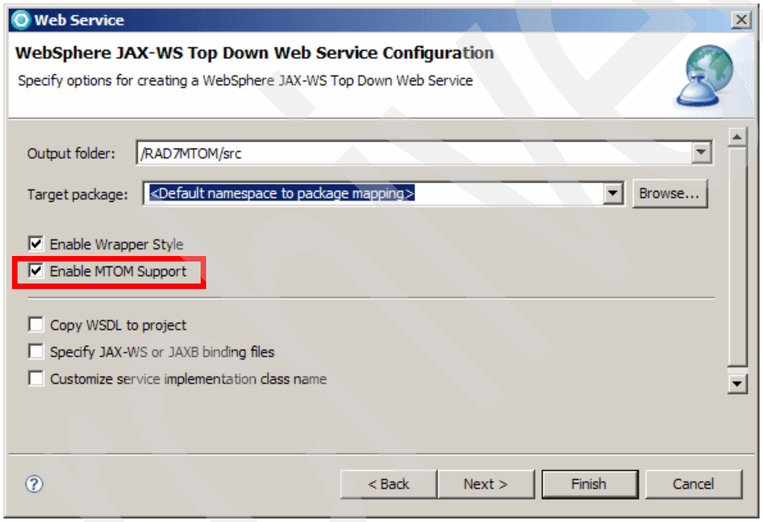
Generate the Web service and client

To create the Web service and client using the Web Service wizard, do these steps:

- Start the WebSphere v6.1 Server WSFP (if not running).
- Right-click **ProcessDocumentService.wsdl** → **Web Services** → **Generate Java bean skeleton**. The Web Service wizard starts with the Web Services page.
- Select the following options for the Web service:
  - Server: **WebSphere v6.1 Server**.
  - Select **IBM WebSphere JAX-WS** as Web service runtime.
  - Service project: **RAD7WSMTOM**
  - Service EAR project: **RAD7WSMTOMEAR**

- Select the following options for the Web service client:
  - Move the slider to **Test client**.
  - Server: **WebSphere v6.1 Server**.
  - Web service runtime: **IBM WebSphere JAX-WS**
  - Client project: **RAD7WSMTOMClient**
  - Client EAR project: **RAD7WSMTOMClientEAR**

  Because the Web service client project is not yet in the workspace when we run the Web service wizard, the wizard creates the project and enables the WebSphere 6.1 Feature Pack for Web Services facet.
- Select **Monitor the Web service** and then click **Next**.
- In the WebSphere JAX-WS Top Down Web Service Configuration page, select **Enable MTOM Support** and click **Next** (Figure 18-48).



Web Service

WebSphere JAX-WS Top Down Web Service Configuration

Specify options for creating a WebSphere JAX-WS Top Down Web Service

Output folder: /RAD7MTOM/src

Target package: <Default namespace to package mapping>

☑ Enable Wrapper Style

☑ Enable MTOM Support

☐ Copy WSDL to project

☐ Specify JAX-WS or JAXB binding files

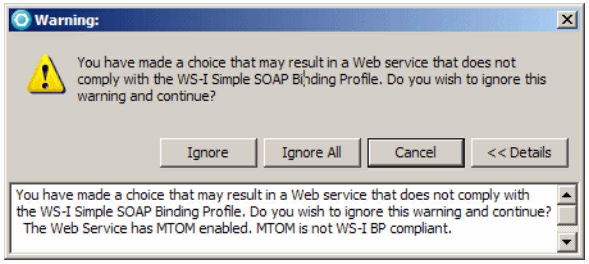
☐ Customize service implementation class name

Figure 18-48 WebSphere JAX-WS Bottom Up Web Service Configuration

- A warning pops up. Click **Details** to view the complete message (Figure 18-49.)

  MTOM support is not included in WS-I Basic Profile version 1.1. However, it is one of the main new features in WS-I Basic Profile version 1.2, which is not finalized yet.

  Click **Ignore** to continue the code generation.



You have made a choice that may result in a Web service that does not comply with the WS-I Simple SOAP Binding Profile. Do you wish to ignore this warning and continue?

You have made a choice that may result in a Web service that does not comply with the WS-I Simple SOAP Binding Profile. Do you wish to ignore this warning and continue?
The Web Service has MTOM enabled. MTOM is not WS-I BP compliant.

Figure 18-49 WS-I warning against MTOM

- In the Test Web Service page click **Next**.
- In the WebSphere JAX-WS Web Service Client Configuration page accept the default, and click **Next**.
- In the Web Service Client Test page, select **JAX-WS 2.0 JSPs** as the Test Facility, and click **Finish**. The generated JavaBean skeleton is opened, as well as the sample JSP client.

Implement the JavaBean skeleton

Before we test the sample JSP client, we have to implement the generated Java Bean skeleton. The Web services stores the received document on the local hard drive and then passes the same document back to the client. Do these steps:

- Examine the generated skeleton. We can see that `sendWordFile` is mapped to `byte[]`, `sendPDFFile` is mapped to `javax.activation.DataHandler`, and `sendImage` is mapped to `java.awt.Image`, as we expected.
- Copy/paste the code into `ProcessDocumentPortImpl.java` and then save the file (Example 18-6). The code can be found in `C:\7501code\webservices`.

Example 18-6 ProcessDocumentPortImpl.java

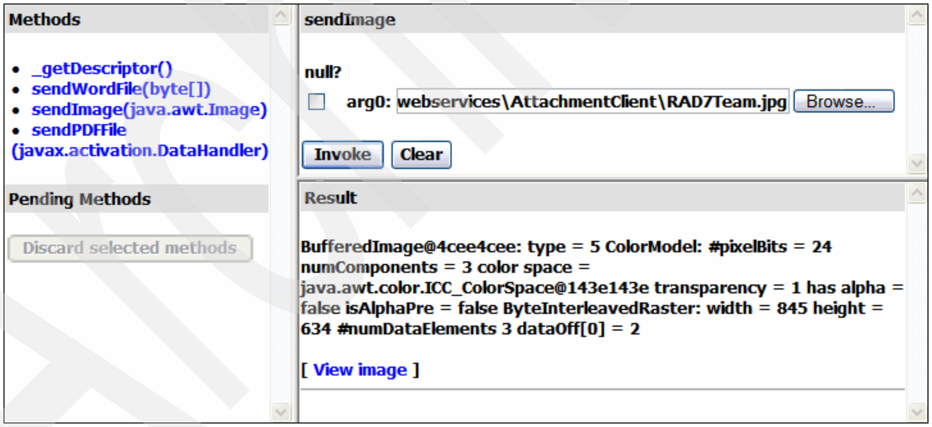
```
package com.ibm.rad7.mtom;

import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.image.BufferedImage;
import java.io.BufferedInputStream;
import java.io.File;
import java.io.FileOutputStream;
import javax.activation.DataHandler;
import javax.imageio.ImageIO;

@javax.jws.WebService(endpointInterface =
         "com.ibm.rad7.mtom.ProcessDocumentDelegate",
         targetNamespace = "http://mtom.rad7.ibm.com/", serviceName =
         "ProcessDocumentService", portName = "ProcessDocumentPort")
@javax.xml.ws.BindingType(value =
         javax.xml.ws.soap.SOAPBinding.SOAP11HTTP_MTOM_BINDING)
public class ProcessDocumentPortImpl {

   public byte[] sendWordFile(byte[] arg0) {
      try {
         FileOutputStream fileOut = new FileOutputStream(new File(
               "C:/7501code/WebServices/AttachmentServer/ws-tips.doc"));
         fileOut.write(arg0);
      } catch (Exception e) {
         e.printStackTrace();
      }
      return arg0;
   }

   public Image sendImage(Image arg0) {
      try {
         File file = new File(
               "C:/7501code/WebServices/AttachmentServer/RAD7Team.jpg");
         BufferedImage bi = new BufferedImage(arg0.getWidth(null),
                         arg0.getHeight(null), BufferedImage.TYPE_INT_RGB);
         Graphics2D g2d = bi.createGraphics();
         g2d.drawImage(arg0, 0, 0, null);
         ImageIO.write(bi, "jpeg", file);
      } catch (Exception e) {
         e.printStackTrace();
      }
      return arg0;
   }

   public DataHandler sendPDFFile(DataHandler arg0) {
      try {
         FileOutputStream fileOut = new FileOutputStream(new File(
               "C:/7501code/WebServices/AttachmentServer/ws-https.pdf"));
         BufferedInputStream fileIn = new BufferedInputStream
                                            (arg0.getInputStream());
         while (fileIn.available() != 0) {
            fileOut.write(fileIn.read());
         }
      } catch (Exception e) {
         e.printStackTrace();
      }
      return arg0;
   }

}
```

- Examine the code listed in Example 18-6.
  - The `sendWordFile` method takes a byte[] as input and stores the binary data as `C:/7501code/webservices/AttachmentServer/ws-tips.doc`.
  - The `sendImage` method takes an image as input and stores the binary data as `C:/7501code/webservices/AttachmentServer/RAD7Team.jpg`.
  - The `sendPDFFile` method takes a `DataHandler` as input and stores the data in `C:/7501code/webservices/AttachmentServer/ws-https.pdf`.
  - All the three methods return the received data to the client after storing it on the local drive.

Test and monitor the MTOM enabled Web service

Now it is the time to see if MTOM really optimizes the transmission of the data.

- Create a folder `C:\7501code\WebServices\AttachmentServer`. We use this folder to store the document received by the Web service JavaBean.
- In the sample JSP client, select the **sendImage** method.
- Click **Browse** and navigate to `C:\7501code\webservices\AttachmentClient`. Select **RAD7Team.jpg** and click **Open**.
- Click **Invoke** to invoke the `sendImage` method (Figure 18-50).

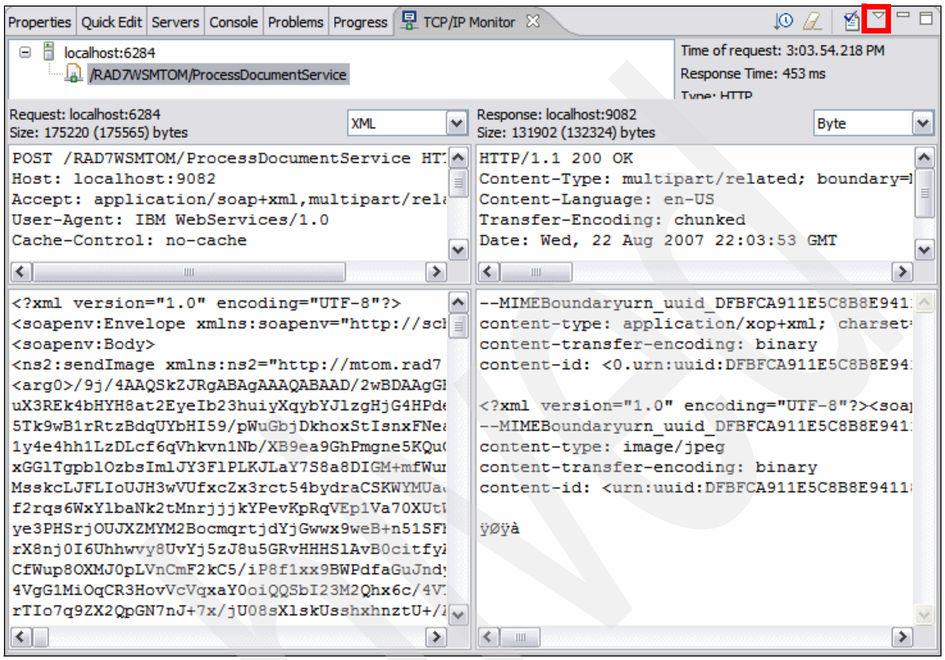


```
BufferedImage@4cee4cee: type = 5 ColorModel: #pixelBits = 24
numComponents = 3 color space =
java.awt.color.ICC_ColorSpace@143e143e transparency = 1 has alpha =
false isAlphaPre = false ByteInterleavedRaster: width = 845 height =
634 #numDataElements 3 dataOff[0] = 2
```

Figure 18-50 Invoking the MTOM Web service sendImage method

- In the Result pane, click **View image**. The author image is displayed in the Results pane (the author image is in the Preface of this document).
- Examine the `C:\7501code\webservices\AttachmentServer` folder. You can see that `RAD7Team.jpg` is stored in this folder. Note that the size is different from the same file in the `AttachmentClient` folder (probably a different JPEG compression is used).

- Select the TCP/IP Monitor tab to view the SOAP traffic. Click the ▽ icon and then select **Show Header**. The HTTP header and the SOAP traffic are shown in Figure 18-51.



Time of request: 3:03.54.218 PM
Response Time: 453 ms
Type: HTTP

Request: localhost:6284
Size: 175220 (175565) bytes

```
POST /RAD7WSMTOM/ProcessDocumentService HT
Host: localhost:9082
Accept: application/soap+xml,multipart/rel
User-Agent: IBM WebServices/1.0
Cache-Control: no-cache
```

```
<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://sc
<soapenv:Body>
<ns2:sendImage xmlns:ns2="http://mtom.rad7
<arg0>/9j/4AAQSkZJRgABAgAAAQABAAD/2wBDAAgG
uX3REk4bHYH8at2EyeIb23huiyXqybYJlzgHjG4HPd
5Tk9wB1rRtzBdqUYbHI59/pWuGbjDkhoxStIsnxFNe
1y4e4hh1LzDLcf6qVhkvn1Nb/XB9ea9GhPmgne5KQu
xGGlTgpblOzbsImlJY3F1PLKJLaY7S8a8DIGM+mfWu
MsskcLJFLIoUJH3wVUfxcZx3rct54bydraCSKWYMUa
f2rqs6WxYlbaNk2tMnrjjjkYPevKpRqVEp1Va70XUt
ye3PHSrjOUJXZMYM2BocmqrtjdYjGwwx9weB+n51SF
rX8nj0I6Uhhwvy8UvYj5zJ8u5GRvHHHS1AvB0citfy
CfWup8OXMJ0pLVnCmF2kC5/iP8f1xx9BWPdfaGuJnd
4VgG1MiOqCR3HovVcVqxaY0oiQQSbI23M2Qhx6c/4V
rTIo7q9ZX2QpGN7nJ+7x/jU08sXlskUsshxhnztU+/
```

Response: localhost:9082
Size: 131902 (132324) bytes

```
HTTP/1.1 200 OK
Content-Type: multipart/related; boundary=
Content-Language: en-US
Transfer-Encoding: chunked
Date: Wed, 22 Aug 2007 22:03:53 GMT
```

```
--MIMEBoundaryurn_uuid_DFBFCA911E5C8B8E941
content-type: application/xop+xml; charset
content-transfer-encoding: binary
content-id: <0.urn:uuid:DFBFCA911E5C8B8E94

<?xml version="1.0" encoding="UTF-8"?><soa
--MIMEBoundaryurn_uuid_DFBFCA911E5C8B8E941
content-type: image/jpeg
content-transfer-encoding: binary
content-id: <urn:uuid:DFBFCA911E5C8B8E9411

ÿØÿà
```

Figure 18-51 SOAP traffic when MTOM is only enabled for the Web service

- Take a look at the SOAP request and response:
  - The Web service (provider) has MTOM enabled after the code generation. Therefore, the SOAP response has a smaller payload! The Web service sends the binary data as a MIME attachment outside the XML document to realize the optimization.
  - The SOAP request has a much larger payload because MTOM is not enabled. The Web service client sends binary data as base64 encoded data within the XML document.
- The SOAP response and its HTTP header are shown in Example 18-7 (formatted by hand).

Example 18-7 SOAP response message and HTTP header with MTOM enabled

```
HTTP/1.1 200 OK
Content-Type: multipart/related;
   boundary=MIMEBoundaryurn_uuid_0A4725631EA4903E161185655686209;
   type="application/xop+xml";
   start="<0.urn:uuid:0A4725631EA4903E161185655686210@apache.org>";\
   start-info="text/xml"; charset=UTF-8
Content-Language: en-CA
Transfer-Encoding: chunked
Date: Sat, 28 Jul 2007 20:48:05 GMT
Server: WebSphere Application Server/6.1
==============================================================================
--MIMEBoundaryurn_uuid_0A4725631EA4903E161185655686209
content-type: application/xop+xml; charset=UTF-8; type="text/xml"
content-transfer-encoding: binary
content-id: <0.urn:uuid:0A4725631EA4903E161185655686210@apache.org>

<?xml version="1.0" encoding="UTF-8"?>
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
   <soapenv:Body>
      <ns2:sendImageResponse xmlns:ns2="http://mtom.rad7.ibm.com/">
         <return>
            <xop:Include
               href="cid:urn:uuid:0A4725631EA4903E161185655686383@apache.org"
               xmlns:xop="http://www.w3.org/2004/08/xop/include">
            </xop:Include>
         </return>
      </ns2:sendImageResponse>
   </soapenv:Body>
</soapenv:Envelope>
--MIMEBoundaryurn_uuid_0A4725631EA4903E161185655686209
content-type: image/jpeg
content-transfer-encoding: binary
content-id: <urn:uuid:0A4725631EA4903E161185655686383@apache.org>

ÿØÿà
```

The `type` and `content-type` attributes have the value **`application/xop+xml`**, which indicates that the message was successfully optimized using XML-binary Optimized packaging (XOP) when MTOM was enabled.

Enabling MTOM on the client

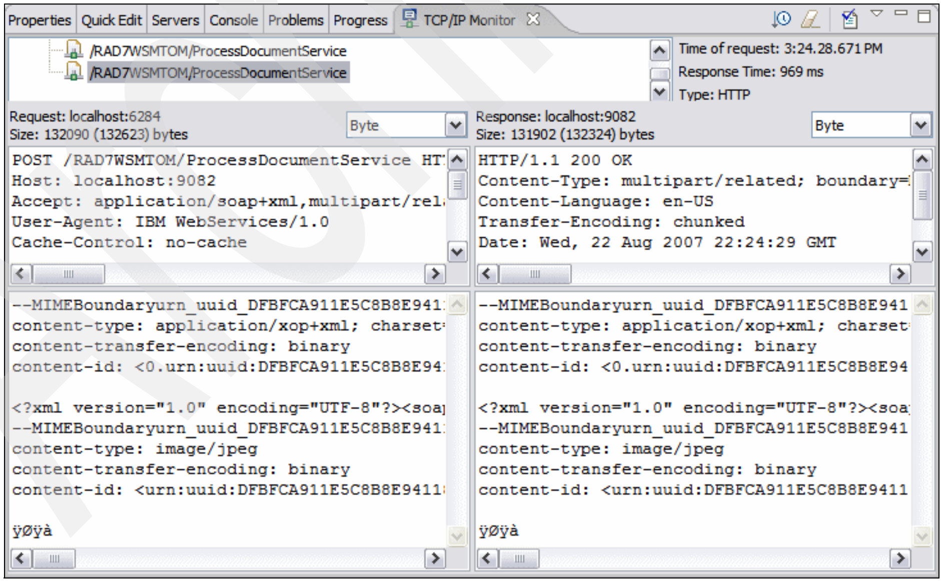
Now let us enable MTOM on the client side as well:

- In the Project Explorer expand **RAD7MTOMClient** → **Java Resources** → **src** → **com.ibm.rad7.mtom** and open **ProcessDocumentPortProxy.java**.
- Add two lines of code to each business method (Example 18-8). The modified `ProcessDocumentPortProxy.java` can also be found in the `C:\7501code\webservices` folder.

Example 18-8   Enable MTOM for the Web service client

```
import javax.xml.ws.soap.SOAPBinding;
......
public byte[] sendWordFile(byte[] arg0) {
   SOAPBinding binding = (SOAPBinding)((BindingProvider)_getDescriptor()
                                          .getProxy()).getBinding();
   binding.setMTOMEnabled(true);
    return _getDescriptor().getProxy().sendWordFile(arg0);
}
public Image sendImage(Image arg0) {
   SOAPBinding binding = (SOAPBinding)((BindingProvider)_getDescriptor()
                                          .getProxy()).getBinding();
   binding.setMTOMEnabled(true);
   return _getDescriptor().getProxy().sendImage(arg0);
}
public DataHandler sendPDFFile(DataHandler arg0) {
    SOAPBinding binding = (SOAPBinding)((BindingProvider)_getDescriptor()
                                          .getProxy()).getBinding();
   binding.setMTOMEnabled(true);
   return _getDescriptor().getProxy().sendPDFFile(arg0);
}
```

- Run the sample JSP again.The SOAP request also has a small payload after MTOM is enabled for the Web service client (Figure 18-52).



/RAD7WSMTOM/ProcessDocumentService
/RAD7WSMTOM/ProcessDocumentService

Time of request: 3:24.28.671 PM
Response Time: 969 ms
Type: HTTP

Request: localhost:6284
Size: 132090 (132623) bytes

Response: localhost:9082
Size: 131902 (132324) bytes

```
POST /RAD7WSMTOM/ProcessDocumentService HT
Host: localhost:9082
Accept: application/soap+xml,multipart/rel
User-Agent: IBM WebServices/1.0
Cache-Control: no-cache
```

```
HTTP/1.1 200 OK
Content-Type: multipart/related; boundary=
Content-Language: en-US
Transfer-Encoding: chunked
Date: Wed, 22 Aug 2007 22:24:29 GMT
```

```
--MIMEBoundaryurn_uuid_DFBFCA911E5C8B8E941
content-type: application/xop+xml; charset
content-transfer-encoding: binary
content-id: <0.urn:uuid:DFBFCA911E5C8B8E94

<?xml version="1.0" encoding="UTF-8"?><soa
--MIMEBoundaryurn_uuid_DFBFCA911E5C8B8E941
content-type: image/jpeg
content-transfer-encoding: binary
content-id: <urn:uuid:DFBFCA911E5C8B8E9411

ÿØÿà
```

```
--MIMEBoundaryurn_uuid_DFBFCA911E5C8B8E941
content-type: application/xop+xml; charset
content-transfer-encoding: binary
content-id: <0.urn:uuid:DFBFCA911E5C8B8E94

<?xml version="1.0" encoding="UTF-8"?><soa
--MIMEBoundaryurn_uuid_DFBFCA911E5C8B8E941
content-type: image/jpeg
content-transfer-encoding: binary
content-id: <urn:uuid:DFBFCA911E5C8B8E9411

ÿØÿà
```

Figure 18-52   SOAP message with MTOM enabled for both client and server

- In the sample JSP client, invoke the `sendWordFile` method. Click **Browse** to locate the word document ...`\AttachmentClient\`**ws-tips.doc,** and click **Invoke**. Watch the SOAP traffic in the TCP/IP Monitor.
- Invoke the `sendPDFFile` method. Click **Browse** to locate the PDF document ...`\AttachmentClient\`**ws-https.pdf**, and click **Invoke**. Watch the SOAP traffic in the TCP/IP Monitor.
- Verify the `C:\7501code\webservices\AttachmentServer` folder. We can see that the image file, word document, and PDF file are all stored successfully.

Creating Web services using annotations

The Java API for XML-Based Web Services (JAX-WS) programming standard relies on the use of annotations to specify metadata that is associated with Web service implementations. The standard also relies on annotations to simplify the development of Web services. The JAX-WS standard supports the use of annotations that are based on several Java Specification Requests (JSRs):

- A Metadata Facility for the Java Programming Language (JSR 175)
- Web Services Metadata for the Java Platform (JSR 181)
- Java API for XML-Based Web Services (JAX-WS) 2.0 (JSR 224)
- Common Annotations for the Java Platform (JSR 250)

Using annotations from the JSR 181 standard, you can annotate a service implementation class or a service interface. Then you can generate a Web service with a wizard or by publishing the application to a server. Using annotations within both Java source code and Java classes simplifies Web service development. Using annotations in this way defines additional information that is typically obtained from deployment descriptor files, Web Services Description Language (WSDL) files, or mapping metadata from XML and WSDL into source artifacts.

In this section, we create a bottom-up Web service from a JavaBean using annotations. The Web services are generated by publishing the application to a server. No wizard is required in this example.

Import the sample

This section describes the steps required for preparing the Web services annotations samples. Follow these steps:

- Switch to the J2EE perspective Project Explorer view.
- Select **File** → **Import** → **Other** → **Project Interchange**.

- In the Import Projects dialog, click **Browse** to navigate and select the **RAD7WSFPAnnotationStart.zip** from the `c:\7501code\webservices` folder, and click **Open**.
- There are three projects in the interchange file. If you already have the `RAD7WebServiceUtility` project in the workspace, only select the `RAD7WSFPAnnotationEAR` and `RAD7WSFPAnnotationWeb` projects, otherwise click **Select All**. Click **Finish**.

The sample project is almost identical to the project we imported at the beginning of this chapter. There is one difference: the WebSphere 6.1 Feature Pack for Web Services facet is added to the project.

Annotate a JavaBean

You can annotate types, methods, fields, and parameters in the JavaBean to specify a Web service. To annotate the JavaBean, open the `SimpleBankBean` in the Web project **RAD7WSFPAnnotationWeb** and add the imports and annotation tags highlighted in Example 18-9 (available in `C:\7501code\webservices\SimpleBankBeanAnnotation.txt`).

Example 18-9 Add annotation tags to a JavaBean

```
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.jws.WebResult;
import javax.jws.WebService;
import javax.jws.soap.SOAPBinding;

@WebService(name="BankWebService",
            targetNamespace="http://ibm.com/rad7/BankWebService")
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT,use=SOAPBinding.Use.LITERAL,
            parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
public class SimpleBankBean implements Serializable {

   public SimpleBankBean() {
   }

@WebMethod(operationName="RetrieveCustomerName",
            action="urn:getCustomerFullName")
@WebResult(name="CustomerFullName")
   public String getCustomerFullName(@WebParam(name="ssn")String ssn)
      ......
```

- The **@WebService** tag marks a Java class as implementing a Web service.
  - The *name* attribute is used as the name of the `wsdl:portType` when mapped to WSDL 1.1.

  - *The targetNamespace* attribute is the XML namespace used for the WSDL and XML elements generated from this Web service.

- The **@SOAPBinding** tag specifies the mapping of the Web Service to the SOAP message protocol. In this example, we use *Document/Literal/Wrapped* style.
- The **@WebMethod** tag identifies the individual methods of the Java class that are exposed externally as Web Service operations. In this example, we only expose the *getCustomerFullName* method as a Web service operation.
  - *operationName* is the name of the `wsdl:operation` matching this method.
  - *action* determines the value of the soap action for this operation.
- The **@WebParam** and **@WebResult** tags customize the mapping of the method parameters and results to message parts and XML elements.

Validate Web services annotations

When developing Web services, you can benefit from two levels of validation. The first level involves validating syntax and Java-based default values. This level of validation is performed by the Eclipse Java Development Tools (JDT). The second level of validation involves implicit default checking, as well as verification of Web Services Description Language (WSDL) contracts. This second level is performed by a JAX-WS annotations processor in the WebSphere 6.1 Feature Pack for Web Services.

When you select the WebSphere 6.1 Feature Pack for Web Services facet for a project, you enable the JAX-WS annotations processor. When you enable the annotations processor, warnings and errors for annotations are displayed like Java errors. You can work with these warnings and errors in various workbench locations, such as the Problems view.

By using the annotations processor to detect problems at build time, you can prevent these problems from occurring at run time. For example, if you make the changes of Example 18-10 (instead of Example 18-9), you receive validation errors as shown in Example 18-11.

Example 18-10 Validate Web services annotations

```
@WebService(name="!BankWebService",
            targetNamespace="/ibm.com/rad7/BankWebService")
@SOAPBinding(style=SOAPBinding.Style.DOCUMENT, use=SOAPBinding.Use.LITERAL,
parameterStyle=SOAPBinding.ParameterStyle.WRAPPED)
public class SimpleBankBean implements Serializable {

   public SimpleBankBean() {
   }
@WebMethod(operationName= "!RetrieveCustomerName",
                              action="urn:getCustomerFullName")
@WebResult(name="CustomerFullName")
@Oneway
   public String getCustomerFullName(@WebParam(name="ssn")String ssn)
         throws CustomerDoesNotExistException {
```

Example 18-11   JAX-WS annotation processor validation results

```
JSR-181, 4.3.1: Oneway methods cannot return a value
JSR-181, 4.3.1: Oneway methods cannot throw checked exceptions
name must be a valid nmToken
operationName must be a valid nmToken
targetNamespace must be a valid URI
```

Creating a Web service from an annotated JavaBean by publishing to the WSFP enabled server

After annotating a Java bean, you can generate a Web service application by publishing the application project of the bean directly to a server. When the Web service is generated, no WSDL file is created in your project.

To create a Web service from an annotated JavaBean:

- In the Servers view start **WebSphere v6.1 Server WSFP** (if not running).
- Right-click **WebSphere v6.1 Server WSFP**, select **Add and Remove Projects**, and add the **RAD7WSFPAnnotationEAR** project. Click **Finish**.

Test the JAX-WS Web service using the Web Services Explorer

To test the Web service, do these steps:

- In the Project Explorer expand the **JAX-WS Web Services** folder.
- Expand **Services** → **RAD7WSFPAnnotationWeb:{http://ibm.com/rad7/BankWebService}SimpleBankBeanService** and select **Test with Web Services Explorer** (Figure 18-53).

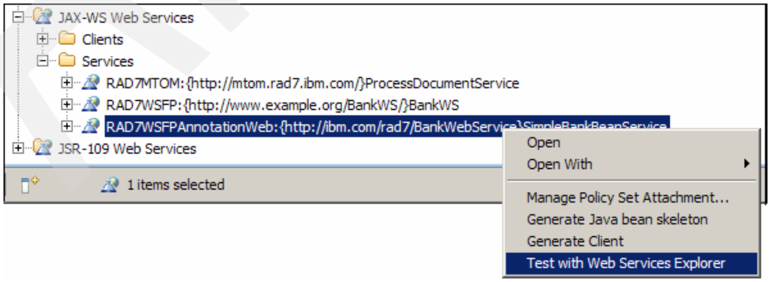
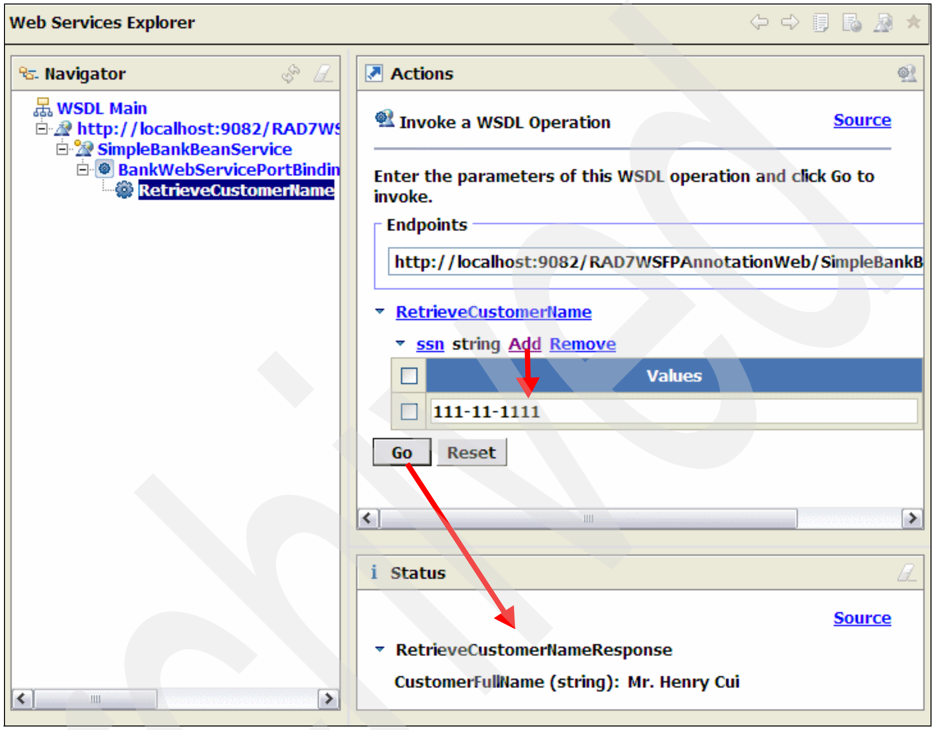

JAX-WS Web Services
Clients
Services
RAD7MTOM:{http://mtom.rad7.ibm.com/}ProcessDocumentService
RAD7WSFP:{http://www.example.org/BankWS/}BankWS
RAD7WSFPAnnotationWeb:{http://ibm.com/rad7/BankWebService}SimpleBankBeanService
JSR-109 Web Services
1 items selected
Open
Open With
Manage Policy Set Attachment...
Generate Java bean skeleton
Generate Client
Test with Web Services Explorer

Figure 18-53   Test JAX-WS Web service using Web Services Explorer

- The Web Services Explorer comes up. Select the **RetrieveCustomerName** operation.
- Click **Add** and type 111-11-1111 in the ssn field.
- Click **Go** and the result `Mr. Henry Cui` is displayed in the status pane.



Invoke a WSDL Operation

Source

Enter the parameters of this WSDL operation and click Go to invoke.

Endpoints

http://localhost:9082/RAD7WSFPAnnotationWeb/SimpleBankB

RetrieveCustomerName

ssn string Add Remove

□	Values
□	111-11-1111

Go Reset

Source

RetrieveCustomerNameResponse

CustomerFullName (string): Mr. Henry Cui

Figure 18-54 JAX-WS Web service test result with Web Services Explorer

View the dynamically generated WSDL

JAX-WS requires no deployment descriptors because it uses annotations. The WSDL file can also be dynamically generated by the runtime based on information it gathers from the annotations added to the Java classes.

The dynamically generated WSDL is in this format:

```
http://<hostname>:<port>/<Web project context root>/<JavaBean name>Service
                                                                       ?wsdl
```

To view the dynamically generated WSDL:

- Determine the port of the Web service if you do not already know the port from running the Web Services Explorer:
  - Launch the WebSphere Administrative Console.
  - Expand **Servers** → **Application servers** and select **server1**.
  - Select the **Configuration** tab, look for the **Communications** heading and expand **Ports**.
  - The port used is called `WC_defaulthost`.
- Determine the Web project context root in the **Properties** dialog of the `RAD7WSFPAnnotationWeb` project under **Web Project Settings**. By default, the context root is the same as the Web project name.
- Enter the following URL in the browser (**xxxx** is the port number, most probably `9082`):

  ```
  http://localhost:xxxx/RAD7WSFPAnnotationWeb/SimpleBankBeanService?wsdl
  ```

- The dynamically generated WSDL file is displayed. We also notice that the URL for the WSDL is changed to:

  ```
  http://localhost:xxxx/RAD7WSFPAnnotationWeb/SimpleBankBeanService
                                           /simplebankbeanservice.wsdl
  ```

- This WSDL imports the XML schema types from `SimpleBankBeanService_schema1.xsd`:

  ```
  <xsd:import namespace="http://ibm.com/rad7/BankWebService"
              schemaLocation="SimpleBankBeanService_schema1.xsd" />
  ```

  You can retrieve the `SimpleBankBeanService_schema1.xsd` using this URL:

  ```
  http://localhost:xxxx/RAD7WSFPAnnotationWeb/SimpleBankBeanService
                                   /SimpleBankBeanService_schema1.xsd
  ```

- Examine the generated WSDL and XSD file. We can see that the generated WSDL and XSD match the Web services annotations added in Example 18-9 on page 890. The generated WSDL snippet is listed in Example 18-12.

Example 18-12 Dynamically generated WSDL snippet

```
<?xml version="1.0" encoding="UTF-8" ?>
<definitions name="SimpleBankBeanService"
   targetNamespace="http://ibm.com/rad7/BankWebService"
   xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:tns="http://ibm.com/rad7/BankWebService"
   xmlns="http://schemas.xmlsoap.org/wsdl/">
   ......
   <portType name="BankWebService">
      <operation name="RetrieveCustomerName">
         <input message="tns:RetrieveCustomerName" />
         <output message="tns:RetrieveCustomerNameResponse" />
         <fault name="CustomerDoesNotExistException"
            message="tns:CustomerDoesNotExistException" />
      </operation>
   </portType>
   <binding name="BankWebServicePortBinding"
      type="tns:BankWebService">
      <soap:binding style="document"
         transport="http://schemas.xmlsoap.org/soap/http" />
      <operation name="RetrieveCustomerName">
         <soap:operation soapAction="urn:getCustomerFullName" />
         <input>
            <soap:body use="literal" />
         ......
      </operation>
   </binding>
   <service name="SimpleBankBeanService">
      <port name="BankWebServicePort" binding="tns:BankWebServicePortBinding">
         <soap:address location="http://localhost:9082/RAD7WSFPAnnotationWeb
                                                       /SimpleBankBeanService"/>
      </port>
   </service>
</definitions>
```

- A simple test to verify that the Web service is running in the server can be performed using this URL:

  ```
  http://localhost:xxxx/RAD7WSFPAnnotationWeb/SimpleBankBeanService
  ```

  The result displayed in the browser is:

  ```
  {http://ibm.com/rad7/BankWebService}SimpleBankBeanService
  Hello! This is an Axis2 Web Service!
  ```

More information

For more information on Web services, refer to these resources:

- See the Redbooks publication: *Web Services Handbook for WebSphere Application Server Version 6.1*, SG24-6957.
- IBM developerWorks has a whole section on **SOA and Web services**:

  `http://www.ibm.com/developerworks/webservices`
- An online list of current and emerging Web services standards can be found on developerWorks under **SOA and Web services** → **Standards**:

  `http://www.ibm.com/developerworks/webservices/standards/`
- Search for **JAX-WS** or **MTOM** on developerWorks and you can find a number of relevant documents.
- The JAX-WS specification (JSR-224 Java API for XML-Based Web Services 2.0) is available at:

  `http://jcp.org/aboutJava/communityprocess/pfd/jsr224/index.html`
- The MTOM specification is available at:

  `http://www.w3.org/TR/soap12-mtom/`
- Read about JAX-WS annotations at:

  `https://jax-ws.dev.java.net/jax-ws-ea3/docs/annotations.html`

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.wsfep.multiplatform.doc/info/ae/ae/rwbs_jaxwsannotations.html`

19

Develop portal applications

This chapter describes the portal development tools that are included with IBM Rational Application Developer V7.0. We also highlight how the portal tools in Application Developer can be used to develop a portal and associated portlet applications for WebSphere Portal v6.0. Finally, we have included a development scenario to demonstrate how to use the new integrated portal tooling to develop a portal, customize the portal, and develop two portlets.

The chapter is organized into the following sections:

- Introduction to portal technology
- Developing applications for WebSphere Portal
- Developing portal solutions using portal tools

Note: For more detailed information on IBM WebSphere Portal v6.0, refer to "More information" on page 960.

The following Redbooks publications cover the Portlet Application Development for the older versions of WebSphere Portal:

- *IBM WebSphere Portal V5 A Guide for Portlet Application Development*, SG24-6076
- *IBM WebSphere Portal V5.1 Portlet Application Development*, SG24-6681

Introduction to portal technology

As J2EE technology has evolved, much emphasis has been placed on the challenges of building enterprise applications and bringing those applications to the Web. At the core of the challenges currently being faced by Web developers is the integration of disparate user content into a seamless Web application and well-designed user interface. Portal technology provides a framework to build such applications for the Web.

Because of the increasing popularity of portal technologies, the tooling and frameworks used to support the building of new portals has evolved. The main job of a portal is to aggregate content and functionality. Portal servers provide:

- A server to aggregate content
- A scalable infrastructure
- A framework to build portal components and extensions

Additionally, many portals offer personalization and customization features. Personalization enables the portal to deliver user-specific information targeting a user based on their unique information. Customization allows the user to organize the look and feel of the portal to suit their individual needs and preferences.

Portals deliver e-business applications over the Web to many types of client devices from PCs to PDAs. Portals provide site users with a single point of access to multiple types of information and applications. Regardless of where the information resides or what format it is in, a portal aggregates all of the information in a way that is relevant to the user.

The goal of implementing an enterprise portal is to enable a working environment that integrates people, their work, personal activities, and supporting processes and technology.

Portal concepts and definitions

Before beginning development for portals, you should become familiar with some common definitions and descriptions of portal-related terminology.

Portal page

A portal page is a single Web page that can be used to display content aggregated from multiple sources. The content that appears on a portal page is displayed by an arrangement of one or more portlets. For example, a World Stock Market portal page might contain two portlets that display stock tickers for popular stock exchanges and a third portlet that displays the current exchange rates for world currencies.

Portlet

A portlet is an individual application that displays content on a portal page. To a user, a portlet is a single window or panel on the portal page that provides information or Web application functionality. To a developer, portlets are Java-based pluggable modules that can access content from a source such as another Web site, an XML feed, or a database, and display this content to the user as part of the portal page.

Portlet application

Portlet applications are collections of related portlets and resources that are packaged together. Portlets within the same portlet application can exchange and share data and act as a unit. All portlets packaged together share the same context, which contains all resources such as images, properties files, and classes.

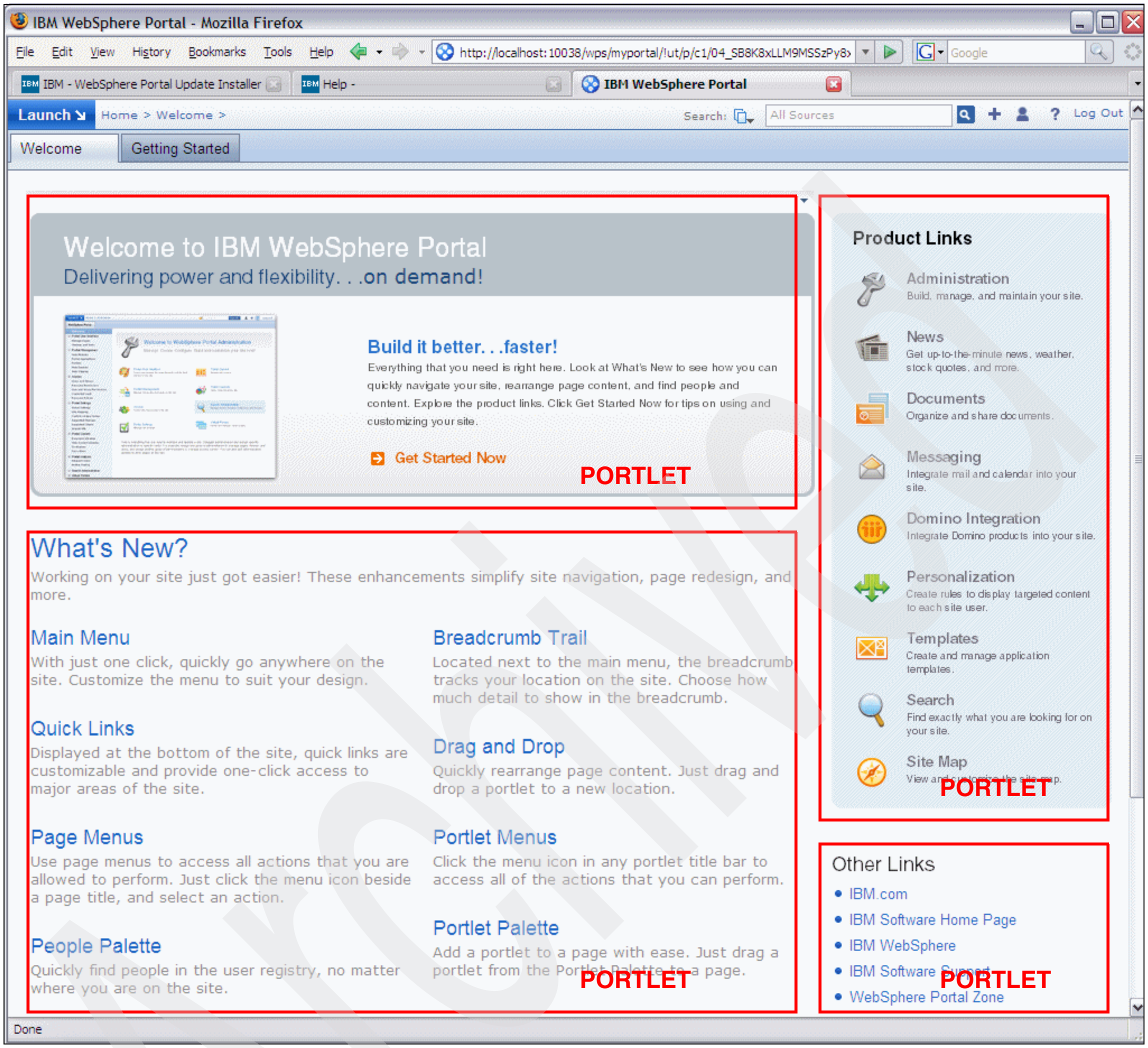
Portlet states

Portlet states determine how individual portlets look when a user accesses them on the portal page. These states are very similar to minimize, restore, and maximize window states of applications run on any popular operating system just in a Web-based environment.

The state of the portlet is stored in the `PortletWindow.State` object and can be queried for changing the way a portlet looks or behaves based on its current state. The IBM portlet API defines three possible states for a portlet:

- **Normal**: The portlet is displayed in its initial state, as defined when it was installed.
- **Minimized**: Only the portlet title bar is visible on the portal page.
- **Maximized**: The portlet fills the entire body of the portal page, hiding all other portlets.

Figure 19-1 shows a portal page with several portlets.


IBM WebSphere Portal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:10038/wps/myportal/!ut/p/c1/04_SB8K8xLLM9MSSzPy8x
Google
IBM - WebSphere Portal Update Installer
Help -
IBM WebSphere Portal
Launch
Home > Welcome >
Search:
All Sources
Log Out
Welcome
Getting Started
Done

Welcome to IBM WebSphere Portal

Delivering power and flexibility. . .on demand!

Build it better. . .faster!

Everything that you need is right here. Look at What's New to see how you can quickly navigate your site, rearrange page content, and find people and content. Explore the product links. Click Get Started Now for tips on using and customizing your site.

Get Started Now

PORTLET

What's New?

Working on your site just got easier! These enhancements simplify site navigation, page redesign, and more.

Main Menu

With just one click, quickly go anywhere on the site. Customize the menu to suit your design.

Breadcrumb Trail

Located next to the main menu, the breadcrumb tracks your location on the site. Choose how much detail to show in the breadcrumb.

Quick Links

Displayed at the bottom of the site, quick links are customizable and provide one-click access to major areas of the site.

Drag and Drop

Quickly rearrange page content. Just drag and drop a portlet to a new location.

Page Menus

Use page menus to access all actions that you are allowed to perform. Just click the menu icon beside a page title, and select an action.

Portlet Menus

Click the menu icon in any portlet title bar to access all of the actions that you can perform.

People Palette

Quickly find people in the user registry, no matter where you are on the site.

Portlet Palette

Add a portlet to a page with ease. Just drag a portlet from the Portlet Palette to a page.

PORTLET

Product Links

Administration
Build, manage, and maintain your site.

News
Get up-to-the-minute news, weather, stock quotes, and more.

Documents
Organize and share documents.

Messaging
Integrate mail and calendar into your site.

Domino Integration
Integrate Domino products into your site.

Personalization
Create rules to display targeted content to each site user.

Templates
Create and manage application templates.

Search
Find exactly what you are looking for on your site.

Site Map
View and customize the site map.

PORTLET

Other Links

- IBM.com
- IBM Software Home Page
- IBM WebSphere
- IBM Software Support
- WebSphere Portal Zone

PORTLET

Figure 19-1 Portlets laid out on the Welcome Portal Page

Portlet modes

Portlet modes allow the portlet to display a different *face* depending on how it is being used. This allows different content to be displayed within the same portlet, depending on its mode. Modes are most commonly used to allow users and administrators to configure portlets or to offer help to the users. There are four modes in the IBM Portlet API:

- **View**: Initial face of the portlet when created. The portlet normally functions in this mode.

- **Edit**: This mode allows the user to configure the portlet for their personal use (for example, specifying a city for a localized weather forecast).
- **Help**: If the portlet supports the help mode, this mode displays a help page to the user.
- **Configure**: If provided, this mode displays a face that allows the portal administrator to configure the portlet for a group of users or a single user.

Portlet events

Some portlets only display static content in independent windows. To allow users to interact with portlets and to allow portlets to interact with each other, portlet events are used. Portlet events contain information to which a portlet might need to respond. For example, when a user clicks a link or button, this generates an *action* event. To receive notification of a given event, the portlet must also have the appropriate *event listener* implemented within the portlet class. There are three commonly used types of portlet events:

- **Action**: Generated when an HTTP request is received by the portlet that is associated with an action, such as when a user clicks a link.
- **Message**: Generated when one portlet within a portlet application sends a message to another portlet.
- **Window**: Generated when the user changes the state of the portlet window.

IBM WebSphere Portal

IBM WebSphere Portal provides an extensible framework that allows the end user to interact with enterprise applications, people, content, and processes. They can personalize and organize their own view of the portal, manage their own profiles, and publish and share documents. WebSphere Portal provides additional services such as single sign-on (SSO), security, credential vault, directory services, document management, Web content management, personalization, search, collaboration, search and taxonomy, support for mobile devices, accessibility support, internationalization, e-learning, integration to applications, and site analytics. Clients can further extend the portal solution to provide host integration and e-commerce.

WebSphere Portal allows you to plug in new features or extensions using portlets. In the same way that a servlet is an application within a Web server, a portlet is an application within WebSphere Portal. Developing portlets is the most important task when providing a portal that functions as the user's interface to information and tasks.

Portlets are an encapsulation of content and functionality. They are reusable components that combine Web-based content, application functionality, and access to resources. Portlets are assembled into portal pages that, in turn, make up a portal implementation.

Portal solutions such as IBM WebSphere Portal are proven to shorten the development time. Pre-built adapters and connectors are available so that customers can leverage on the company's existing investment by integrating with the existing legacy systems without re-inventing the wheel.

Portal and portlet development features in Application Developer

Application Developer provides development tools for portal and portlet applications destined to WebSphere Portal. Bundled with IBM Rational Application Developer V7.0 are a number of portal tools that allow you to create, test, debug, and deploy portal and portlet applications. Application Developer supports portlet development using the Standard and IBM portlet APIs.

The following tools are provided to support development of your portlet applications:

- Portlet project
- Portlet application samples
- New Portlet Project wizard
- WPAI mediator access
- Portlet deployment descriptor editor
- Portal server configuration
- Portal server test, debug, and deploy
- Import from a Web Archive (WAR) file
- Export to a Web Archive (WAR) file
- Visual tooling to insert portlet objects into JSP files, using Page Designer.
- Cooperative portlet wizards
- Business process message access
- Personalization wizard

Portal test environments

Application Developer v7.0 provides the following versions of integrated test environments to run and test your portal and portlet projects from within the Application Developer Workbench:

- IBM WebSphere Portal Version 6.0
- IBM WebSphere Portal Version 5.1

Both of these versions come included with the product. In this chapter we are using the Version 6.0 of WebSphere Portal. We later upgrade the version of our WebSphere Portal to the latest available version (when this book was being written), which is Version 6.0.1. We also upgrade the version of the WebSphere Application Server used by portal to Version 6.0.2.17.

Setting up Application Developer with the Portal test environment

Setting up of Portal test environment in Application Developer is now a much easier and more streamlined task. We perform the following high-level activities to complete the setup of Portal test environment in Application Developer, documented under “Installing the WebSphere Portal V6.0 test environment” on page 1296:

- Installing WebSphere Portal V6.0
- Adding WebSphere Portal V6.0 to Application Developer
- Upgrading the Portal Server runtime to Version 6.0.1
- Optimizing the Portal Server for development

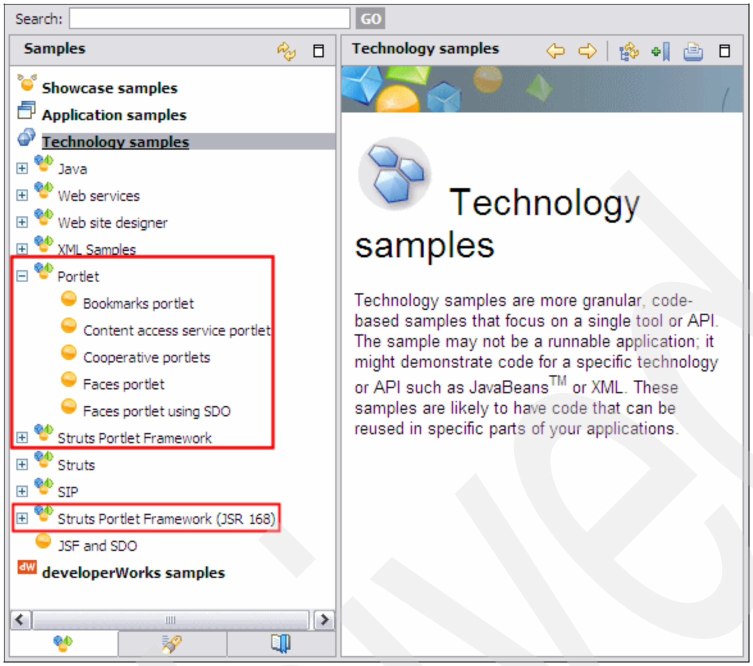
Developing applications for WebSphere Portal

Application Developer includes many tools to help you quickly develop portals and individual portlet applications. In this section, we cover some basic portlet development strategies and provide an overview of the tools included Application Developer to aid with development of WebSphere Portal.

Portal samples and tutorials

Application Developer also comes with several samples and tutorials to aid you with the development of WebSphere Portal. The Samples Gallery provides sample portlet applications to illustrate portlet development.

To access portlet samples, click **Help** → **Samples Gallery**. Then expand **Technology samples** and **Portlet** (Figure 19-2). Here you can select a Basic Portlet, Faces Portlet, or Struts Portlet Framework to view sample portlet application projects that you can then modify and build upon for your own purposes.



Technology samples are more granular, code-based samples that focus on a single tool or API. The sample may not be a runnable application; it might demonstrate code for a specific technology or API such as JavaBeans™ or XML. These samples are likely to have code that can be reused in specific parts of your applications.

Figure 19-2 Samples Gallery

The Tutorials Gallery provides detailed tutorials to illustrate portlet development. These are accessible by selecting **Help** → **Tutorials Gallery**. Then expand **Do and Learn**. You can select **Create a portal application** or **Examine the differences between portlet APIs** to view the content of these tutorials.

Development strategy

A portlet application consists of Java classes, JSP files, and other resources, such as deployment descriptors and image files. Before beginning development, several decisions must be made regarding the development strategy and technologies that is used to develop a portlet application.

Choosing a portlet API: Standard (JSR 168) or IBM

WebSphere Portal v6 supports portlet development using the following two APIs:

- Standard portlet API which was previously referred to as JSR 168 portlet API
- IBM portlet API

The portal tools included with Application Developer support both APIs. The IBM portlet API was initially supported in WebSphere Portal V4.x and will continue to be supported by WebSphere Portal. It is important to note that the IBM portlet API extends the servlet API. More information about the IBM portlet API can be found at:

`http://www.ibm.com/developerworks/websphere/zones/portal/portlet/5.0api/WPS`

The standard portlet API was formalized as the JSR 168 specification from the Java Community Process (JCP) program that addresses the requirements of content aggregation, personalization, presentation, and security for portlets running in a portal environment. It was finalized in October of 2003. Portlets conforming to the standard portlet API are more portable and reusable, because they can be deployed to any JSR 168 compliant portlet container. Application Developer supports two portlet containers: One for portlets written using the Standard portlet API and one for the portlets written using the IBM portlet API.

Unlike the IBM portlet API, the standard portlet API does not extend the servlet API. It does, however, share many of the same characteristics, such as servlet specification, session management, and request dispatching. The JSR 168 container—as implemented in WebSphere Portal V5.1—provided the Property Broker support, which can act as a messaging broker for either portlet messaging or wired (automatic cooperating) portlets. But this container did not support the *click-to-Action* (user-initiated cooperating) portlets. Application Developer and its Portal 6 Server support this feature now.

> **Note:** IBM recommends using JSR 168 for all new portlet development when the functionality it provides is sufficient for the application requirements, or when the portlet is expected to be published as a Web Service for Remote Portlets (WSRP) service. IBM is committed to the wider adoption of open standards in WebSphere Portal.

More information on JSR 168 can be found at:

`http://www.jcp.org/en/jsr/detail?id=168`

Choosing markup languages

WebSphere Portal supports mobile devices by generating pages in any markup language. Three markup languages are officially supported in Application Developer:

- HyperText Markup Language (HTML) is used for Web browsers on desktop computers. All portlet applications support HTML at a minimum.
- Wireless Markup Language (WML) is used for WAP devices that are typically Web-enabled mobile telephones.

- compact Hyper Text Markup Language (cHTML) is used for mobile devices in the NTT DoCoMo i-mode network. For more information on the i-mode network, visit the following Web site:

  http://www.nttdocomo.co.jp/english/

Adding emulator support for other markup languages

To run a portlet application that supports WML or cHTML, you must use an emulator provided by the device vendor. To add device emulator support to Application Developer do these steps:

- Select **Window** → **Preferences**.
- Expand **General** and select **Web Browser**.
- Click the **New** button to locate the device emulator appropriate for the device that you wish to test and debug.

Enabling transcoding for development in other markup languages

Transcoding is the process by which WebSphere Portal makes portal content displayable on mobile devices. By default, it is *not enabled* in the WebSphere Portal Test Environment. Therefore, you have to make some configuration changes before you can test and debug applications on mobile device emulators. You have to remove the comments from lines beginning with `#Disable Transcoding` from the following three files:

- The `PortletFilterService.properties` and `PortalFilterService.properties` files are all located by default in the following directory:

  ```
  <rad_home>\runtimes\portal_v60\shared\app\config\services
  ```

- The `services.properties` file is located by default in the following directory:

  ```
  <rad_home>\runtimes\portal_v60\shared\app\config
  ```

Choosing other frameworks

JavaServer Faces (JSF) and Struts technology can be easily incorporated into a portlet development strategy. Application Developer provides extensive tooling support to help in the creation and code generation for creating the JSF portlet or Struts portlet.

JavaServer Faces (JSF)

JavaServer Faces is a popular GUI framework for developing J2EE Web applications (JSR 127). It includes reusable user interface components, input validation, state management, server-side event handling, page life cycle management, accessibility, and internationalization. Faces-based application development can be applied to portlets, similar to the way that Faces development is implemented in Web applications.

There are many wizards to help you with Faces development. Though the previous version of Application Developer, v6, along with its Portal test environment v5.1 provided support and tooling for JSF portlets, this has been significantly enhanced in the version 7 of Application Developer and Portal v6.

JSF is now seemingly a preferred UI framework for developing J2EE Web and portlet applications because of its component-based architecture.

Refer to the Application Developer Faces documentation in the InfoCenter for usage details. Also refer to Chapter 14, “Develop Web applications using JSF and SDO” on page 587 for more detailed information on application development using the JSF framework.

Struts

Struts-based application development can also be applied to portlets, similar to the way that Struts development is implemented in Web applications. The Struts Portal Framework (SPF) was developed to merge these two technologies. SPF support in Application Developer simplifies the process of writing Struts portlet applications and eliminates the need to manage many of the underlying requirements of portlet applications. In addition, multiple wizards are present to help you create Struts portlet-related artifacts. These are the same wizards used in Struts development. Refer to the Application Developer Struts documentation for usage details.

In WebSphere Portal V5.1, Struts was fully supported in both the IBM and Standard portlet APIs; however, there was no tooling support in Application Developer v6 for this configuration. Application Developer v7 provides tooling support for Struts portlet configuration.

More information on Struts can be found at:

`http://struts.apache.org/`

Web development using Struts for non-portal applications is covered in Chapter 13, “Develop Web applications using Struts” on page 541.

Portal tools for developing portals

A portal is essentially a J2EE Web application. It provides an aggregation framework where developers can associate many portlets and portlet applications via one or more portal pages.

Application Developer includes several new portal site creation tools that enable you to visually customize portal page layout, themes, skins, and navigation.

Portal Import wizard

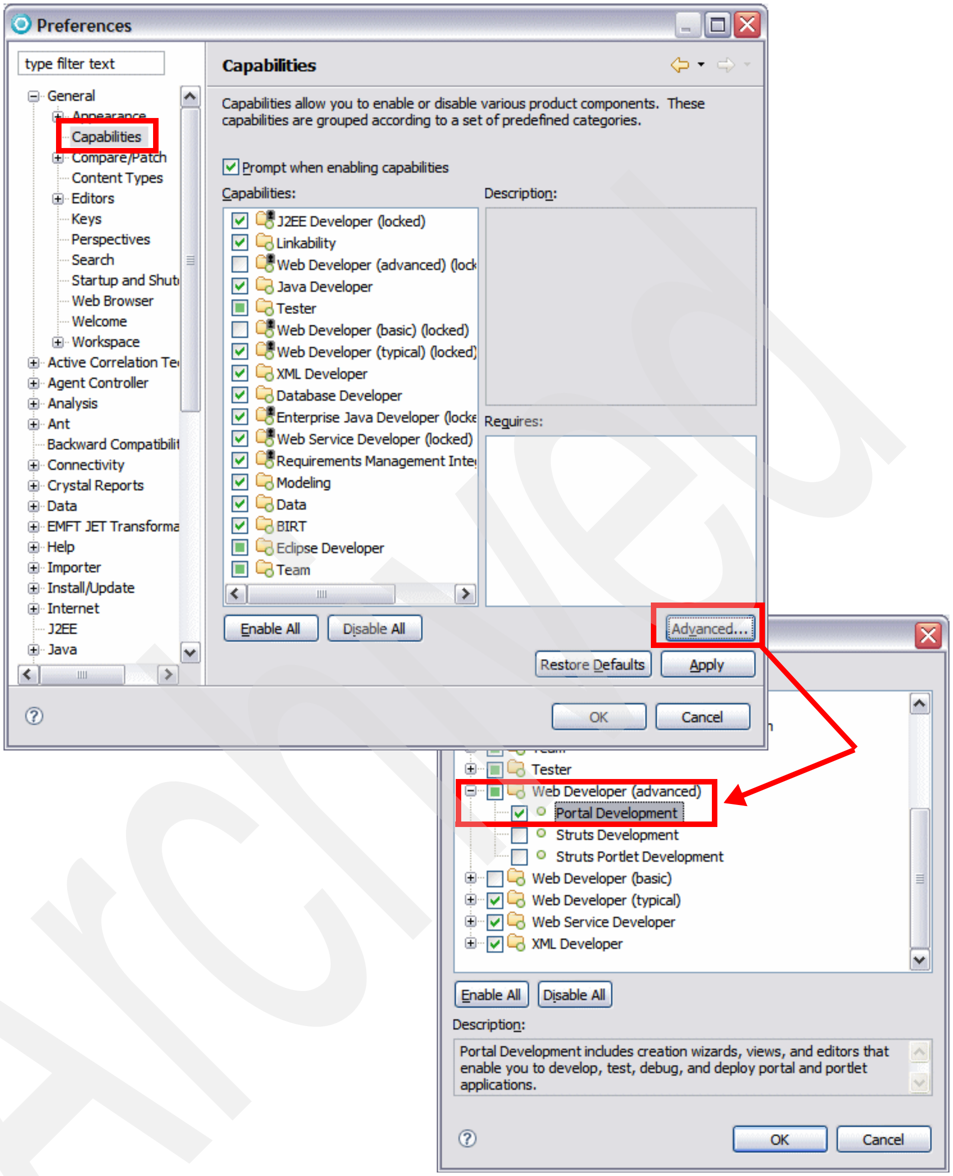
One way to create a new Portal project is to import an existing portal site from a WebSphere Portal server into Application Developer. Importing is also useful for updating the configuration of a project that already exists in Application Developer.

The portal site configuration on WebSphere Portal server contains the following resources: the global settings, the resource definitions, the portal content tree, and the page layout. Importing these resources from WebSphere Portal server to Application Developer overwrites duplicate resources within the existing Portal project. Non-duplicate resources from the server configuration are copied into the existing Portal project. Likewise, resources that are unique to the Portal project are not affected by the import.

Application Developer uses the XML configuration interface to import a server configuration, and optionally retrieves files from `installedApps/node/wps.ear` of the Application Server installation. These files include the JSP, CSS, and image files for themes and skins. When creating a new Portal project, retrieving files is mandatory. To retrieve files, Application Developer must have access to this directory, as specified when you define a new server for this project.

By default, Application Developer portal development capability is not enabled. To enable portal development capability, do these steps:

- Select **Window** → **Preferences**.
- In the Preferences dialog, expand **General** → **Capabilities** and click **Advanced** (Figure 19-3).
- In the Advanced dialog, expand **Web Developer (advanced)**, select **Portal Development** and click **OK**.
- Click **OK** in the Preferences dialog and portal development is enabled.



Capabilities allow you to enable or disable various product components. These capabilities are grouped according to a set of predefined categories.

☑ Prompt when enabling capabilities

Capabilities:

- ☑ J2EE Developer (locked)
- ☑ Linkability
- ☐ Web Developer (advanced) (lock
- ☑ Java Developer
- ▣ Tester
- ☐ Web Developer (basic) (locked)
- ☑ Web Developer (typical) (locked)
- ☑ XML Developer
- ☑ Database Developer
- ☑ Enterprise Java Developer (locke
- ☑ Web Service Developer (locked)
- ☑ Requirements Management Inte
- ☑ Modeling
- ☑ Data
- ☑ BIRT
- ▣ Eclipse Developer
- ▣ Team

- ▣ Tester
- ▣ Web Developer (advanced)
  - ☑ Portal Development
  - ☐ Struts Development
  - ☐ Struts Portlet Development
- ☐ Web Developer (basic)
- ☑ Web Developer (typical)
- ☑ Web Service Developer
- ☑ XML Developer

Portal Development includes creation wizards, views, and editors that enable you to develop, test, debug, and deploy portal and portlet applications.

Figure 19-3 Enable Portal Development capability

Now you can access the Portal Import wizard by selecting **File** → **Import** and expand Portal then selecting **Portal**. You have to specify the server and options for importing the project into Application Developer.

Follow the instructions in the Help Topics on Developing Portal Applications to ensure that the configuration in the development environment accurately reflects that of the staging or runtime environment. If you do not do this, you might experience compilation errors after the product is imported or encounter unexpected portal behaviors.

Portal Project wizard

The New Portal Project wizard will guide you through the process of creating a Portal project within Application Developer.

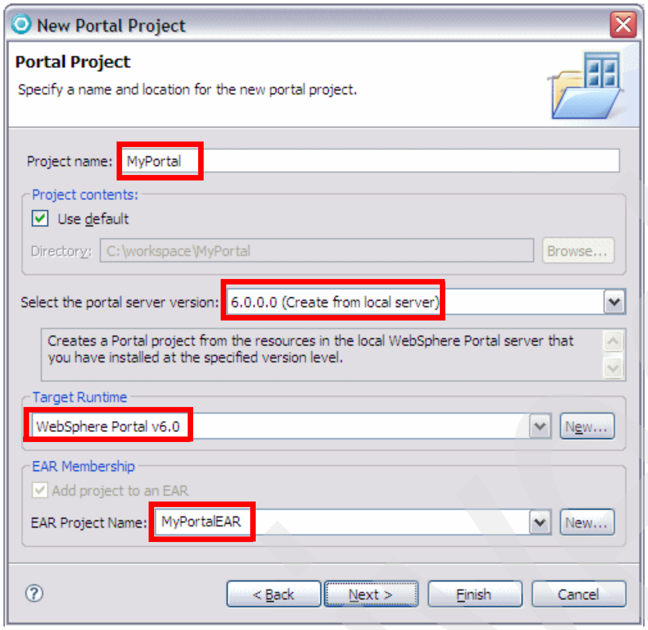
During this process, you are able to:

- Specify a project name.
- Select the version of the Portal server.
- Choose a default theme.
- Choose a default skin for the chosen theme.

Important: You should not name your project *wps* or anything that resembles this string, in order to avoid internal naming conflicts.

The project that you create with this wizard does not have any portlet definitions, labels, or pages. The themes and skins that are available in this wizard are the same as if you had imported a portal site from a WebSphere Portal server. To create a new Portal Project, do these steps:

- Select **File** → **New** → **Project** → **Portal**.
- Expand **Portal** and select **Portal Project**. Click **Next**.
- In the New Portal Project dialog, enter `MyPortal` in the Project Name field. Use the default path, select the Portal Server V6.0 as your server, select the WebSphere Portal v6.0 as the target runtime environment, enter the name of the EAR (`MyPortalEAR`) project, and click **Next** (Figure 19-4).



Portal Project

Specify a name and location for the new portal project.

Project name: MyPortal

Project contents:

☑ Use default

Directory: C:\workspace\MyPortal

Select the portal server version: 6.0.0.0 (Create from local server)

Creates a Portal project from the resources in the local WebSphere Portal server that you have installed at the specified version level.

Target Runtime

WebSphere Portal v6.0

EAR Membership

☑ Add project to an EAR

EAR Project Name: MyPortalEAR

Figure 19-4 New Portal Project wizard

- In the Select Theme dialog, select the default theme (IBM) and click **Next**.
- In the Select Theme dialog, select the default skin (IBM) and click **Finish**.
- Click **Yes** in the Open Associated Perspective dialog.
- The Portal Designer editor is opened with the selected theme and skin.

Portal Designer

Application Developer allows for editing both the graphic design and portlet layout within your portal. Portal Designer is the Workbench interface that you see upon opening a Portal project.

When using Portal Designer, the portal page layout can be altered. The layout refers to the number of content areas within a page and the number of portlets within those content areas. Page content includes rows, columns, URLs, and portlets.

Once the project is published to the portal server, portal administrators can use the administration portlets to give site users permission to edit the page layout.

In terms of portal layout and appearance, you can think of Portal Designer as a What-You-See-Is-What-You-Get (WYSIWYG) editor. It will render graphic interface items such as themes, skins, and page layouts.

Portal Designer also renders the initial pages of JSF and Struts portlets within your portal pages, but not anything else with regard to portlet content.

Portal Designer provides the capability to alter the layout of the content within a portal page with respect to navigation (the hierarchy of your labels, pages, and URLs) and content (the arrangement of portlets via rows and columns on the portal pages).

`PortalConfiguration` is the name of the layout source file that resides in the root of the Portal project folder (Figure 19-5). To open Portal Designer, double-click the `ibm-portal-topology.xml` file in the Project Explorer.

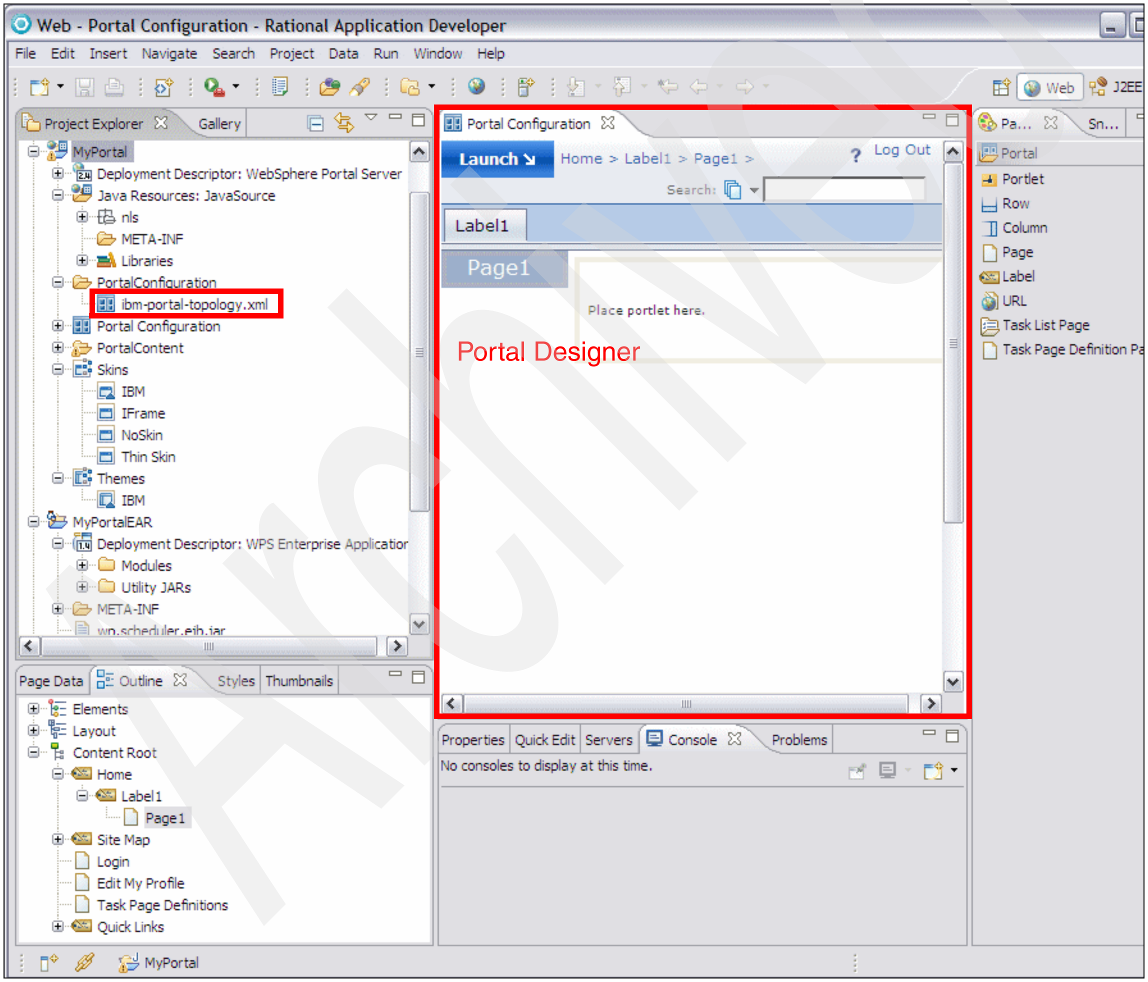


Figure 19-5 Portal Designer Workbench

Skin and theme design and editing

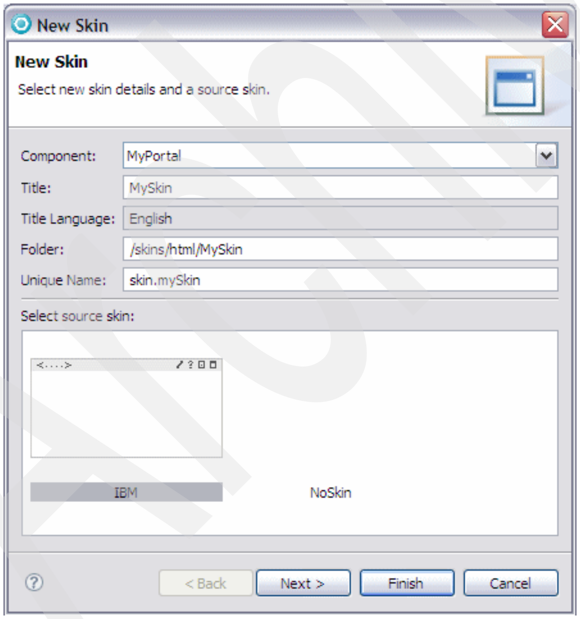
A skin is the border around each portlet within a portal page. Unlike themes, which apply to the overall look and feel of the portal, skins are limited to the look and feel of each portlet that you insert into your portal application.

Application Developer installation includes pre-built themes and skins to use with portal projects. There are also wizards to create new themes and skins. Changing themes and skins was previously done through portal administration. In addition to these wizards for creating new skins and themes, there are tools that can be used to change or edit these.

Once created, skins and themes are displayed in the Project Explorer view. Double-click a skin or theme to manually edit it.

New Skin wizard

In addition to using the pre-made skins that came with the installation, you can use the New Skin wizard to create customized skins for your project (Figure 19-6). Right-click the Portal project in the Project Explorer view and select **New** → **Skin**.



New Skin

New Skin

Select new skin details and a source skin.

Component: MyPortal

Title: MySkin

Title Language: English

Folder: /skins/html/MySkin

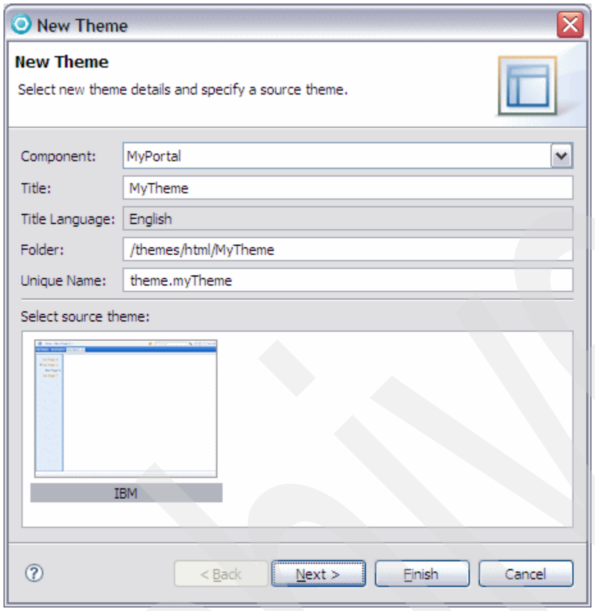
Unique Name: skin.mySkin

Select source skin:

Figure 19-6 New Skin wizard

New Theme wizard

Themes provide the overall look and feel of your portal application. In addition to using the pre-existing themes, you can use the New Theme wizard to create customized themes for your project (Figure 19-7). Right-click the Portal project in the Project Explorer view and select **New** → **Theme**.



New Theme

New Theme

Select new theme details and specify a source theme.

Component: MyPortal

Title: MyTheme

Title Language: English

Folder: /themes/html/MyTheme

Unique Name: theme.myTheme

Select source theme:

IBM

Figure 19-7 New Theme wizard

Developing portal solutions using portal tools

In this section, we give you step-by-step tutorials on how to develop different types of portlets using the various portal tools provided by Application Developer. And along the way, you are also exposed to several important portal and portlet development concepts. By the end of this section, you will know how to do the following tasks:

- Develop create, retrieve, update, and delete (CRUD) Faces portlets that use SDO.
- Consume a Web service in a Faces portlet and display the results using a Faces component.
- Create a Portal project by importing an existing portal solution.
- Modify portal themes and skins.

- Understand portlet-to-portlet communication.
- Develop collaborative portlets.

Creating CRUD Faces portlets using SDO

One of the most common use cases for the manipulation of the back-end database are what are commonly known as CRUD use cases: Create, retrieve, update and delete. In this section we show how to create faces portlets that use SDO to communicate with the database and create the CRUD scenarios.

Application Developer already comes with a sample portlet application that does exactly the same. Therefore, we begin by importing the sample code in our Application Developer workspace. We will explore the code and then run it. After that, we develop two new Portlets and use the tooling available in the Application Developer to:

- Retrieve records from a Derby database table and display the data in the JSF Datatable component.
- Use the JSF Form component to gather data and add a new record in the Derby database.

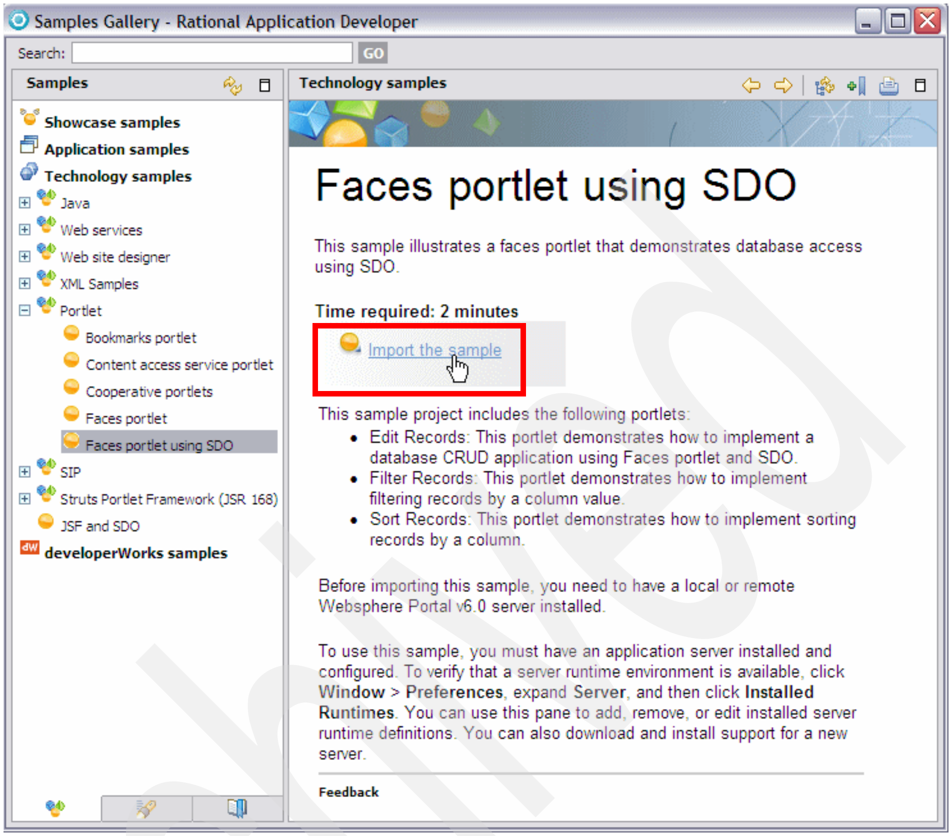
Import the technology sample

To import the technology sample, do these steps:

- Select **Help** → **Samples Gallery**.
- In the Samples Gallery window, select **Technology Samples** → **Portlet** → **Faces portlet using SDO** in the left hand navigation pane (Figure 19-8).
- In the right-hand pane, click **Import the sample** to start importing the sample application to the Application Developer.
- In the Import Faces portlet SDO sample dialog, two projects are listed:

```
FacesPortletSDOExample
FacesPortletSDOExampleEAR
```

- Click **Finish** to import the projects.



Faces portlet using SDO

This sample illustrates a faces portlet that demonstrates database access using SDO.

Time required: 2 minutes

This sample project includes the following portlets:

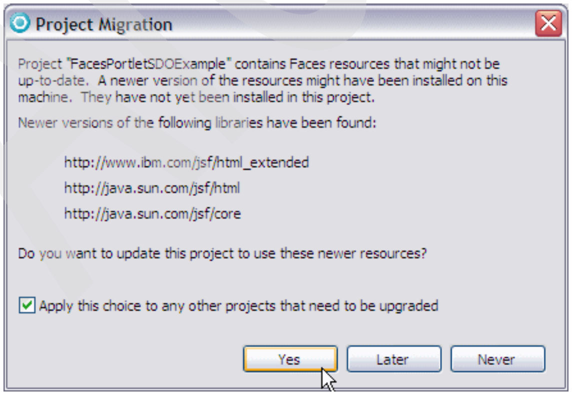
- Edit Records: This portlet demonstrates how to implement a database CRUD application using Faces portlet and SDO.
- Filter Records: This portlet demonstrates how to implement filtering records by a column value.
- Sort Records: This portlet demonstrates how to implement sorting records by a column.

Before importing this sample, you need to have a local or remote Websphere Portal v6.0 server installed.

To use this sample, you must have an application server installed and configured. To verify that a server runtime environment is available, click **Window** > **Preferences**, expand **Server**, and then click **Installed Runtimes**. You can use this pane to add, remove, or edit installed server runtime definitions. You can also download and install support for a new server.

Figure 19-8   Import the technology sample

- ► If you are presented with a Migration Dialog (Figure 19-9), click **Yes**.



Project "FacesPortletSDOExample" contains Faces resources that might not be up-to-date. A newer version of the resources might have been installed on this machine. They have not yet been installed in this project.

Newer versions of the following libraries have been found:

http://www.ibm.com/jsf/html_extended
http://java.sun.com/jsf/html
http://java.sun.com/jsf/core

Do you want to update this project to use these newer resources?

☑ Apply this choice to any other projects that need to be upgraded

Figure 19-9   Project migration with a newer version of Faces resources

- This completes the import of the technology sample that includes an EAR file (`FacesPortletSDOExampleEAR`) and associated Portlet project (`FacesPortletSDOExample`). See Figure 19-10.

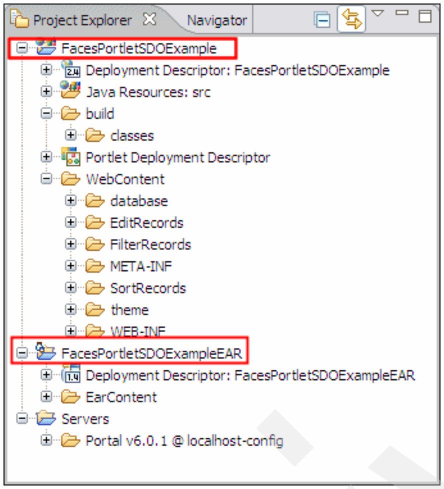

Project Explorer
Navigator
FacesPortletSDOExample
Deployment Descriptor: FacesPortletSDOExample
Java Resources: src
build
classes
Portlet Deployment Descriptor
WebContent
database
EditRecords
FilterRecords
META-INF
SortRecords
theme
WEB-INF
FacesPortletSDOExampleEAR
Deployment Descriptor: FacesPortletSDOExampleEAR
EarContent
Servers
Portal v6.0.1 @ localhost-config

Figure 19-10 Imported Portlet and EAR projects

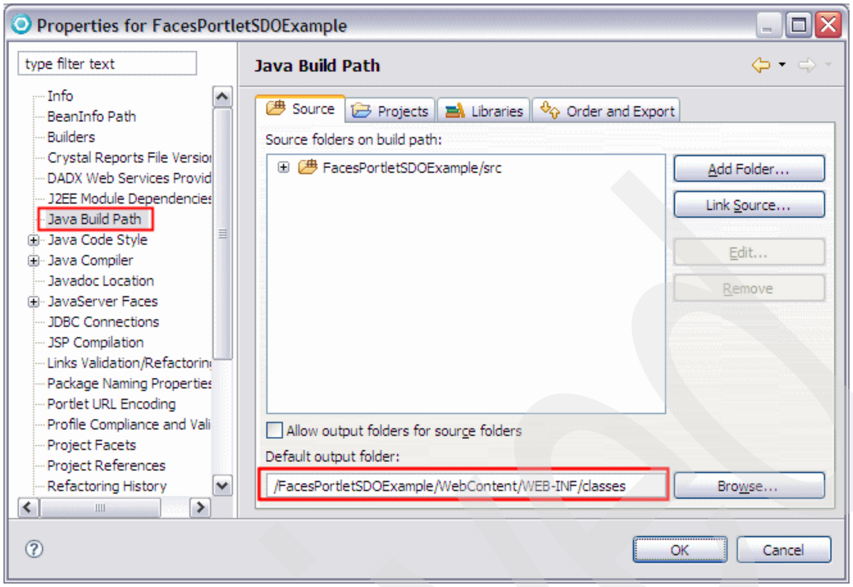
Run the technology sample

Before we run this sample portlet application, we have to complete one last step, and that is to change the default output folder for Portlet project as follows:

- Create an empty folder called `classes` under the `WEB-INF` folder of the `FacesPortletSDOExample` Portlet project. The relative path of this folder should be `/FacesPortletSDOExample/WebContent/WEB-INF/classes`.
- In the Project Explorer, right-click the **FacesPortletSDOExample** Portlet project and select **Properties**.
- In the Properties dialog, select **Java Build Path** and select the **Source** tab (Figure 19-11).
- In the Default output folder field, click **Browse** and locate:

  ```
  /FacesPortletSDOExample/WebContent/WEB-INF/classes
  ```

  Click **OK**.



Source folders on build path:

FacesPortletSDOExample/src

☐ Allow output folders for source folders

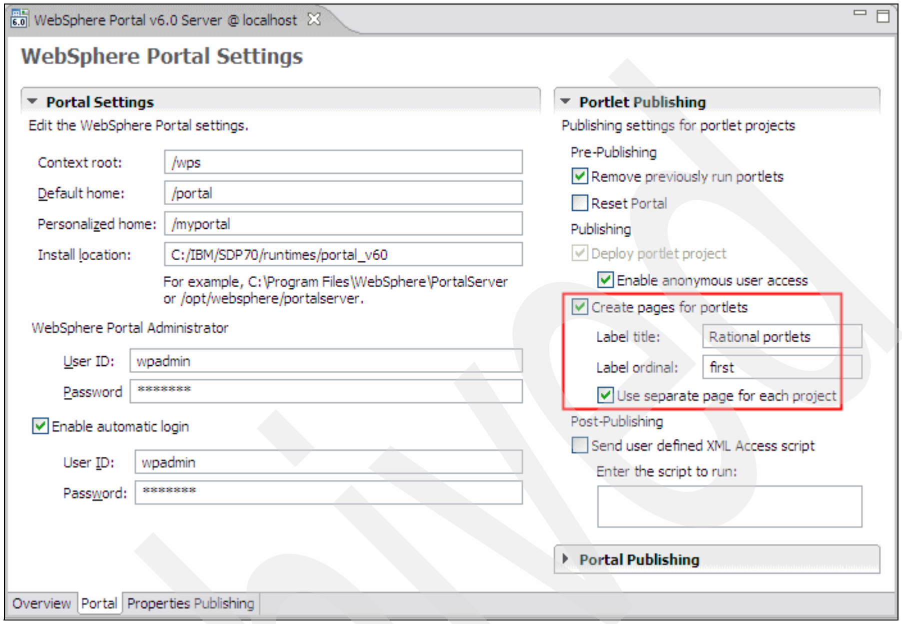
Default output folder: /FacesPortletSDOExample/WebContent/WEB-INF/classes

Figure 19-11   Change the default output folder for the Portlet project

> **Important:** In this step, ensure that you delete the WebSphere Application Server V6.1 from the Servers view and restart the Workbench. The presence of Application Server V6.1 and Portal V6.0 Server causes an issue where the Application Developer cannot sense the proper state of Portal Server V6.x.

- In the Servers view, right-click the **WebSphere Portal V6.0** and select **Publish**. The publishing of Portal server can take some time.
- After publishing has completed, right-click the **FacesPortletSDOExample** Portlet project and select **Run As** → **Run on Server**.
- In the Run on Server dialog, select **WebSphere Portal V6.0** as the server you want to use to run this Portlet project. Also select **Select server as project default (do not ask again)** so you are not presented with this dialog the next time you are trying to run the portlets in this Portlet project. Click **Finish**.
- After the portal server and the `FacesPortletSDOExampleEAR` enterprise application has started, Application Developer opens a browser session in the Workbench view and loads the running Portal application it. A new portal page is created as the first portal page before the usual Portal Welcome page. A child portal page is also created for each Portlet project that displays the initial views of each portlet in that Portlet project.

This is the default behavior while testing the Portlet projects in Application Developer. This can be changed in the configuration settings of the Portal server (Figure 19-12).


WebSphere Portal v6.0 Server @ localhost

WebSphere Portal Settings

Portal Settings

Edit the WebSphere Portal settings.

Context root: /wps

Default home: /portal

Personalized home: /myportal

Install location: C:/IBM/SDP70/runtimes/portal_v60

For example, C:\Program Files\WebSphere\PortalServer or /opt/websphere/portalserver.

WebSphere Portal Administrator

User ID: wpadmin

Password ********

☑ Enable automatic login

User ID: wpadmin

Password: ********

Portlet Publishing

Publishing settings for portlet projects

Pre-Publishing

☑ Remove previously run portlets

☐ Reset Portal

Publishing

☑ Deploy portlet project

☑ Enable anonymous user access

☑ Create pages for portlets

Label title: Rational portlets

Label ordinal: first

☑ Use separate page for each project

Post-Publishing

☐ Send user defined XML Access script

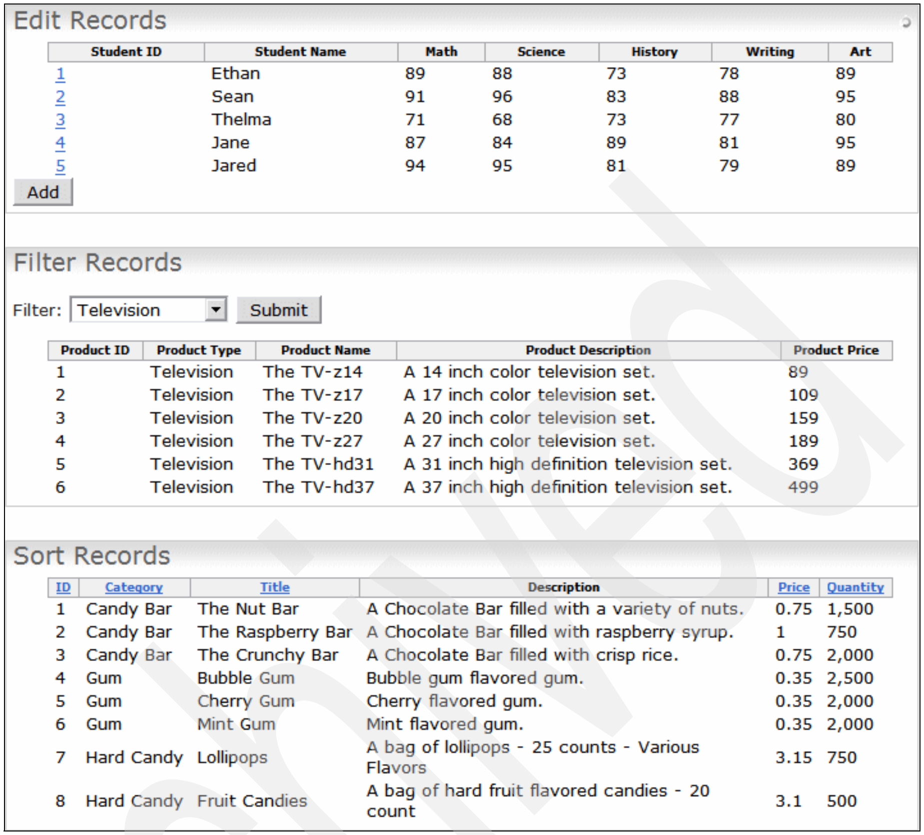
Enter the script to run:

Portal Publishing

Figure 19-12 WebSphere Portal Settings

- For this technology sample, a portal page titled `Rational portlets` is created as the first page. It has a child portal page titled `FacesPortletSDOExample` that displays three portlets: Edit Records, Filter Records. and Sort Records (Figure 19-13).

At this moment, take some time to explore the project and its artifacts, especially the JSF components and their action methods in their page code Java classes. The **Page Data** view for the Portlet JSP pages is also an excellent view to start exploring.



Edit Records

Student ID	Student Name	Math	Science	History	Writing	Art
1	Ethan	89	88	73	78	89
2	Sean	91	96	83	88	95
3	Thelma	71	68	73	77	80
4	Jane	87	84	89	81	95
5	Jared	94	95	81	79	89

Filter Records

Filter: Television

Product ID	Product Type	Product Name	Product Description	Product Price
1	Television	The TV-z14	A 14 inch color television set.	89
2	Television	The TV-z17	A 17 inch color television set.	109
3	Television	The TV-z20	A 20 inch color television set.	159
4	Television	The TV-z27	A 27 inch color television set.	189
5	Television	The TV-hd31	A 31 inch high definition television set.	369
6	Television	The TV-hd37	A 37 inch high definition television set.	499

Sort Records

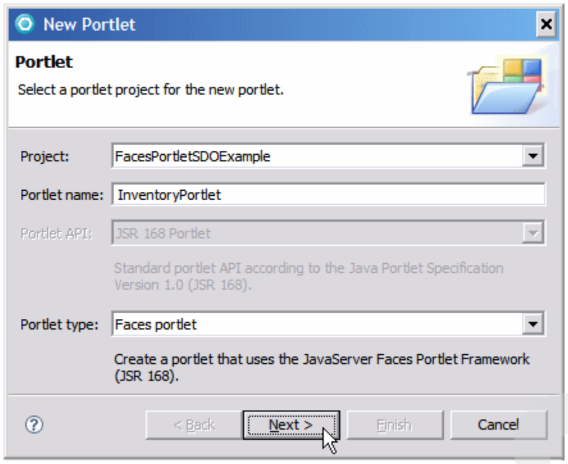
ID	Category	Title	Description	Price	Quantity
1	Candy Bar	The Nut Bar	A Chocolate Bar filled with a variety of nuts.	0.75	1,500
2	Candy Bar	The Raspberry Bar	A Chocolate Bar filled with raspberry syrup.	1	750
3	Candy Bar	The Crunchy Bar	A Chocolate Bar filled with crisp rice.	0.75	2,000
4	Gum	Bubble Gum	Bubble gum flavored gum.	0.35	2,500
5	Gum	Cherry Gum	Cherry flavored gum.	0.35	2,000
6	Gum	Mint Gum	Mint flavored gum.	0.35	2,000
7	Hard Candy	Lollipops	A bag of lollipops - 25 counts - Various Flavors	3.15	750
8	Hard Candy	Fruit Candies	A bag of hard fruit flavored candies - 20 count	3.1	500

Figure 19-13 Three faces portlets using SDO technology sample

Create a new Faces portlet that uses SDO to retrieve records

In this section we create a new Faces portlet similar to the Sort Records portlet that retrieves the records from the `INVENTORYSORT` table from the Derby database using SDO and then display it in a JSF Datatable component.

- In the Project Explorer, expand the **FacesPortletSDOExample** Portlet project.
- Right-click the **Portlet Deployment Descriptor** and select **New** → **Portlet**.
- In the New Portlet dialog, select **FacesPortletSDOExample** as the Project, **Faces portlet** as the Portlet type and type **InventoryPortlet** as the Portlet name (Figure 19-14). Click **Next.**



Portlet

Select a portlet project for the new portlet.

Project: FacesPortletSDOExample

Portlet name: InventoryPortlet

Portlet API: JSR 168 Portlet

Standard portlet API according to the Java Portlet Specification Version 1.0 (JSR 168).

Portlet type: Faces portlet

Create a portlet that uses the JavaServer Faces Portlet Framework (JSR 168).

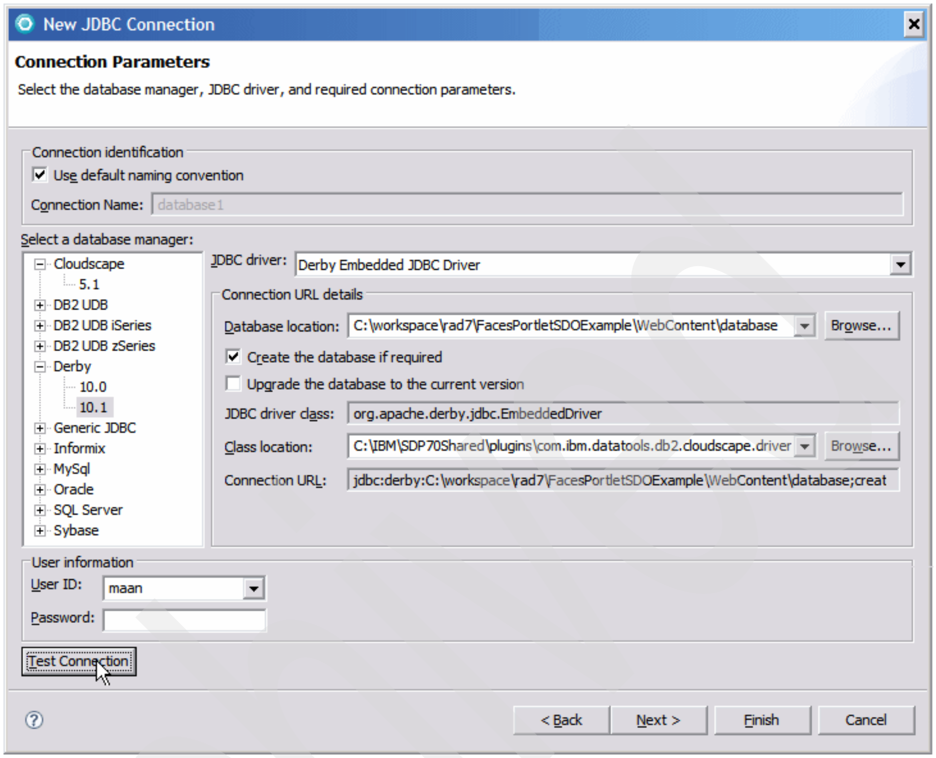
Figure 19-14 Create a new Faces Portlet in the FacesPortletSDOExample

- In the Portlet Settings dialog, click **Finish**.

  This creates an empty Portlet with Faces support in the existing `FacesPortletSDOExample` Portlet project.

- The view JSP (`InventoryPortletView.jsp`) file is opened automatically in the Page Designer.
- In the Page Data view, right-click the **Relational Records** and select **New** → **Relational Record List**.
- In the Add Relational Record List dialog, enter **inventoryList** as the reference name for the Record list and click **Next**.
- In the Record List Properties dialog, click **New** to create a new JDBC connection.
- In the New JDBC Connection dialog, select **Create a new connection** and click **Next**.
- In the Connection Parameters dialog (Figure 19-15), do these steps:
  - Select Derby 10.1 as the Database Manager.
  - For the Database location field click **Browse** and locate `C:\<workspace_dir>\FacesPortletSDOExample\WebContent\database`. Note that the sample database has already created when you import the samples gallery in your workspace.
  - For the Class location field, click **Browse** and locate `C:/<rad_install_dir>/plugins/com.ibm.datatools.db2.cloudscape.driver_1.0.0.v200610121320/driver/derby.jar`.
  - In the User ID field, you can enter your login user id and no password.

– Click **Test Connection**. If the test is successful, click **Finish**.



New JDBC Connection

Connection Parameters

Select the database manager, JDBC driver, and required connection parameters.

Connection identification

☑ Use default naming convention

Connection Name: database1

Select a database manager:

- Cloudscape
  - 5.1
- DB2 UDB
- DB2 UDB iSeries
- DB2 UDB zSeries
- Derby
  - 10.0
  - 10.1
- Generic JDBC
- Informix
- MySql
- Oracle
- SQL Server
- Sybase

JDBC driver: Derby Embedded JDBC Driver

Connection URL details

Database location: C:\workspace\rad7\FacesPortletSDOExample\WebContent\database

☑ Create the database if required

☐ Upgrade the database to the current version

JDBC driver class: org.apache.derby.jdbc.EmbeddedDriver

Class location: C:\IBM\SDP70Shared\plugins\com.ibm.datatools.db2.cloudscape.driver

Connection URL: jdbc:derby:C:\workspace\rad7\FacesPortletSDOExample\WebContent\database;creat

User information

User ID: maan

Password:

Figure 19-15 Connection Parameters for the new JDBC connection

- In the Record List Properties dialog, expand **CANDYFACTORY** schema and select **INVENTORYSORT** table. Click **Next** (Figure 19-16).

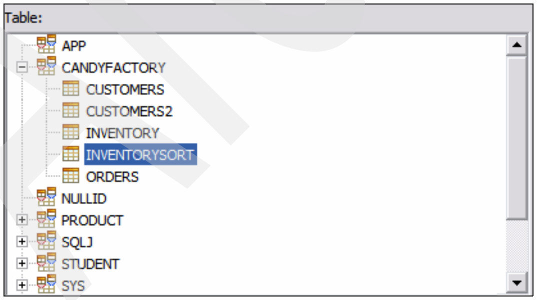


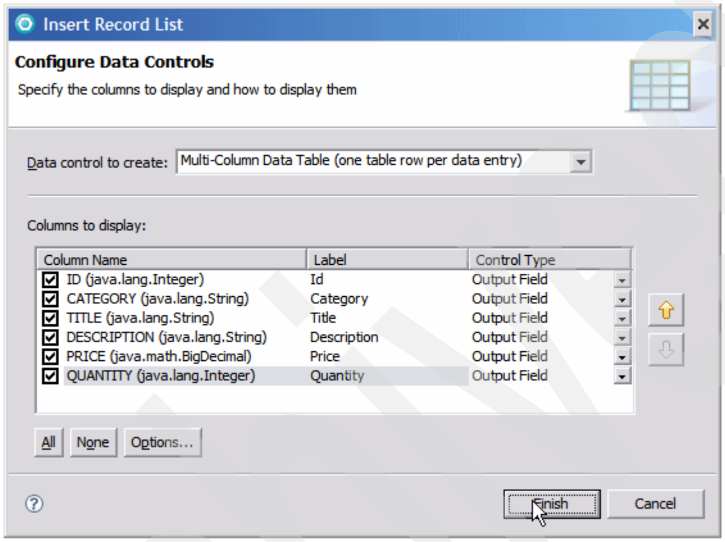
Table:

- APP
- CANDYFACTORY
  - CUSTOMERS
  - CUSTOMERS2
  - INVENTORY
  - INVENTORYSORT
  - ORDERS
- NULLID
- PRODUCT
- SQLJ
- STUDENT
- SYS

Figure 19-16 Select the database table

- In the Add Relational Record dialog, select the columns that you want to display in your view. Click **Finish**.

- In the Page Data view, expand **Relational Records** → **inventoryList (Service Data Object)**, select the **inventoryList (INVENTORYSORT)** relational record component, and drag it onto the page. The help-tip will show the text `Drop here to insert controls for "inventoryList"`.
- In the Configure Data Controls dialog, select **Displaying data**, specify which columns to display in the JSF Datatable component and how to display them (Figure 19-17). Click **Finish**.



Insert Record List

Configure Data Controls

Specify the columns to display and how to display them

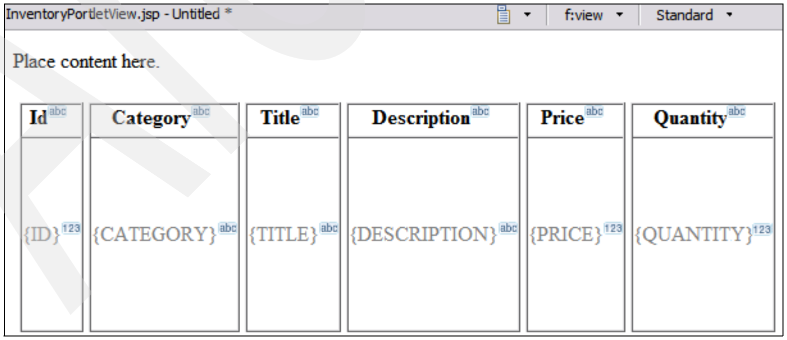
Data control to create: Multi-Column Data Table (one table row per data entry)

Columns to display:

Column Name	Label	Control Type
☑ ID (java.lang.Integer)	Id	Output Field
☑ CATEGORY (java.lang.String)	Category	Output Field
☑ TITLE (java.lang.String)	Title	Output Field
☑ DESCRIPTION (java.lang.String)	Description	Output Field
☑ PRICE (java.math.BigDecimal)	Price	Output Field
☑ QUANTITY (java.lang.Integer)	Quantity	Output Field

Figure 19-17 Specify which columns to display

- This generates the necessary code and places a JSF Datatable component on the JSP page with the selected columns (Figure 19-18).



InventoryPortletView.jsp - Untitled *

Place content here.

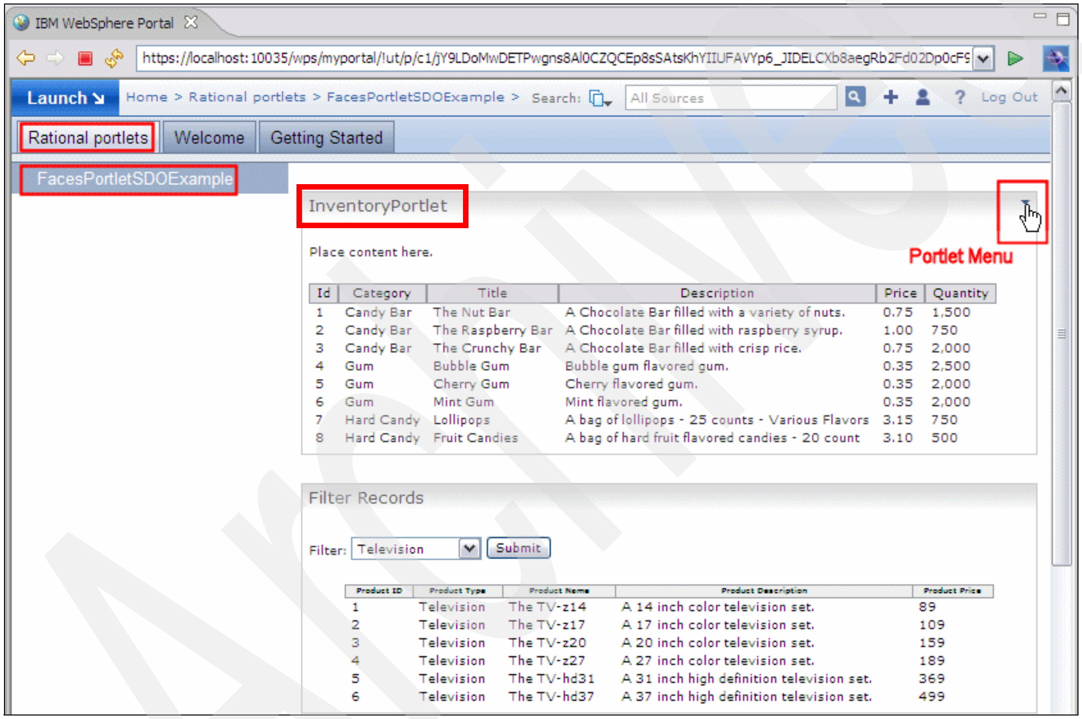
Id	Category	Title	Description	Price	Quantity
{ID}	{CATEGORY}	{TITLE}	{DESCRIPTION}	{PRICE}	{QUANTITY}

Figure 19-18 InventoryPortletView.jsp with inventoryList Relational Record List.

- In the Project Explorer, right-click `InventoryPortletView.jsp` and select **Run As → Run on Server**. The new InventoryPortlet portlet is rendered along with the other three existing portlets (Figure 19-19).

Again, you can notice that a PTE portlets page has been created by the Portal server as the first portal page. This page has a child portal page titled same as the name of the Portlet project and all the four portlets (one new portlet and three existing portlets) of this Portlet project are rendered on this page.

If you are wondering why the new portlet is rendered at the top, actually it was not. By default, it would render at the bottom. However, by using the Portlet Menu, we moved it to the top.


IBM WebSphere Portal
https://localhost:10035/wps/myportal/!ut/p/c1/jY9LDoMwDETPwgns8Al0CZQCEp8sSAtsKhYIIUFAVYp6_JIDELCXb8aegRb2Fd02Dp0cF
Launch
Home > Rational portlets > FacesPortletSDOExample >
Search:
All Sources
Log Out
Rational portlets
Welcome
Getting Started
FacesPortletSDOExample
InventoryPortlet
Portlet Menu

Place content here.

Id	Category	Title	Description	Price	Quantity
1	Candy Bar	The Nut Bar	A Chocolate Bar filled with a variety of nuts.	0.75	1,500
2	Candy Bar	The Raspberry Bar	A Chocolate Bar filled with raspberry syrup.	1.00	750
3	Candy Bar	The Crunchy Bar	A Chocolate Bar filled with crisp rice.	0.75	2,000
4	Gum	Bubble Gum	Bubble gum flavored gum.	0.35	2,500
5	Gum	Cherry Gum	Cherry flavored gum.	0.35	2,000
6	Gum	Mint Gum	Mint flavored gum.	0.35	2,000
7	Hard Candy	Lollipops	A bag of lollipops - 25 counts - Various Flavors	3.15	750
8	Hard Candy	Fruit Candies	A bag of hard fruit flavored candies - 20 count	3.10	500

Filter: Television

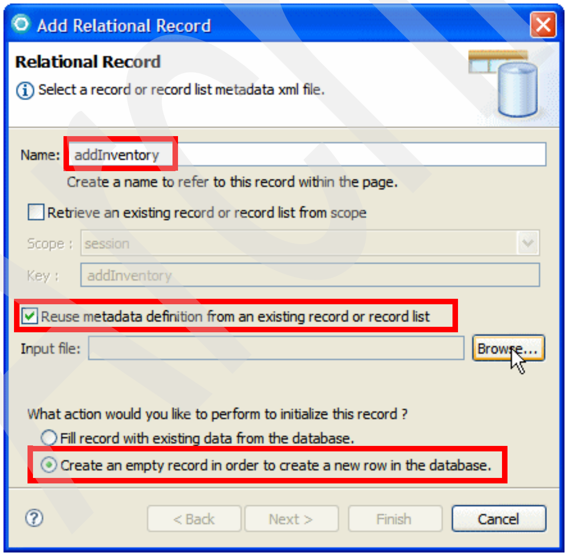
Product ID	Product Type	Product Name	Product Description	Product Price
1	Television	The TV-z14	A 14 inch color television set.	89
2	Television	The TV-z17	A 17 inch color television set.	109
3	Television	The TV-z20	A 20 inch color television set.	159
4	Television	The TV-z27	A 27 inch color television set.	189
5	Television	The TV-hd31	A 31 inch high definition television set.	369
6	Television	The TV-hd37	A 37 inch high definition television set.	499

Figure 19-19 The new InventoryPortlet renders records of the INVENTORYSORT table

Create a Faces portlet that uses SDO to add a record

In this section we create a Faces portlet that creates a new record in the `INVENTORYSORT` table. Again, let us create the new Faces portlet in the existing `FacesPortletSDOExample` Portlet project. Currently this Portlet project has four portlets: Edit Records, Filter Records, Sort Records, and the InventoryPortlet that we created in the last section.

- In the Project Explorer, expand the **FacesPortletSDOExample** Portlet project, right-click **Portlet Deployment Descriptor**, and select **New** → **Portlet**.
- In the New Portlet dialog, select **FacesPortletSDOExample** as the project, **Faces portlet** as the Portlet type, and enter **AddInventoryPortlet** as the Portlet name. Click **Next**.
- In the Portlet Settings dialog, click **Finish**. This creates an empty Portlet with Faces support in the existing `FacesPortletSDOExample` Portlet project.
- The view JSP (`AddInventoryPortletView.jsp`) file is opened in Page Designer.
- In the Page Data view for this JSP page, right-click the **Relational Records** and select **New** → **Relational Record**.
- In the Add Relational Record dialog, enter **addInventory** in the Name field and select **Create an empty record** (Figure 19-20).
- We want to reuse the JDBC connection and table definition we created for the last portlet. Select **Reuse metadata definition from an existing record or record list**.
- For the input file, click **Browse** to locate `/WEB-INF/wdo/inventoryList.xml`. This file is generated when we create the record list for the InventoryPortlet portlet. Click **Next**.



Relational Record

Select a record or record list metadata xml file.

Name: addInventory

Create a name to refer to this record within the page.

□ Retrieve an existing record or record list from scope

Scope : session

Key : addInventory

☑ Reuse metadata definition from an existing record or record list

Input file:

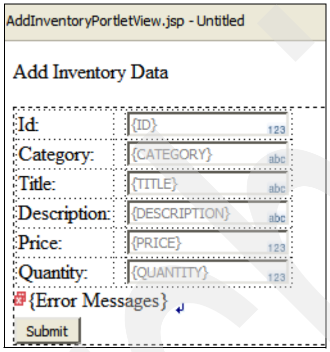
What action would you like to perform to initialize this record ?

○ Fill record with existing data from the database.

● Create an empty record in order to create a new row in the database.

Figure 19-20 Reuse the metadata definition from the previous record list

- In the Database Authorization dialog, enter any userid and leave the password black and click **OK**.
- In the Record Properties dialog, select the connection that you created for the record list in the last section. Click **Finish**.
- In the Page Data view, expand **Relational Records** → **addInventory (Service Data Object)**, select the **addInventory (INVENTORYSORT)** relational record component, and drag it onto the page. The help-tip will show the text `Drop here to insert controls for` “`addInventory`”.
- In the Configure Data Controls dialog, select **Inputting data**, and specify which columns to display in the JSF Form component and how to display them (input fields). Click **Finish**.
- This generates the necessary code and place a new JSF Form component on the JSP page with required Input Fields. You might also want to change the text at the top of the page to something more relevant (Figure 19-21).



AddInventoryPortletView.jsp - Untitled

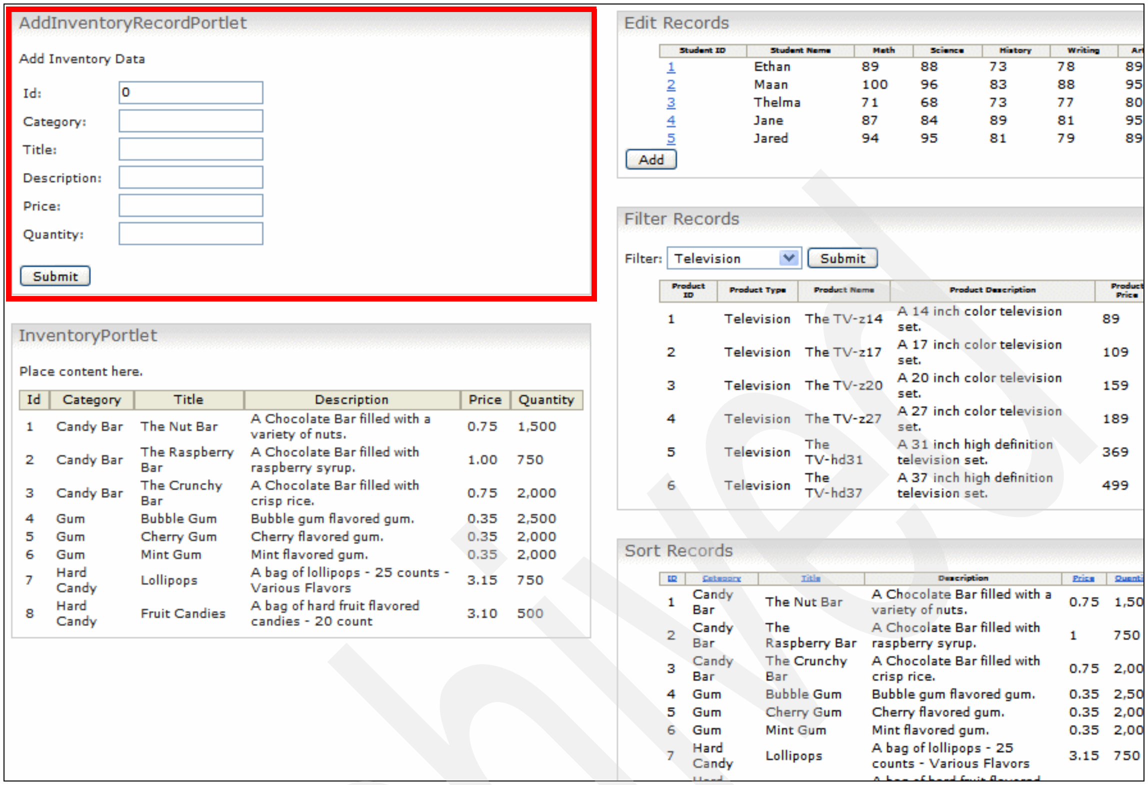
Add Inventory Data

Id:	{ID}
Category:	{CATEGORY}
Title:	{TITLE}
Description:	{DESCRIPTION}
Price:	{PRICE}
Quantity:	{QUANTITY}

{Error Messages}

Figure 19-21 JSF Form component added

- From the Project Explorer view, right-click `AddInventoryPortletView.jsp` and select **Run As** → **Run on Server**. The new AddInventoryPortlet portlet is rendered along with the other four existing portlets (Figure 19-22).



AddInventoryRecordPortlet

Add Inventory Data

Id:	0
Category:	
Title:	
Description:	
Price:	
Quantity:	

InventoryPortlet

Place content here.

Id	Category	Title	Description	Price	Quantity
1	Candy Bar	The Nut Bar	A Chocolate Bar filled with a variety of nuts.	0.75	1,500
2	Candy Bar	The Raspberry Bar	A Chocolate Bar filled with raspberry syrup.	1.00	750
3	Candy Bar	The Crunchy Bar	A Chocolate Bar filled with crisp rice.	0.75	2,000
4	Gum	Bubble Gum	Bubble gum flavored gum.	0.35	2,500
5	Gum	Cherry Gum	Cherry flavored gum.	0.35	2,000
6	Gum	Mint Gum	Mint flavored gum.	0.35	2,000
7	Hard Candy	Lollipops	A bag of lollipops - 25 counts - Various Flavors	3.15	750
8	Hard Candy	Fruit Candies	A bag of hard fruit flavored candies - 20 count	3.10	500

Edit Records

Student ID	Student Name	Math	Science	History	Writing	Ar
1	Ethan	89	88	73	78	89
2	Maan	100	96	83	88	95
3	Thelma	71	68	73	77	80
4	Jane	87	84	89	81	95
5	Jared	94	95	81	79	89

Filter Records

Filter: Television

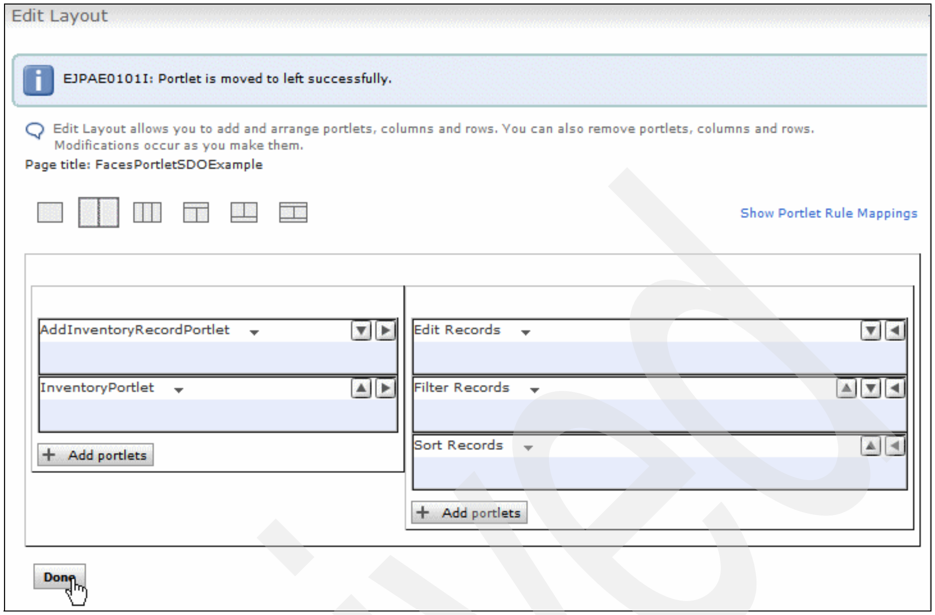
Product ID	Product Type	Product Name	Product Description	Product Price
1	Television	The TV-z14	A 14 inch color television set.	89
2	Television	The TV-z17	A 17 inch color television set.	109
3	Television	The TV-z20	A 20 inch color television set.	159
4	Television	The TV-z27	A 27 inch color television set.	189
5	Television	The TV-hd31	A 31 inch high definition television set.	369
6	Television	The TV-hd37	A 37 inch high definition television set.	499

Sort Records

ID	Category	Title	Description	Price	Quanti
1	Candy Bar	The Nut Bar	A Chocolate Bar filled with a variety of nuts.	0.75	1,50
2	Candy Bar	The Raspberry Bar	A Chocolate Bar filled with raspberry syrup.	1	750
3	Candy Bar	The Crunchy Bar	A Chocolate Bar filled with crisp rice.	0.75	2,00
4	Gum	Bubble Gum	Bubble gum flavored gum.	0.35	2,50
5	Gum	Cherry Gum	Cherry flavored gum.	0.35	2,00
6	Gum	Mint Gum	Mint flavored gum.	0.35	2,00
7	Hard Candy	Lollipops	A bag of lollipops - 25 counts - Various Flavors	3.15	750

Figure 19-22 All the five portlets in the FacesPortletSDOExample Portlet project

- Again, as you can notice, we have changed the layout of the page using the Edit Layout portlet for the new page from the Portal administration console. (Figure 19-23).



Edit Layout

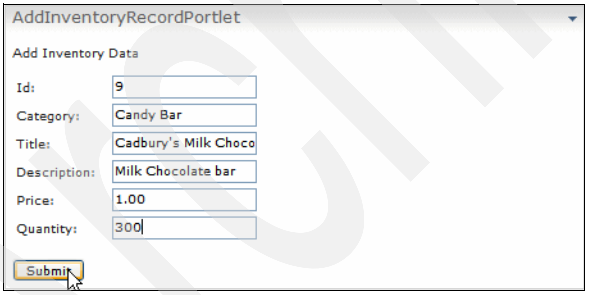
EJPAE0101I: Portlet is moved to left successfully.

Edit Layout allows you to add and arrange portlets, columns and rows. You can also remove portlets, columns and rows. Modifications occur as you make them.

Page title: FacesPortletSDOExample

Figure 19-23 Change the layout of the Portal page

- Test the new AddInventoryRecord portlet by adding a record (Figure 19-24).



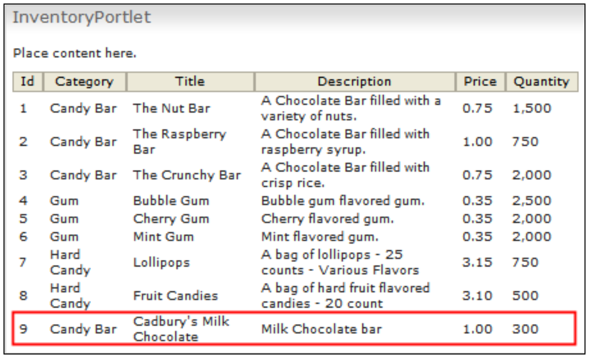
AddInventoryRecordPortlet

Add Inventory Data

Id:	9
Category:	Candy Bar
Title:	Cadbury's Milk Choco
Description:	Milk Chocolate bar
Price:	1.00
Quantity:	300

Figure 19-24 Submit a new Record

- This record appears in the InventoryPortlet at the bottom (Figure 19-25).



InventoryPortlet

Place content here.

Id	Category	Title	Description	Price	Quantity
1	Candy Bar	The Nut Bar	A Chocolate Bar filled with a variety of nuts.	0.75	1,500
2	Candy Bar	The Raspberry Bar	A Chocolate Bar filled with raspberry syrup.	1.00	750
3	Candy Bar	The Crunchy Bar	A Chocolate Bar filled with crisp rice.	0.75	2,000
4	Gum	Bubble Gum	Bubble gum flavored gum.	0.35	2,500
5	Gum	Cherry Gum	Cherry flavored gum.	0.35	2,000
6	Gum	Mint Gum	Mint flavored gum.	0.35	2,000
7	Hard Candy	Lollipops	A bag of lollipops - 25 counts - Various Flavors	3.15	750
8	Hard Candy	Fruit Candies	A bag of hard fruit flavored candies - 20 count	3.10	500
9	Candy Bar	Cadbury's Milk Chocolate	Milk Chocolate bar	1.00	300

Figure 19-25 A new record is added

Consuming a Web service in a portlet and displaying the results using a Faces component

In this section we first deploy a Web service to a WebSphere Application Server V6.1 and then create a portlet that acts as a client to this Web service and consume its operations and display the data retrieved using JSF components.

This Web service is basically the JavaBean Web service developed in “Creating Web services from a JavaBean” on page 820.

Deploy the Web service

To deploy the Web service, do these steps:

- Import the project interchange file from:

  `c:\7501code\portal\webservice\PortalService.zip`

- This action imports three projects:
  - `RAD7PortalServiceEAR`—Enterprise application
  - `RAD7PortalServiceUtility`—Utility code with banking model
  - `RAD7PortalServiceWeb`—Web project with JavaBean Web service

Tailor the WSDL file

The Web service has to run on the portal server. In every installation, the Web port of the Portal Server might be different. Do these steps:

- Locate and edit the file:

  `RAD7PortalServiceWeb\WebContent\WEB-INF\wsdl\SimpleBankBean.wsdl`

- At the bottom of the file, edit the location with the correct hostname and port (Example 19-1).

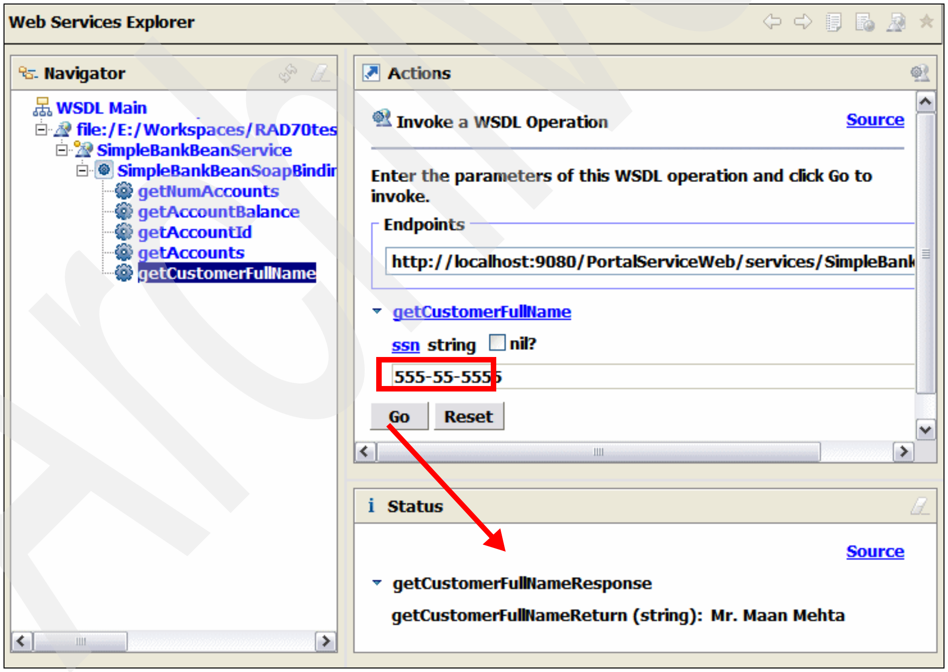
Example 19-1 SimpleBankBean.WSDL - Web service endpoint address

```
<wsdl:service name="SimpleBankBeanService">
   <wsdl:port binding="intf:SimpleBankBeanSoapBinding" name="SimpleBankBean">
      <wsdlsoap:address location="http://<localhost/hostname>:<port>/
                                   PortalServiceWeb/services/SimpleBankBean"/>
   </wsdl:port>
</wsdl:service>
```

Add the Web service to the Portal Server and test

We install the Web service in Portal Server and test it using the Web Services Explorer:

- Right-click the Portal Server in the Servers view and select **Add and Remove Projects**. Select **RAD7PortalServiceEAR** and click **Add >**, then click **Finish**.
- Test the Web service with Web Services Explorer (Figure 19-26). Right-click the `SimpleBankBean.wsdl` file and select **Web Services** → **Test with Web Services Explorer**. Expand the service in the left pane, select the `getCustomerFullname` operation, enter an SSN, and click **Go**.



Invoke a WSDL Operation

Enter the parameters of this WSDL operation and click Go to invoke.

Endpoints

http://localhost:9080/PortalServiceWeb/services/SimpleBank

getCustomerFullName

ssn string □ nil?

555-55-5555

getCustomerFullNameResponse

getCustomerFullNameReturn (string): Mr. Maan Mehta

Figure 19-26 Testing the deployed Web service using the Web Services Explorer

- You can also test the deployment in a Web browser using the following URL to view the contents of the WSDL file.

  ```
  http://<hostname>:<port>/PortalServiceWeb/services/SimpleBankBean?wsdl
  ```

Use case: What do we want to build?

As you can tell from the contents of the WSDL file, or might observe when you are testing the Web service through the Web Services Explorer, the author of this Web service is providing the following five operations (Table 19-1).

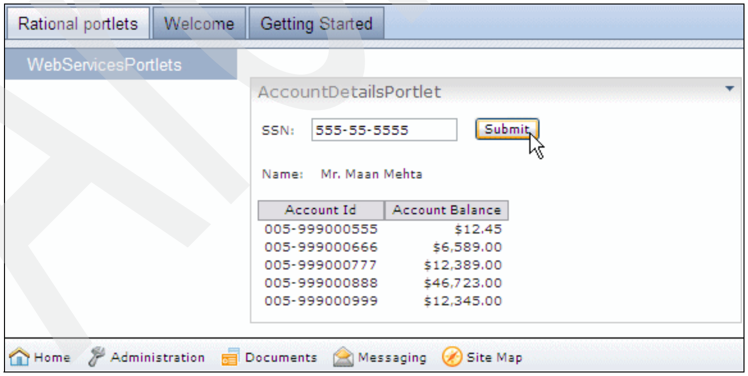
Table 19-1 Web service operations

Operation	Input	Output
`getNumAccounts`	`CustomerId (xsd:string)`	# of accounts `(xsd:int)`
`getAccountBalance`	`CustomerId (xsd:string)`	Account balance `(xsd:int)`

`getAccountId`	`CustomerId (xsd:string),`
	`Account (xsd:int)`
		AccountId `(xsd:string)`

`getAccounts`	`CustomerId (xsd:string)`	Array of `Account` `Account` is the complex object consisting of `id` and `balance`
`getCustomerFullName`	`SSN (xsd:string)`	Full Name (xsd:string)

We want to create a Faces portlet that takes the social security number (SSN) as an input and returns the full name of the customer, a list of all the accounts, and the account balance, in a tabular format. The desired display should look somewhat as shown in Figure 19-27.



SSN: 555-55-5555

Name: Mr. Maan Mehta

Account Id	Account Balance
005-999000555	$12.45
005-999000666	$6,589.00
005-999000777	$12,389.00
005-999000888	$46,723.00
005-999000999	$12,345.00

Figure 19-27 Get account details

Design details

Just looking at the desired view and based on the use case requirements, we can quickly tell that Datatable and Form JSF components are used in the view JSP of the portlet.

Also, based on the details of operations of the Web services, we can easily determine that we have to use the following two operations:

- `getCustomerFullName`
- `getAccounts`

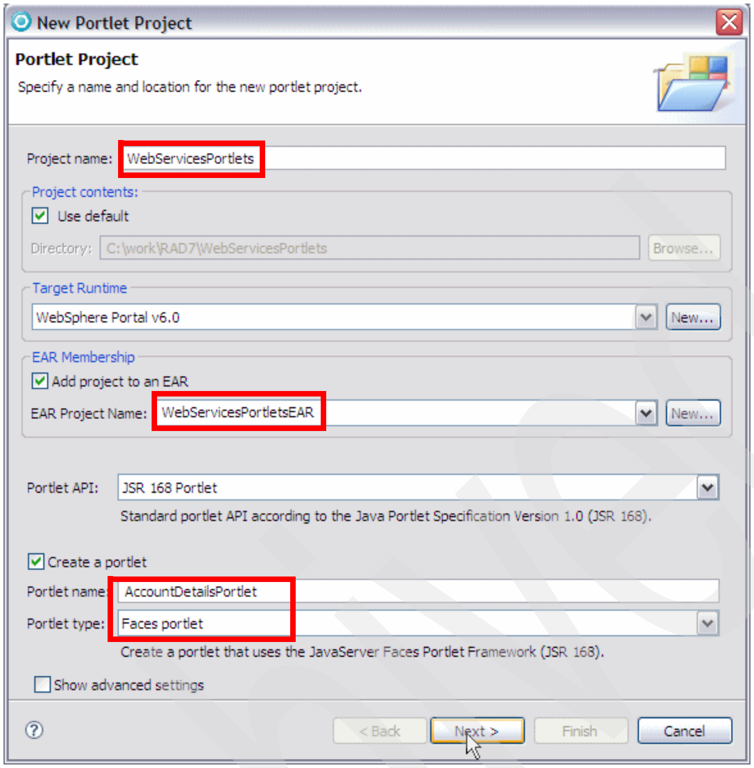
Thus we have to make two Web services calls from the portlet and display the result using the Faces components. That sums up the design discussion for this use case; no need of class or sequence diagrams. This is one of the advantages when you have chosen framework and technologies such as Portal and JSF and using Web services to deliver the application logic and the data.

Create a Faces portlet that calls the getFullName operation

In this section, we use Portal tooling available in the Application Developer to create a Faces portlet that calls the `getFullName` operation of the Web service and display the result in its view JSP. As in the previous exercise, we will not be writing even a single line of code, and we will build a small SOA use case.

To create a new Faces Portlet Project, do these steps:

- In the Project Explorer, select **File** → **New** → **Portlet Project**.
- In the New Portlet Project dialog, do these steps and click **Next** (Figure 19-28):
  - Project name: **WebServicesPortlets**
  - Select **Create a Portlet**
  - Portlet name: **AccountDetailsPortlet**
  - Portlet type: **Faces portlet**



New Portlet Project

Portlet Project

Specify a name and location for the new portlet project.

Project name: WebServicesPortlets

Project contents:

☑ Use default

Directory: C:\work\RAD7\WebServicesPortlets [Browse...]

Target Runtime

WebSphere Portal v6.0 [New...]

EAR Membership

☑ Add project to an EAR

EAR Project Name: WebServicesPortletsEAR [New...]

Portlet API: JSR 168 Portlet

Standard portlet API according to the Java Portlet Specification Version 1.0 (JSR 168).

☑ Create a portlet

Portlet name: AccountDetailsPortlet

Portlet type: Faces portlet

Create a portlet that uses the JavaServer Faces Portlet Framework (JSR 168).

□ Show advanced settings

Figure 19-28 Create a new Portlet project with an empty Portlet

- In the Portlet Settings dialog, click **Finish**.
- The view JSP (`AccountDetailsPortletView.jsp`) file is opened automatically in Page Designer.
- Open the Data drawer in the Palette view and drag and drop the Web Service component to the `AccountDetailsPortletView.jsp` (Figure 19-29). This initiates a Web Services Discovery Dialog.

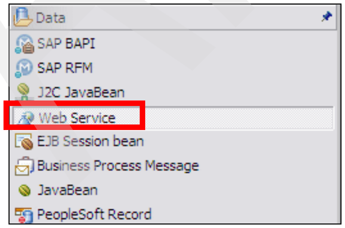
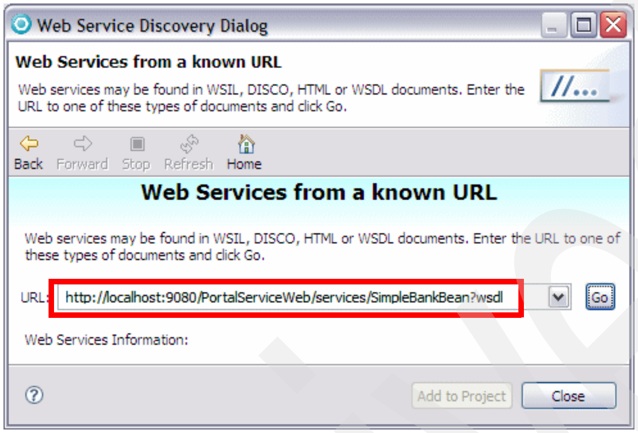


Figure 19-29 Drag the Web Service component to the JSP to start the Web Services Discovery dialog

- In the Web Services Discovery dialog, select **Web Services from a known URL**.
- In the Web Services from a known URL dialog, enter the URL and click **Go** (Figure 19-30):

```
http://<WS_SERVER>:<port>/PortalServiceWeb/services/SimpleBankBean?wsdl
```



Web Service Discovery Dialog

Web Services from a known URL

Web services may be found in WSIL, DISCO, HTML or WSDL documents. Enter the URL to one of these types of documents and click Go.

Web Services from a known URL

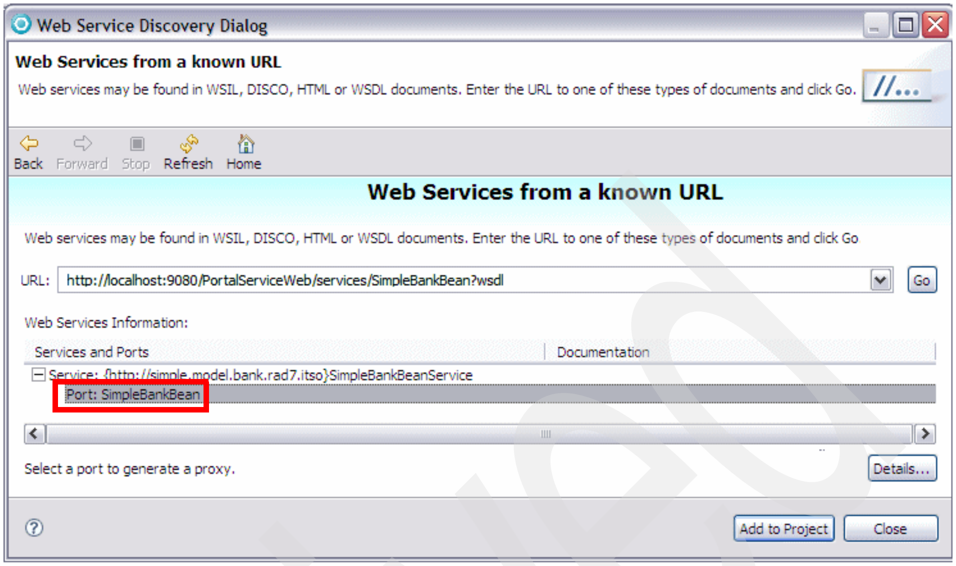
Web services may be found in WSIL, DISCO, HTML or WSDL documents. Enter the URL to one of these types of documents and click Go.

URL: http://localhost:9080/PortalServiceWeb/services/SimpleBankBean?wsdl

Web Services Information:

Figure 19-30 Enter the WSDL URL

- Select the Web services port (`Port:SimpleBankBean`) and click **Add to Project** (Figure 19-31).
- Click **Yes** in the Warning dialog to overwrite the `web.xml` file, if necessary.



Web Services from a known URL

Web services may be found in WSIL, DISCO, HTML or WSDL documents. Enter the URL to one of these types of documents and click Go.

Web Services from a known URL

Web services may be found in WSIL, DISCO, HTML or WSDL documents. Enter the URL to one of these types of documents and click Go

URL: http://localhost:9080/PortalServiceWeb/services/SimpleBankBean?wsdl

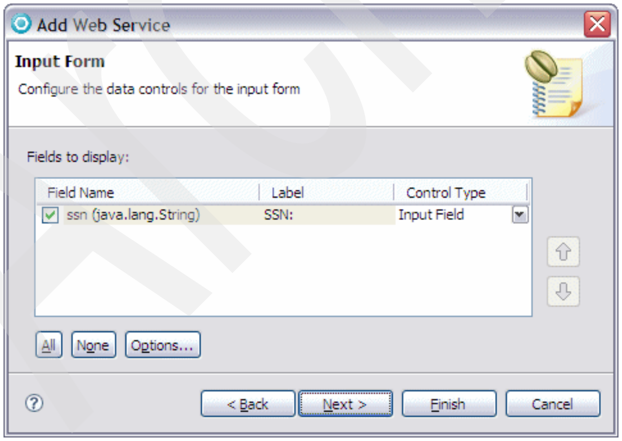
Web Services Information:

Services and Ports	Documentation
Service: {http://simple.model.bank.rad7.itso}SimpleBankBeanService	
Port: SimpleBankBean	

Select a port to generate a proxy.

Figure 19-31 Add the Web service to the project

- ► In the Add Web Service dialog, select the **getCustomerFullName** operation and click **Next**.
- ► In the Input Form dialog, select `ssn` that will be shown on the input HTML form (Figure 19-32). You can also change the label of this field and also configure its look and feel by clicking **Options**. Click **Next**.



Input Form

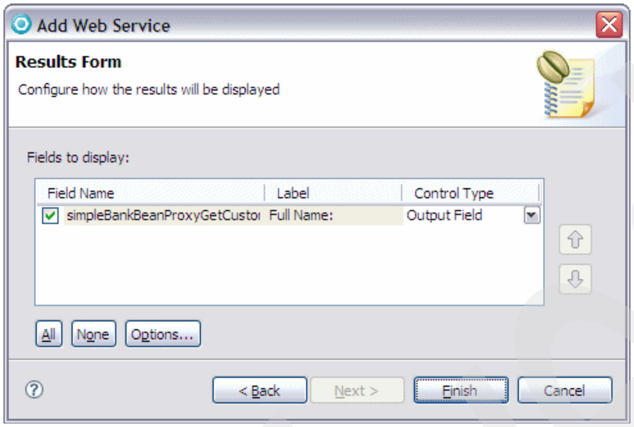
Configure the data controls for the input form

Fields to display:

Field Name	Label	Control Type
☑ ssn (java.lang.String)	SSN:	Input Field

Figure 19-32 Configure the input field

- In the Results Form dialog, select which output field will be shown (Figure 19-33). You can also change the label of this field and also configure its look and feel by clicking **Options**. Click **Finish**.



Results Form

Configure how the results will be displayed

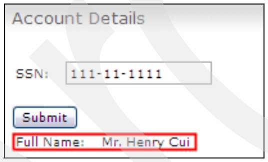
Fields to display:

Field Name	Label	Control Type
☑ simpleBankBeanProxyGetCusto	Full Name:	Output Field

Figure 19-33 Configure the output field

- This completes the process of adding all the necessary input and output fields to the view JSP and generating the code required to communicate with the `getCustomerFullName` method of the Web service.
- Run and test the Portlet project (Figure 19-34).

  Notice that you do not have to write a single line of code, and that the tooling generated all the necessary code.



Account Details

SSN: 111-11-1111

Full Name: Mr. Henry Cui

Figure 19-34 Testing the getCustomerFullName operation

Add the second operation (getAccounts) to the portlet

In this section, we use the same portlet and add the input and output JSF controls needed to execute and retrieve data from the `getAccounts` operation of the Web service. The process is quite similar to what we have been doing so far:

- Drag and drop the Web Service component from the Data drawer in the Palette view to the `AccountDetailsPortletView.jsp` (Figure 19-35).

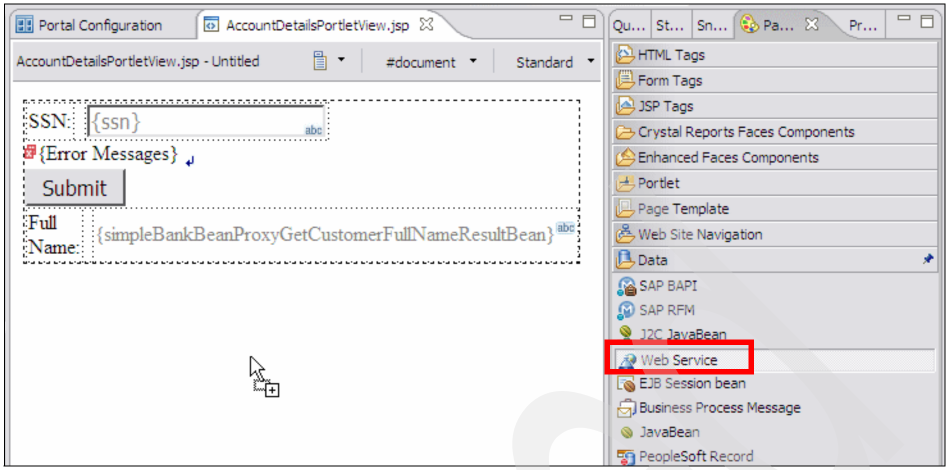
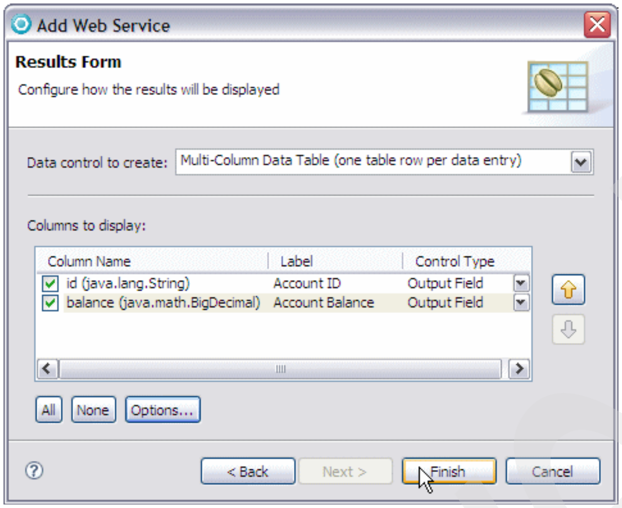


Figure 19-35 Drag and drop the Web services component for the second operation

- This time, do not initiate the Web Services Discovery Dialog, but bring up the dialog for the selection of the method of the same Web service. Click **getAccounts** and click **Next**.
- In the Input Form dialog, select the `ssn` that will be shown on the input HTML form. You can also change the label of this field and also configure its look and feel by clicking **Options**. Click **Next**.
- In the Results Form dialog, select which output fields will be shown. You can also change the label of the fields and configure the look and feel by clicking **Options**. For example, you can configure the Balance out field to be always right aligned. Click **Finish** (Figure 19-36).



Add Web Service

Results Form

Configure how the results will be displayed

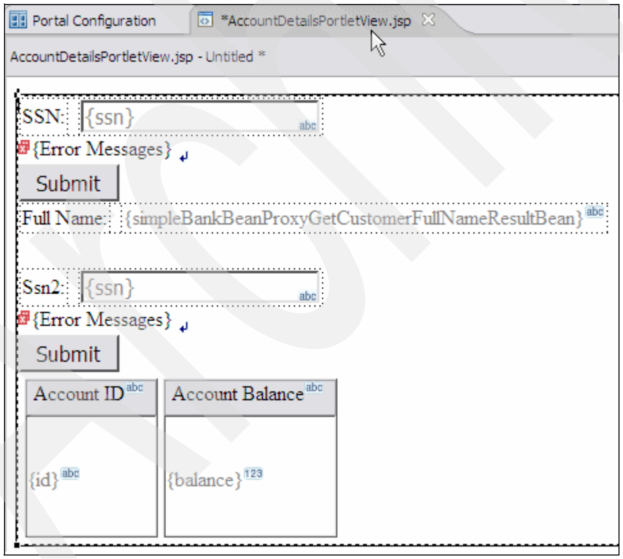
Data control to create: Multi-Column Data Table (one table row per data entry)

Columns to display:

Column Name	Label	Control Type
☑ id (java.lang.String)	Account ID	Output Field
☑ balance (java.math.BigDecimal)	Account Balance	Output Field

Figure 19-36   *Configure look and feel of the output table*

- The resulting view of the JSP now has two separate forms for the two separate Web service methods that we want to execute (Figure 19-37).



AccountDetailsPortletView.jsp - Untitled *

SSN: {ssn}

{Error Messages}

Full Name: {simpleBankBeanProxyGetCustomerFullNameResultBean}

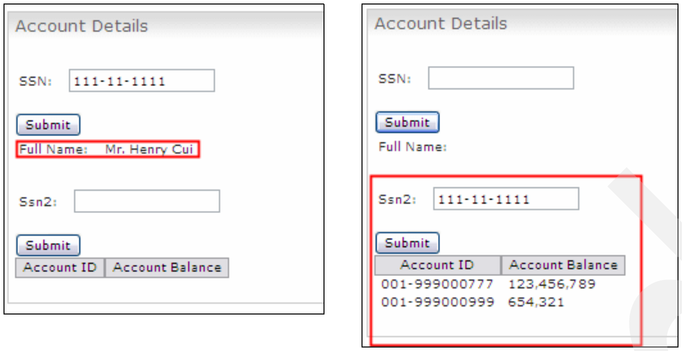
Ssn2: {ssn}

{Error Messages}

Account ID	Account Balance
{id}	{balance}

Figure 19-37   *Two separate forms for the Web service operations*

- Run and test the Portlet project (Figure 19-38).



Account Details

SSN: 111-11-1111

Submit

Full Name: Mr. Henry Cui

Ssn2:

Submit

Account ID	Account Balance

Account Details

SSN:

Submit

Full Name:

Ssn2: 111-11-1111

Submit

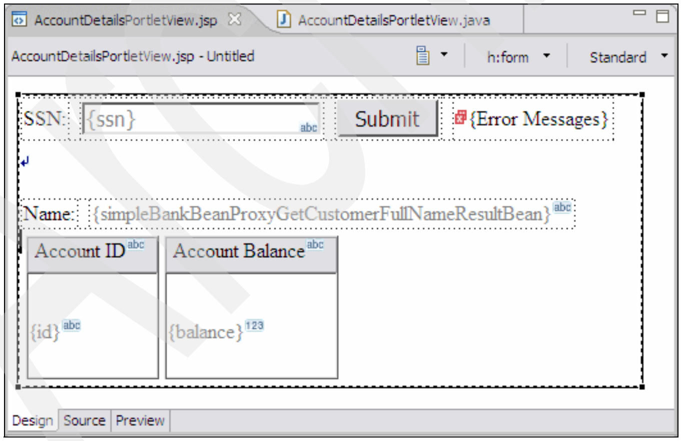
Account ID	Account Balance
001-999000777	123,456,789
001-999000999	654,321

Figure 19-38 Testing the two Web Services methods from the same portlet view

Consolidate the view

Though we have a working Web services portlet client, we still have a bit of work pending to consolidate this view. Our objective is to have one form with one Submit button and a common input field for social security number (SSN) for both Web service calls. We cannot reduce the number of Web service calls because we do have two separate operations that we have to call to get the data we want to display. Do these steps:

- Open the `AccountDetailsPortletView.jsp` and delete the second Submit button and the second input field for SSN so that the view looks as illustrated in Figure 19-39.



AccountDetailsPortletView.jsp
AccountDetailsPortletView.java
AccountDetailsPortletView.jsp - Untitled
h:form
Standard
Design
Source
Preview

SSN: {ssn}

Submit

{Error Messages}

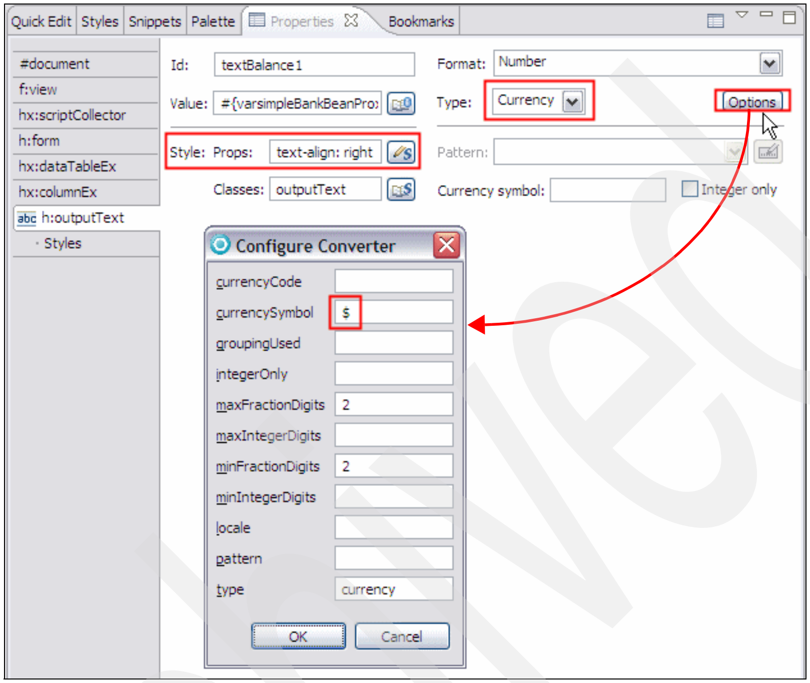
Name: {simpleBankBeanProxyGetCustomerFullNameResultBean}

Account ID	Account Balance
{id}	{balance}

Figure 19-39 Consolidated view with two Web services

- Another step (optional) that you might want to complete is to format the result that will be displayed in the account balance column so that the numbers displayed are right aligned, displayed as a currency with a $ sign in front.

To change the formatting of the balance, select the **{balance}** output component and use the Properties view to configure the format (Figure 19-40).



Id: textBalance1
Value: #{varsimpleBankBeanPro:
Style: Props: text-align: right
Classes: outputText
Format: Number
Type: Currency
Pattern:
Currency symbol:
□ Integer only

currencyCode	
currencySymbol	$
groupingUsed	
integerOnly	
maxFractionDigits	2
maxIntegerDigits	
minFractionDigits	2
minIntegerDigits	
locale	
pattern	
type	currency

Figure 19-40 Configure the look and feel of the account balance

- As you can imagine, if we execute this portlet, we are only submitting the request to execute the `getCustomerFullName` method of the Web service, and the `getAccounts` method is not called at all. This is because the Submit button is bound to a method (in the page code Java class) that only has code that communicates with the `getCustomerFullName` method. Therefore, we create a new method called `getAccountDetails` and bind it to the Submit button and from that method call the two methods that were bound to the two Submit buttons we had before. Do these steps:
  - To open the underlying page code of a Faces JSP, right-click in the Design view and select **Edit Page Code**. This opens the Java class with the same name as the JSP.
  - Browse the code, and you can find the two methods bound to the Submit buttons:

```
doSimpleBankBeanProxyGetCustomerFullNameAction();
doSimpleBankBeanProxy2GetAccountsAction();
```

   – Add a new `getAccountDetails` method at the end of the file (Example 19-2).

Example 19-2 getAccountDetails method

```
public String getAccountDetails() {
   doSimpleBankBeanProxyGetCustomerFullNameAction();
   doSimpleBankBeanProxy2GetAccountsAction();
   return null;
}
```

- The best way to determine which method is bound to a particular JSF component is to right-click on that component in the JSP page and click **Edit Page Code**. This opens the page code at the correct method.

  The method bound to a Submit button is also visible in the Properties view, action attribute (you have to click the **All Attributes** icon ▤ to the list of attributes.

- Change the method that is bound to the remaining Submit button to `getAccountDetails`. Select the **Submit** button, then switch to the **Source** tab to see the JSP source code. Change the code as shown in Example 19-3.

Example 19-3 Change the action method for the Submit button

```
<hx:commandExButton
   id="buttonDoSimpleBankBeanProxyGetCustomerFullNameAction1"
   styleClass="commandExButton" type="submit" value="Submit"
   action="#{pc_AccountDetailsPortletView.getAccountDetails}">
</hx:commandExButton>
```

- If you execute the portlet as this moment, the second Web Services method will throw back a Web services fault complaining about the null or missing social security number. This is because we still have to change the code of the second method of the page code class to use the `SSN` from the Faces bean corresponding to the first input field as illustrated in Example 19-4.

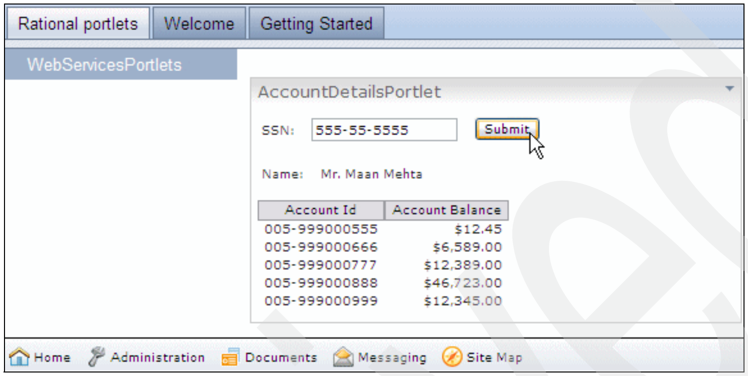
Example 19-4 get SSN from the value submitted from the first input field

```
public String doSimpleBankBeanProxy2GetAccountsAction() {
   try {
      SimpleBankBeanProxy2GetAccountsResultBean = getSimpleBankBeanProxy2()
         .getAccounts(
            getSimpleBankBeanProxyGetCustomerFullNameParamBean().getSsn());
      getSessionScope().put("SimpleBankBeanProxy2GetAccountsResultBean",
               SimpleBankBeanProxy2GetAccountsResultBean);
      } catch (Throwable e) {
         logException(e);
      }
```

```
        return null;
    }
```

Test the finished portlet with two Web service calls

Run and test the Portlet project (Figure 19-41).



SSN: 555-55-5555

Name: Mr. Maan Mehta

Account Id	Account Balance
005-999000555	$12.45
005-999000666	$6,589.00
005-999000777	$12,389.00
005-999000888	$46,723.00
005-999000999	$12,345.00

Figure 19-41 Testing the completed account details portlet

Exploring Application Developer tooling for Portal projects

By now you should have a good idea of how to develop Faces portlets using the tooling provided by the Application Developer. Though portlets bring in the most important facet for a portal application, the content or information, there is one more equally important facet of the portal application that we still want to explore. That is the how portal ties all of this aggregated content with a customized look and feel and laying out of the portlets on portal pages in a page hierarchy.

To be more specific, and in portal terminology, we are talking about the tooling provided by Application Developer to aid the portal developer in creating and management of following portal artifacts in a por*tal* (not port*let*) project:

- Portal themes and skins
- Portal page hierarchy and layout
- Portlet insertion on portal pages
- Deployment, debugging, and testing of the integrated portal application with themes and skins, portal pages, and portlets laid out in a desired page hierarchy.

We will perform the following steps:

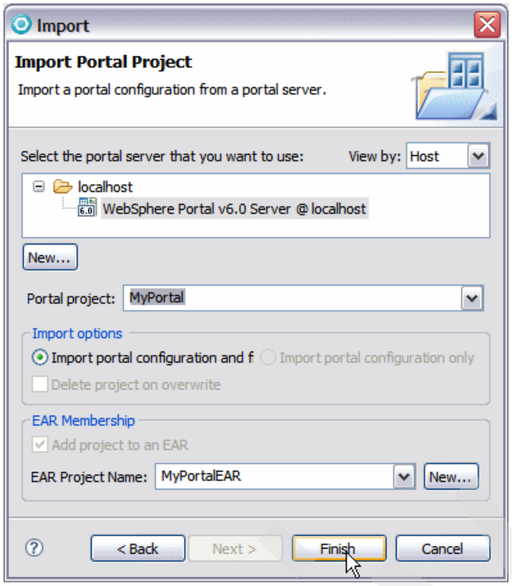
- Import the Portal project from the running Portal Server
- Browse the newly created project and its elements in detail.
- Create new pages.
- Insert portlets.
- Edit the portal theme.
- Publish the Portal project.
- Run the Portal project.

Import a portal solution

There are two ways a portal developer can create a project. One is to create a new Portal project using the Portal Project Creation wizard. Another method (more practical and recommended) is to import the portal configuration and artifacts from a running Portal Server. This section takes you through the steps involved in creating a Portal project in the Application Developer by importing the portal configuration and artifacts.

- Start the Portal Server, if it is not already started.
- From the workbench, select **File** → **Import**.
- Expand **Portal** and select **Portal**. Click **Next**.
- In the Import Portal Project dialog, enter the name of the Portal project (`MyPortal`) and the EAR project (`MyPortalEAR`). Clear **Delete project on overwrite** (Figure 19-42). Click **Finish**.
- Click **OK** in the Project Configuration Overwrite dialog.

  The wizard uses the `XMLAccess` mechanism to import the portal configuration, layout and artifacts deployed to the running Portal Server into the `MyPortal` Portal project.


Import

Import Portal Project

Import a portal configuration from a portal server.

Select the portal server that you want to use: View by: Host

- localhost
  - WebSphere Portal v6.0 Server @ localhost

New...

Portal project: MyPortal

Import options

● Import portal configuration and f ○ Import portal configuration only

□ Delete project on overwrite

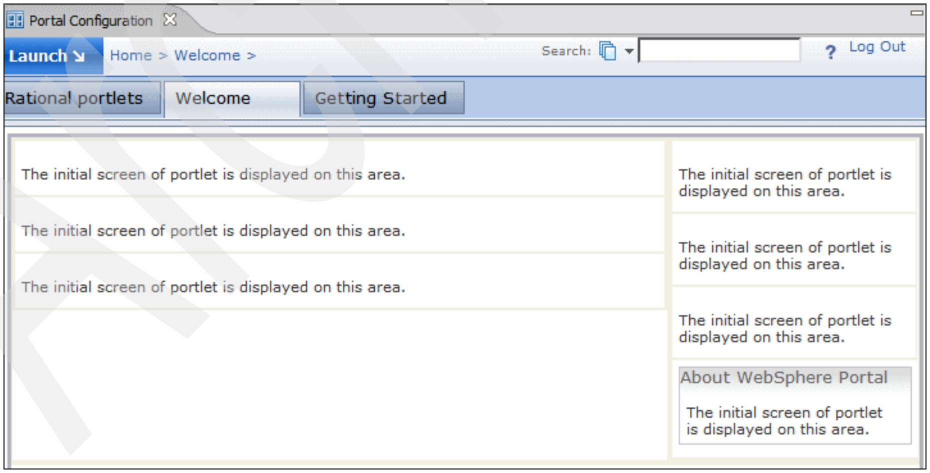
EAR Membership

☑ Add project to an EAR

EAR Project Name: MyPortalEAR New...

Figure 19-42 Import a Portal project from a Portal Server

- In the Project Explorer, expand the `MyPortal` project and double-click the **Portal Configuration** to see the WYSIWYG view of the complete portal solution with all the labels, portal pages, theme and skins, and the portlets on the portal pages (Figure 19-43).



The initial screen of portlet is displayed on this area.

The initial screen of portlet is displayed on this area.

The initial screen of portlet is displayed on this area.

The initial screen of portlet is displayed on this area.

The initial screen of portlet is displayed on this area.

The initial screen of portlet is displayed on this area.

About WebSphere Portal

The initial screen of portlet is displayed on this area.

Figure 19-43 Workbench view of portal configuration of the imported Portal project

- Take some time to explore the `MyPortal` project, its various portal artifacts, folder structure, and contents therein through several Application Developer views, especially the **Project Explorer** and the **Outline** views (Figure 19-44).

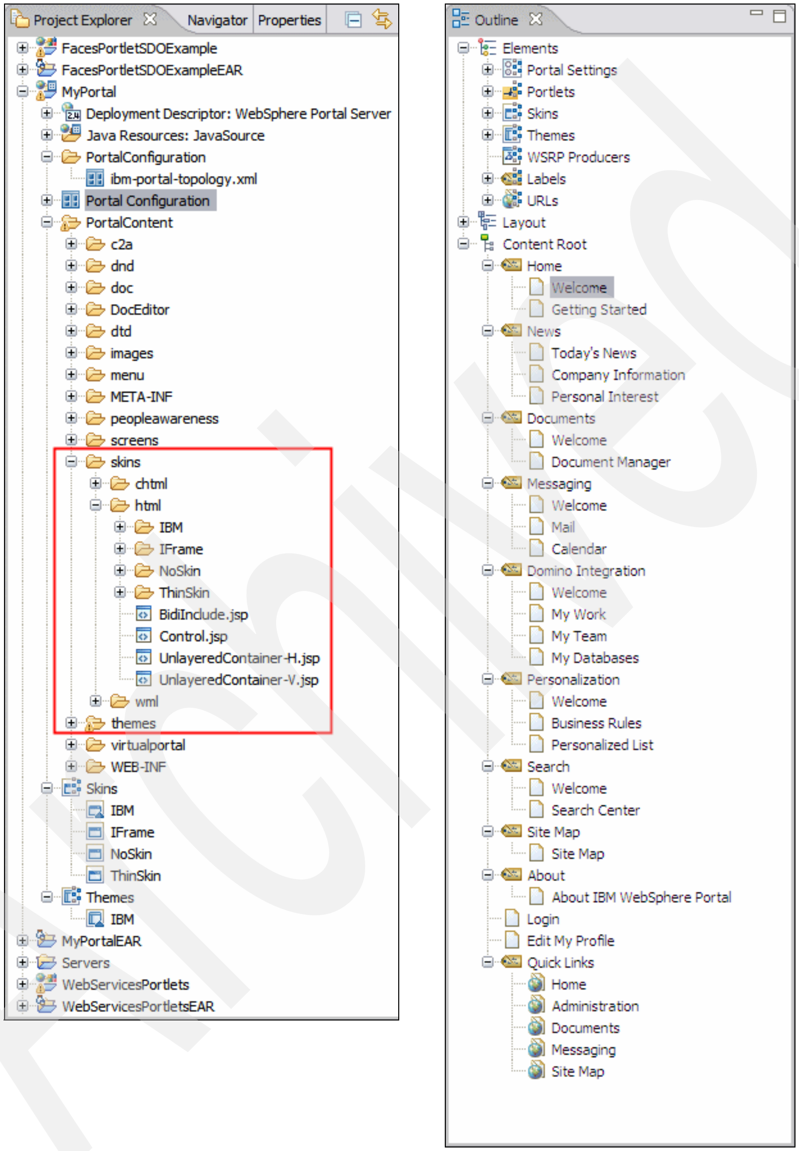


Figure 19-44 Project Explorer and Outline views of the imported Portal project

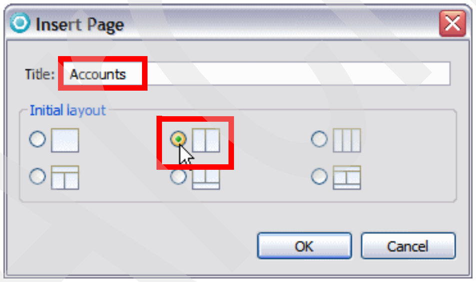
- In the Project Explorer view, you can find the portal themes and skins artifacts under the `PortalContent` folder.

- The `WEB-INF/tld` folder has all the required tag libraries for portal and portlet development.
- In the **Outline** view you can explore all the labels, portal pages and their sub-pages under the **Content Root** folder. Clicking on any page displays the WYSIWYG view of the Page and its layout (Portlets). The Portlet Element lists all the deployed portlets on all the pages of this portal application. Other important Elements to expand in the Outline view are Labels, Themes, and Skins.
- The **Palette** view provides a Portal drawer that can be used to drag and drop new portal pages and portlets into the editor.

Create pages and insert portlets

If you completed the previous exercise, you should have two Portlet projects and their corresponding EAR projects in your workspace. You are at a point where you can start creating your page hierarchy and start inserting the desired portlets on those pages. For this exercise, we create two separate pages and insert the portlets we created in our previous exercises on those new portal pages. There are multiple ways to create a portal page in the Portal project, but the best and easiest way is through the Outline view. To create a page, do these steps:

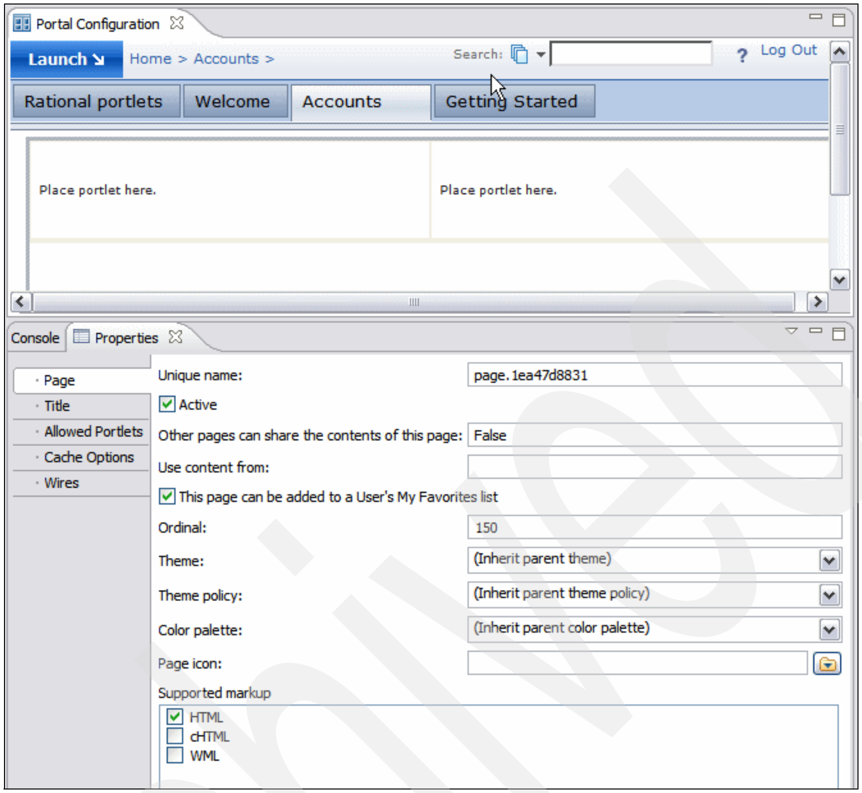
- In the Outline view, expand **Content Root**, right-click on **Home** and select **Insert Page** → **As Child**.
- In the Insert Page dialog, enter the title of the portal page (Accounts) and select the initial layout (columns and rows) for the page (Figure 19-45). Click **OK**.



Title: Accounts

Figure 19-45 Type the name of the new page and select the initial layout

- The Portal Configuration view of this new portal page is displayed in the Portal Designer Workbench (Figure 19-46).



Unique name: page.1ea47d8831

☑ Active

Other pages can share the contents of this page: False

Use content from:

☑ This page can be added to a User's My Favorites list

Ordinal: 150

Theme: (Inherit parent theme)

Theme policy: (Inherit parent theme policy)

Color palette: (Inherit parent color palette)

Page icon:

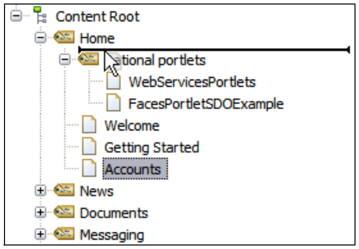
Supported markup

☑ HTML
□ cHTML
□ WML

Figure 19-46 Accounts portal page in the Portal Designer

- Take some time to explore the Properties view for this portal page and you notice that tooling provided by the Application Developer V7.0 lets the portal developer (or designer) configure various properties of the portal pages much more conveniently that it was possible before through the Portal administration console or by using `XMLAccess` scripts. Some of the important properties of the Portal page that can be configured through this console are:
  - Unique name
  - Theme and theme policy
  - Supported markup
  - Ordinality
  - Title of portal page for different locales
  - List of allowed portlets that can be inserted on this page
  - Caching options
- Let us change the order in which pages are displayed and move the new Accounts page to be the first. Through the Outline view, drag and drop the Accounts page as the first child page under the portal Home page

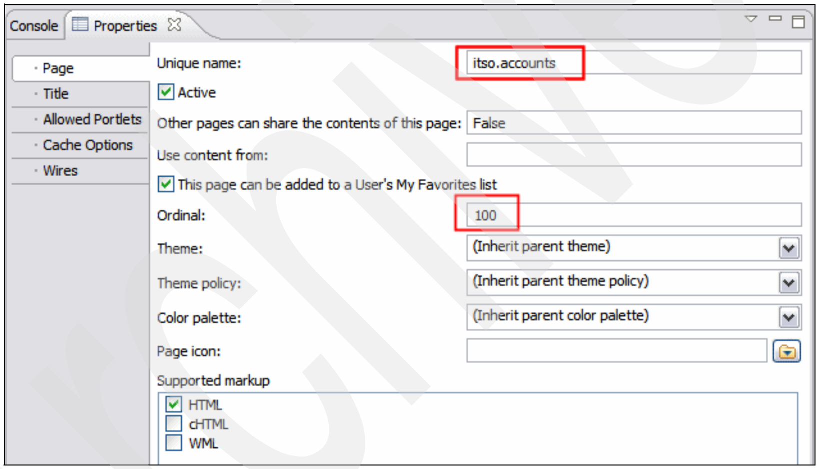
(Figure 19-47). You can also do the same by dragging and dropping the Accounts page tab in the Portal Designer.



- Content Root
  - Home
    - National portlets
      - WebServicesPortlets
      - FacesPortletSDOExample
    - Welcome
    - Getting Started
    - Accounts
  - News
  - Documents
  - Messaging

Figure 19-47 Change the order of portal pages

- ► You also notice that after the Accounts page has been moved to be the first displayed page, the Properties view of this page shows the updated Ordinality for this page (Figure 19-48).



Unique name: itso.accounts

☑ Active

Other pages can share the contents of this page: False

Use content from:

☑ This page can be added to a User's My Favorites list

Ordinal: 100

Theme: (Inherit parent theme)

Theme policy: (Inherit parent theme policy)

Color palette: (Inherit parent color palette)

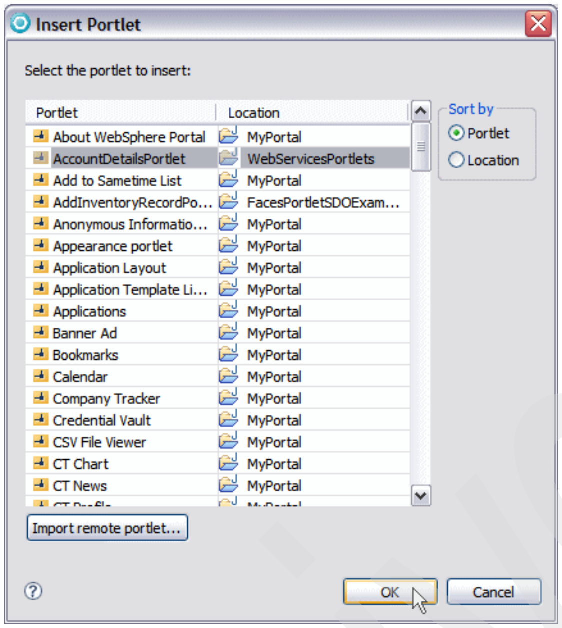
Page icon:

Supported markup

☑ HTML
☐ cHTML
☐ WML

Figure 19-48 Configure properties of a Portal page

- ► Now that we are done creating and configuring the new Accounts page, let us now insert the AccountDetailsPortlet in the left column of this page. To do this, right-click in the left column of the page layout and select **Insert Portlet** → **As Child**.
- ► In the Insert Portlet Dialog, select the **AccountDetailsPortlet** and click **OK**. (Figure 19-49).



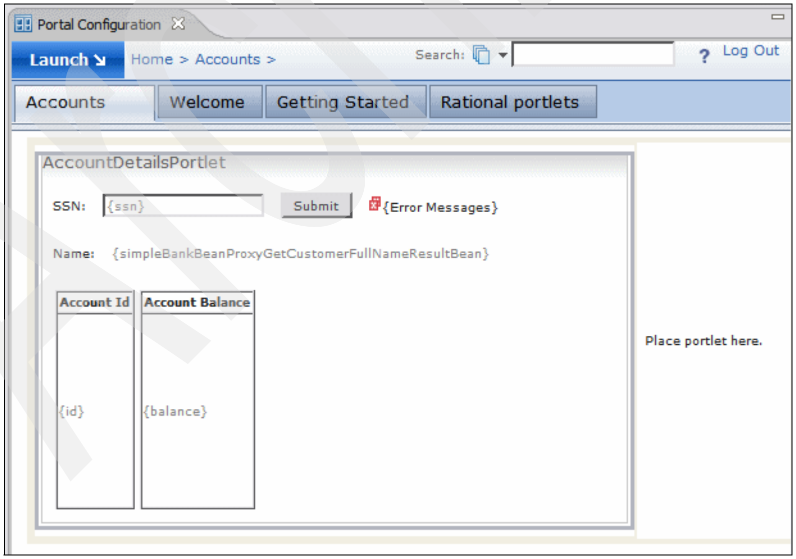
Select the portlet to insert:

Portlet	Location
About WebSphere Portal	MyPortal
AccountDetailsPortlet	WebServicesPortlets
Add to Sametime List	MyPortal
AddInventoryRecordPo...	FacesPortletSDOExam...
Anonymous Informatio...	MyPortal
Appearance portlet	MyPortal
Application Layout	MyPortal
Application Template Li...	MyPortal
Applications	MyPortal
Banner Ad	MyPortal
Bookmarks	MyPortal
Calendar	MyPortal
Company Tracker	MyPortal
Credential Vault	MyPortal
CSV File Viewer	MyPortal
CT Chart	MyPortal
CT News	MyPortal

Sort by
● Portlet
○ Location

Figure 19-49   Select which portlet to insert

- The Portal Designer displays the initial view of the AccountDetailsPortlet in the Accounts page (Figure 19-50).



SSN: {ssn} {Error Messages}

Name: {simpleBankBeanProxyGetCustomerFullNameResultBean}

Account Id	Account Balance
{id}	{balance}

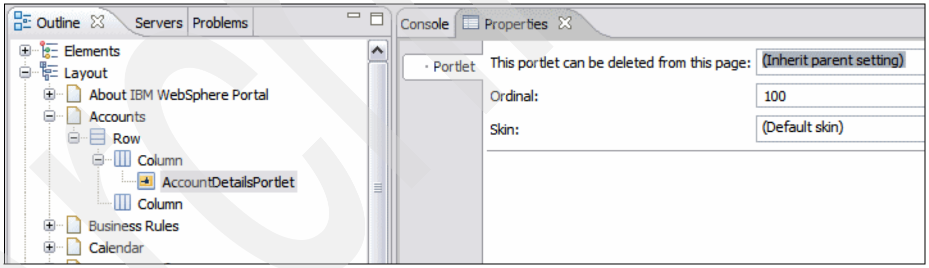
Figure 19-50   Initial view of the Accounts portal page in the Portal Designer

Initial view for the basic portlet

For the JSF and Struts based portlets, the JSP page for the initial view is known and thus displayed correctly in the Portal Designer. The default JSP page for basic portlets is automatically specified, especially if the portlet has been generated by the Portlet wizard. To change the default JSP page to another JSP page or to specify the default JSP page for your basic portlet, you have to do the following tasks:

- In the Project Explorer, find and expand the basic Portlet project. Select and expand the **Portlet Deployment Descriptor** and right-click on the portlet name and select **Properties**.
- In the Properties dialog, select the JSP page that you want to be the default, and click **OK**.

- Now you can view the parent-child relationship between the Accounts page and the AccountDetailsPortlet by expanding the Layout node in the Outline view (Figure 19-51). You can change the look and feel of the portlets by selecting a different portlet skin from the Properties view. You can modify the portlet view by selecting the portlet in the Portal Designer and selecting **Edit** → **Edit Portlet**, which opens the initial JSP page with the Page Designer.



This portlet can be deleted from this page: (Inherit parent setting)

Ordinal: 100

Skin: (Default skin)

Figure 19-51 Outline and Properties view after insertion of AccountDetailsPortlet

- In the right-hand column of the portal page, follow the same process to insert the out of the box **About WebSphere** portlet.

Publish the portal solution

There are the two ways of publishing a portal solution to a Portal Server:

- **Automated**: Deploy the Portal project directly to the Portal Server from Application Developer.

- **Manual**: Export the Portal project as a Project Deploy Set and then manually copy it to the target server and follow the set of deploy instructions generated by the export wizard.

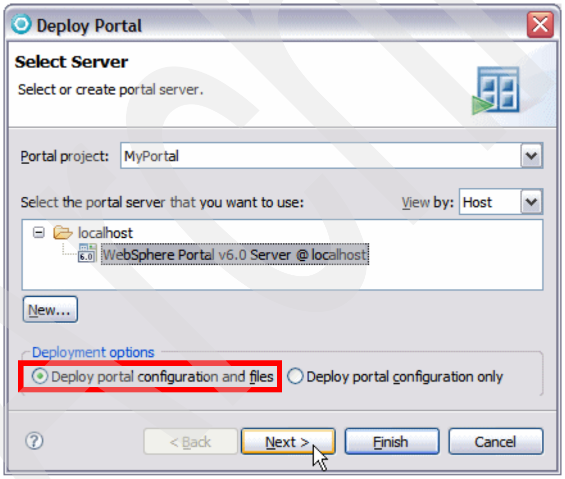
You must use the Export option when you are deploying to the staging and production server. But we also recommend using the Export option when you are testing the portal artifacts and portlets in the test environment.

The first option to deploy the portal solution using the tooling provided in Application Developer is required especially if you want to debug the themes and skins code in Application Developer or if you want to test the portlets in a completely integrated portal.

Deploy and run the Portal project

In this section we demonstrate the use of portal tooling for the deploy option. Do these steps:

- In the Project Explorer, right-click the **MyPortal** Portal project and select **Deploy Portal**.
- In the Deploy Portal dialog, select **WebSphere Portal v6.0** as the portal server you want to use for deployment and select **Deploy the portal configuration and files**. Click **Next** (Figure 19-52).



Deploy Portal

Select Server

Select or create portal server.

Portal project: MyPortal

Select the portal server that you want to use: View by: Host

localhost
WebSphere Portal v6.0 Server @ localhost

Deployment options

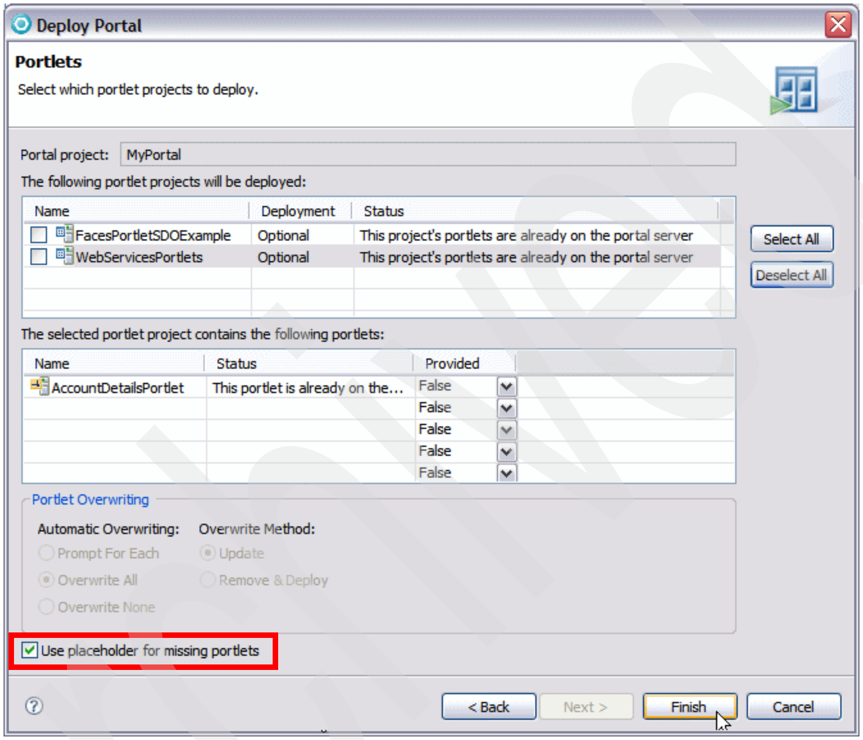
● Deploy portal configuration and files ○ Deploy portal configuration only

Figure 19-52 Deploy Portal dialog

- In the Portlets dialog, the Deploy Portal wizard presents the list of Portlet projects in the workspace and their portlets. The dialog also lists the deployment status of these Portlet projects and their portlets on the selected Portal Server.

In our case, both of our Portlet projects (`FacesPortletSDOExample` and `WebServicesPortlets`) have already been deployed to the Portal Server as part of our previous exercises. Therefore, the deployment of these portlet applications and their portlets to the selected Portal Server is optional.

Select **Use placeholder for missing portlets** and click **Finish** (Figure 19-53).



Deploy Portal

Portlets

Select which portlet projects to deploy.

Portal project: MyPortal

The following portlet projects will be deployed:

Name	Deployment	Status
□ FacesPortletSDOExample	Optional	This project's portlets are already on the portal server
□ WebServicesPortlets	Optional	This project's portlets are already on the portal server

The selected portlet project contains the following portlets:

Name	Status	Provided
AccountDetailsPortlet	This portlet is already on the...	False
		False
		False
		False
		False

Portlet Overwriting

Automatic Overwriting: ○ Prompt For Each ● Overwrite All ○ Overwrite None

Overwrite Method: ● Update ○ Remove & Deploy

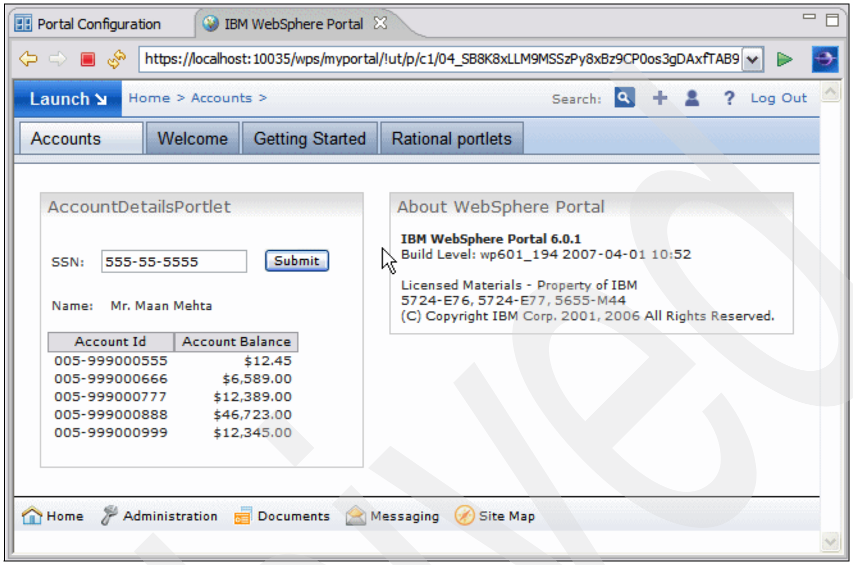
☑ Use placeholder for missing portlets

Figure 19-53 Select which portlets to deploy along with the portal

- In the Server Configuration Changed dialog, you are warned that because changes have been made to the Portal Server configuration, during deployment these changes are overwriting the present server configuration. Click **OK**.
- In the Run on Server dialog, you are asked to select the server you want to run this Portal project. Select **WebSphere Portal v6.0 Server** and click **Finish**.

  This deployment process can take some significant amount of time, depending upon your machine configuration and resources.

- After the publishing and deployment of the Portal project completes, Application Developer opens the portal application in a browser. (Figure 19-54).



AccountDetailsPortlet

SSN: 555-55-5555 Submit

Name: Mr. Maan Mehta

Account Id	Account Balance
005-999000555	$12.45
005-999000666	$6,589.00
005-999000777	$12,389.00
005-999000888	$46,723.00
005-999000999	$12,345.00

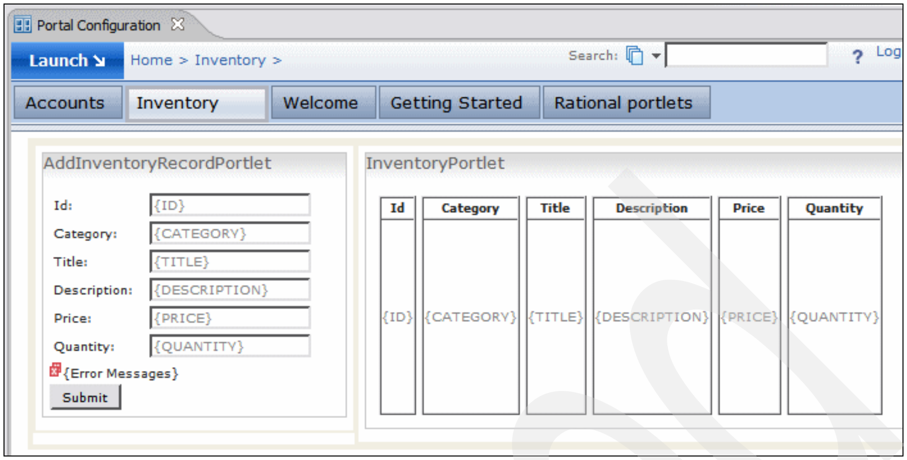
About WebSphere Portal

IBM WebSphere Portal 6.0.1
Build Level: wp601_194 2007-04-01 10:52

Licensed Materials - Property of IBM
5724-E76, 5724-E77, 5655-M44
(C) Copyright IBM Corp. 2001, 2006 All Rights Reserved.

Figure 19-54 New Accounts page with Account Details; About WebSphere portlets

- Following the same steps as above, create a new page titled Inventory and make sure it is displayed to the right of the Accounts page.
- On the Inventory page, insert the following two portlets that we created in the previous exercises:
  - Add Inventory Record portlet
  - Inventory portlet
- The Portal designer then displays the initial views of these two portlets on the new Inventory portal page (Figure 19-55).



AddInventoryRecordPortlet

Id:	{ID}
Category:	{CATEGORY}
Title:	{TITLE}
Description:	{DESCRIPTION}
Price:	{PRICE}
Quantity:	{QUANTITY}

{Error Messages}

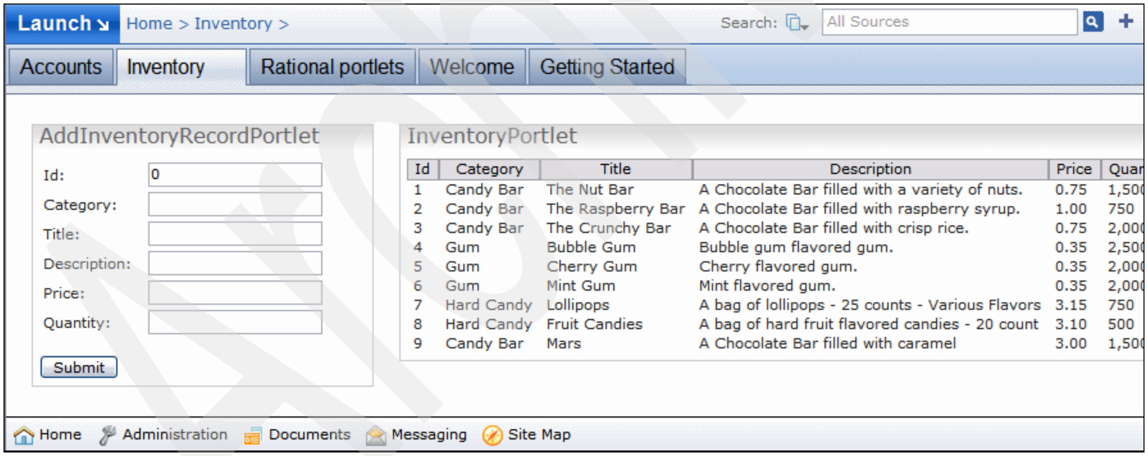
Submit

InventoryPortlet

Id	Category	Title	Description	Price	Quantity
{ID}	{CATEGORY}	{TITLE}	{DESCRIPTION}	{PRICE}	{QUANTITY}

Figure 19-55 Layout of the Inventory page in the Portal Designer

- Deploy the Portal project once again and run it on the WebSphere Portal v6.0 server. Application Developer opens the portal application in the Browser view in the workbench (Figure 19-56). You can test the two new Inventory portlets on the Inventory page.



AddInventoryRecordPortlet

Id:	0
Category:	
Title:	
Description:	
Price:	
Quantity:	

Submit

InventoryPortlet

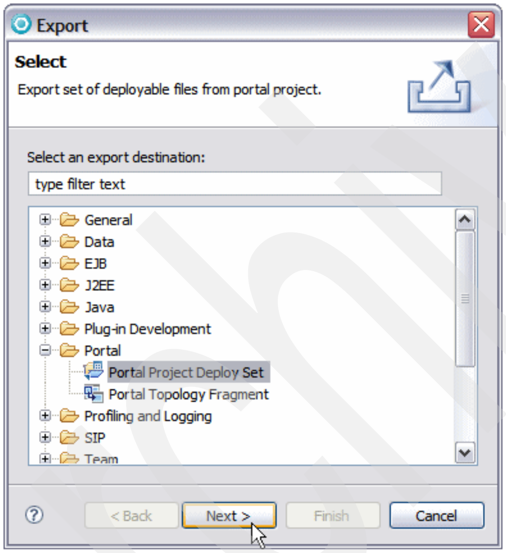
Id	Category	Title	Description	Price	Quar
1	Candy Bar	The Nut Bar	A Chocolate Bar filled with a variety of nuts.	0.75	1,500
2	Candy Bar	The Raspberry Bar	A Chocolate Bar filled with raspberry syrup.	1.00	750
3	Candy Bar	The Crunchy Bar	A Chocolate Bar filled with crisp rice.	0.75	2,000
4	Gum	Bubble Gum	Bubble gum flavored gum.	0.35	2,500
5	Gum	Cherry Gum	Cherry flavored gum.	0.35	2,000
6	Gum	Mint Gum	Mint flavored gum.	0.35	2,000
7	Hard Candy	Lollipops	A bag of lollipops - 25 counts - Various Flavors	3.15	750
8	Hard Candy	Fruit Candies	A bag of hard fruit flavored candies - 20 count	3.10	500
9	Candy Bar	Mars	A Chocolate Bar filled with caramel	3.00	1,500

Figure 19-56 New Inventory page with Add Inventory Record and Inventory portlets

Export the Portal as a Project Deploy Set

As mentioned before, besides using the portal tooling to deploy the portal application to the Portal Server directly from Application Developer, another option available is to export the Portal project as a Project Deploy Set. This produces a set of portal artifacts in a folder on the local file system that can then be deployment manually to the Portal Server. As part of the deployment set, the export wizard also prepares a document with detailed step-by-step instructions for the deployment process.

- In the Project Explorer, right-click the **MyPortal** Portal project and select **Export**.
- In the Export dialog, expand **Portal** and select **Portal Project Deploy Set** and click **Next** (Figure 19-57).



Export

Select

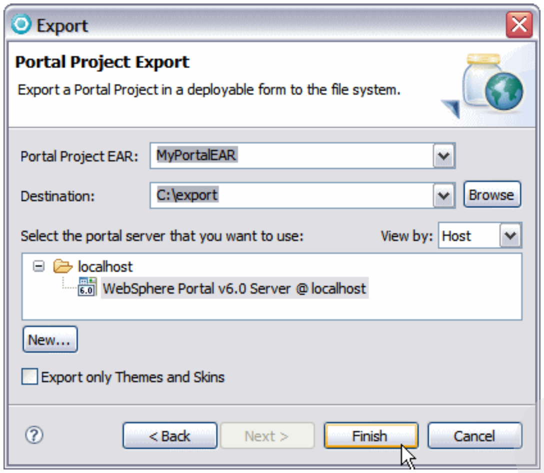
Export set of deployable files from portal project.

Select an export destination:

type filter text

Figure 19-57 Export the Portal project as a Deploy Set

- In the Portal Project Export dialog, select the Portal project EAR, and specify the folder location where this deploy set is placed on the local file system.
- Select **WebSphere Portal v6.0 server** as the Portal Server that you want to deploy to. Clear **Export only Themes and Skins**, because we want to deploy the complete portal application. Click **Finish** (Figure 19-58).


Export
Portal Project Export
Export a Portal Project in a deployable form to the file system.

Portal Project EAR: MyPortalEAR

Destination: C:\export

Select the portal server that you want to use:

View by: Host

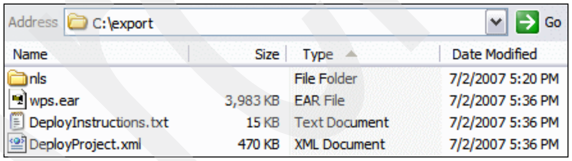
localhost
WebSphere Portal v6.0 Server @ localhost

☐ Export only Themes and Skins

Browse
New...
< Back
Next >
Finish
Cancel

Figure 19-58 Enter details for the Portal Project Deploy Set

- The export wizard exports the following items as a deploy set (Figure 19-59):
  - **nls**—A folder that contains all the resource bundles.
  - **wps.ear**—An enterprise application for deployment to the Portal Server.
  - **DeploymentInstructions.txt**—A text document with detailed step-by-step instructions to follow during the deployment process.
  - **DeployProject.xml**—An XMLAccess document that contains all the portal configuration changes made by us with all the new portal pages and the configuration of all the portlets that were inserted on them.



C:\export

Name	Size	Type	Date Modified
nls		File Folder	7/2/2007 5:20 PM
wps.ear	3,983 KB	EAR File	7/2/2007 5:36 PM
DeployInstructions.txt	15 KB	Text Document	7/2/2007 5:36 PM
DeployProject.xml	470 KB	XML Document	7/2/2007 5:36 PM

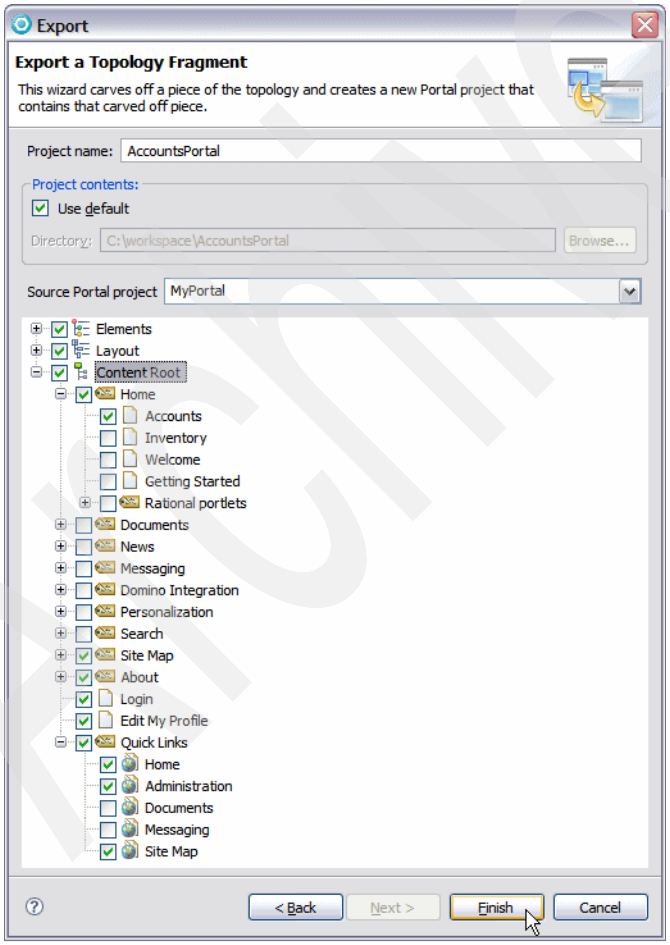
Figure 19-59 Contents of the Portal Project Deploy Set

Export the Portal project as a Portal topology fragment

Besides deploying the Portal project as a complete set, the export wizard presents another choice for exporting the Portal project: Portal topology fragment.

Using this choice, you have control over what portal artifacts you want to include or exclude in a new Portal project which can be regarded as a fragment of the original Portal project. This option is especially handy when you working in a team environment and can divide the non-overlapping portal topology fragments between the resources. Then, when they are done, their fragmented portal projects can be imported back into the original Portal project. Do these steps:

- In the Project Explorer, right-click the **MyPortal** Portal project and select **Export**.
- In the Export dialog, expand **Portal** and select **Portal Topology Fragment** and click **Next** (see Figure 19-57 on page 955).
- In the Export a Topology Fragment dialog, you can select what portal pages, portlets, themes, and skins you want to include in the new fragmented project.
  - Enter `AccountsPortal` as the project name, select `MyPortal` as the Source Portal project, and select which portal elements you want to include. Click **Finish** (Figure 19-60).



Export

Export a Topology Fragment

This wizard carves off a piece of the topology and creates a new Portal project that contains that carved off piece.

Project name: AccountsPortal

Project contents:

☑ Use default

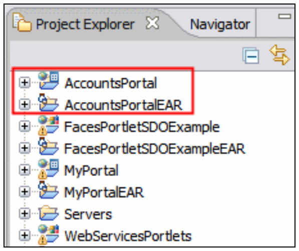
Directory: C:\workspace\AccountsPortal

Source Portal project: MyPortal

- ☑ Elements
- ☑ Layout
- ☑ Content Root
  - ☑ Home
    - ☑ Accounts
    - ☐ Inventory
    - ☐ Welcome
    - ☐ Getting Started
    - ☐ Rational portlets
  - ☐ Documents
  - ☐ News
  - ☐ Messaging
  - ☐ Domino Integration
  - ☐ Personalization
  - ☐ Search
  - ☑ Site Map
  - ☑ About
  - ☑ Login
  - ☑ Edit My Profile
  - ☑ Quick Links
    - ☑ Home
    - ☑ Administration
    - ☐ Documents
    - ☐ Messaging
    - ☑ Site Map

Figure 19-60 Selecting what to export as a fragment

- The export wizard creates a new Portal project named `AccountsPortal` and an EAR project named `AccountsPortalEAR` (Figure 19-61).



AccountsPortal
AccountsPortalEAR
FacesPortletSDOExample
FacesPortletSDOExampleEAR
MyPortal
MyPortalEAR
Servers
WebServicesPortlets

Figure 19-61 The new fragmented Portal projects

Edit themes and skins

One of the key features of Portal framework is the ability to create customized themes and skins. As mentioned before, the Portal project includes the theme and skin artifacts and also provides tooling to create or modify themes and skins.

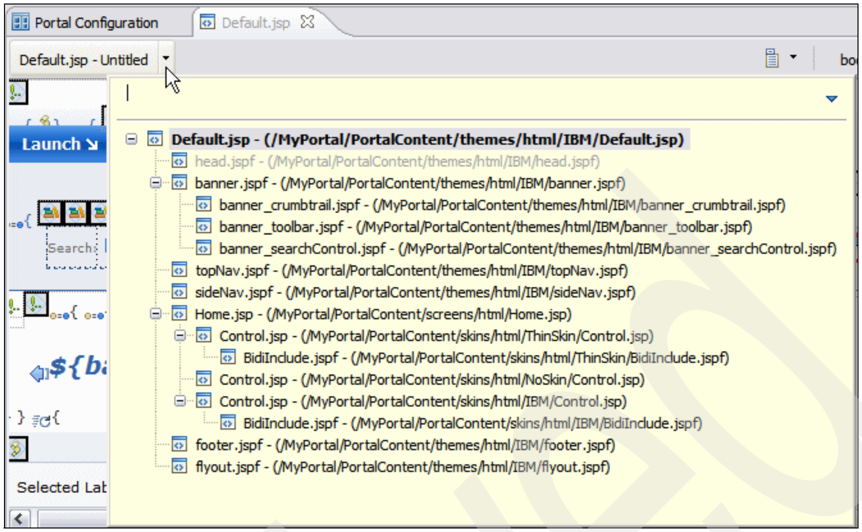
To edit the theme of a Portal page, do these steps:

- Right-click on the portal page in the Portal Designer and select **Edit Theme**.
- This opens the **Default.jsp** file of the theme of that portal page in the Page Designer.

To edit the skin of a Portal page, select that portlet either in the Portal Designer or in the Outline view, right-click, and select **Edit Skin** to open the **Control.jsp** file of the skin for that portlet.

The `Default.jsp` is the top level JSP page for a portal theme and includes several JSP fragments that make up different parts of the Portal view, such as the header section of the JSP page, top navigation, side navigation, banner (toolbar, search control, and crumb trail), footer and flyouts.

The Page designer view provides an easy mechanism in the form of an intuitive *flyout* that can be opened when you click the flyout icon of the `Default.jsp`. This flyout (Figure 19-62) displays links to all the JSP fragments that are included in the `Default.jsp` in an intuitive hierarchy for quick and easy navigation to the JSP fragments and back. The flyout also displays the links to the `Control.jsp` for all the skins of the all the portlets on that portal page.



```
Default.jsp - (/MyPortal/PortalContent/themes/html/IBM/Default.jsp)
    head.jspf - (/MyPortal/PortalContent/themes/html/IBM/head.jspf)
    banner.jspf - (/MyPortal/PortalContent/themes/html/IBM/banner.jspf)
        banner_crumbtrail.jspf - (/MyPortal/PortalContent/themes/html/IBM/banner_crumbtrail.jspf)
        banner_toolbar.jspf - (/MyPortal/PortalContent/themes/html/IBM/banner_toolbar.jspf)
        banner_searchControl.jspf - (/MyPortal/PortalContent/themes/html/IBM/banner_searchControl.jspf)
    topNav.jspf - (/MyPortal/PortalContent/themes/html/IBM/topNav.jspf)
    sideNav.jspf - (/MyPortal/PortalContent/themes/html/IBM/sideNav.jspf)
    Home.jsp - (/MyPortal/PortalContent/screens/html/Home.jsp)
        Control.jsp - (/MyPortal/PortalContent/skins/html/ThinSkin/Control.jsp)
            BidiInclude.jspf - (/MyPortal/PortalContent/skins/html/ThinSkin/BidiInclude.jspf)
        Control.jsp - (/MyPortal/PortalContent/skins/html/NoSkin/Control.jsp)
        Control.jsp - (/MyPortal/PortalContent/skins/html/IBM/Control.jsp)
            BidiInclude.jspf - (/MyPortal/PortalContent/skins/html/IBM/BidiInclude.jspf)
    footer.jspf - (/MyPortal/PortalContent/themes/html/IBM/footer.jspf)
    flyout.jspf - (/MyPortal/PortalContent/themes/html/IBM/flyout.jspf)
```

Figure 19-62 Default.jsp Flyout: Theme and Skin JSP and JSP fragments

Take some time to explore the code in all the JSP fragments and the purpose they serve. You can try the following simple exercises:

- Remove the footer from the portal.
- Remove the search form at the top of a portal page.
- Remove the crumb trail.
- Change the banner image.
- Add a link to the portal flyout menu that takes us to the Inventory page.
- Add a greeting to welcome the user at the top of the page with their first name.
- Change the background color for the whole portal page and also change the background color of the portlets to a different color.

More information

For more information on portal technology, refer to these resources:

- WebSphere Portal version 6.0 refresh pack 1(WP v6.0.1):

  http://www-1.ibm.com/support/docview.wss?rs=688&uid=swg24015257

- WebSphere Portal 6.0.1 refresh pack - upgrade instructions:

  http://publib.boulder.ibm.com/infocenter/wpdoc/v6r0/index.jsp?topic=/com.ibm.wp.ent.doc/wpf/pui_intro601.html

- WebSphere Portal Update Installer:

  http://www-1.ibm.com/support/docview.wss?rs=688&uid=swg24006942

- WebSphere Application Server V6.0.2 Fix Pack 17 (WAS v6.0.2.17) for Windows platforms:

  http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg24014309

- Update Installer for WebSphere Application Server V6.0 releases:

  http://www-1.ibm.com/support/docview.wss?rs=180&uid=swg24008401

- Troubleshooting - Rational Application Developer 7.x cannot sense the proper state of Portal Server 6.x:

  http://www-1.ibm.com/support/docview.wss?rs=2042&context=SSRTLW&context=SSJM4G&context=SSCGQ7C&dc=DB520&uid=swg21258582&loc=en_US&cs=utf-8&lang=en

- Installing and configuring WebSphere Portal V6.0 Servers for development with Rational Application Developer V7.0 and Rational Software Architect V7.0:

  http://www.ibm.com/developerworks/rational/library/07/0327_riordan/index.html

Part 3

Test and debug applications

In this part of the book, we describe the tooling and technologies provided by Application Developer for testing and debugging.

20

Servers and server configuration

Application Developer provides support for testing, debugging, profiling, and deploying Enterprise applications to local and remote test environments.

To run an enterprise application or Web application in Application Developer, the application must be published (deployed) to the server. This is achieved by installing the EAR project for the application into an application server. The server can then be started and the application can be tested in a Web browser, or by using the Universal Test Client (UTC) for EJBs and Web services.

This chapter describes the features and concepts of server configuration, as well as demonstrating how to configure a server to test applications.

The chapter is organized into the following sections:

- Introduction to server configurations
- Understanding WebSphere Application Server V6.1 profiles
- WebSphere Application Server V6.1 installation
- Using WebSphere Application Server V6.1 profiles
- Adding and removing applications to and from a server
- Configuring application and server resources
- Configuring security
- Developing automation scripts

Introduction to server configurations

Application Developer includes integrated test environments for WebSphere Application Server V5.1/V6.0/V6.1 and WebSphere Portal V5.1/V6.0, as well as support for many third-party servers obtained separately. The server configuration is the same for WebSphere Application Server V6.x (base), Express, and Network Deployment Editions. One of the many great features of the server tooling is the ability to simultaneously run multiple server configurations and test environments on the same development node where Application Developer is installed.

When using Application Developer, it is very common for a developer to have multiple test environments or server configurations, which are made up of workspaces, projects, and preferences, and supporting test environments (local or remote).

Some of the key features of test environment configuration include:

- Multiple workspaces with different projects, preferences, and other configuration settings defined
- Multiple Application Developer test environment servers configured
- When using WebSphere Application Server V6.x Test Environments, multiple profiles, each potentially representing a different server configuration

For example, a developer might want to have a separate server configuration for WebSphere Application Server V6.x with a unique set of projects and preferences in a workspace and server configuration pointing to a newly created and customized WebSphere Application Server V6.x profile. On the same system, the developer can create a separate portal server configuration with unique portal workspace projects and preferences, as well as a WebSphere Portal V6.0 Test Environment. This chapter describes how to create, configure, and run multiple WebSphere Application Server V6.1 instances on the same development system.

Compatible application servers for Application Developer

The most commonly used application server with Application Developer is WebSphere Application Server. WebSphere Application Server is tightly integrated with Application Developer, which offers tooling to test, run, and debug applications from the workbench to the server, such as using the run-on-server functionality. You can specify server-specific configurations such as extensions and bindings to a WebSphere Application Server from the Workbench.

You can launch tooling from WebSphere Application Server within the Workbench, such as the WebSphere Administrative Console and the Profile Management Tool. In addition, you can develop, run, and debug administrative scripts against a WebSphere Application Server from within the Workbench.

In Application Developer, the integration with the WebSphere Application Server V6.x for deployment, testing, and administration is the same for WebSphere Application Server V6.x Test Environment, separate install, and the Network Deployment edition. In previous versions of WebSphere Studio, the configuration of the test environment v5.x was different than a separately installed WebSphere Application Server 5.x.

The following application servers are compatible with Application Developer:

- IBM WebSphere Application Server Versions 5.1, 6.0, or 6.1
- IBM WebSphere Application Server Express Version 5.1
- IBM WebSphere Portal Version 5.1 or 6.0
- J2EE Publishing Server (publish EAR to a server)
- Static Web Publishing Server (HTTP server for static Web projects)

Application Developer server tools are based on the Eclipse Web Tools Platform (WTP) project. WTP provides a facility for publishing an Enterprise Application project and all of its modules to a runtime environment for testing purposes.

The server adapters on the following list are included in the Web Tools Platform 1.5.x based on Eclipse technology. Server adapters are tools installed into the Workbench that support a particular server. Here is the list of server adapters:

- IBM WebSphere Application Server Community Edition Version 1.1
- Apache Tomcat Versions 3.2, 4.0, 4.1, 5.0, or 5.5
- BEA WebLogic Server Versions 8.1, 9.0, or 9.2
- JBoss Versions 3.2.3 or 4.0 from a division of Red Hat
- ObjectWeb Java Open Application Server (JOnAS) Version 4
- Oracle Containers for J2EE (OC4J) Standalone Server Version 10.1.3
- Apache Geronimo Version 1.1
- Pramati Server Version 4.1.x

Local and remote test environments

When configuring a test environment, the server can be either a local integrated server or a remote server. Once the server itself is installed and configured, the server definition within Rational Application Developer is very similar for local and remote servers.

In both the local and remote configurations, remote method invocation (RMI) and SOAP connectors are used to make Java Management Extensions (JMX) connections with the WebSphere v6.x server from Application Developer. The RMI (ORB bootstrap) port is designed to improve performance and communication with the server. The SOAP connector port is designed to be more firewall compatible and uses HTTP transport as the base protocol.

Understanding WebSphere Application Server V6.1 profiles

The concept of server profiles was introduced starting with WebSphere Application Server V6.0. The WebSphere Application Server installation process simply lays down a set of core product files required for the runtime processes. After installation, you have to create one or more profiles that define the runtime settings for a functional system. The core product files are shared among the runtime components defined by these profiles.

With WebSphere Application Server Base and Express Editions, you can only have standalone application servers, as shown in Figure 20-1. Each application server is defined within a single cell and node. The administration console is hosted within the application server and can only connect to that application server. No central management of multiple application servers are possible. An application server profile defines this environment.

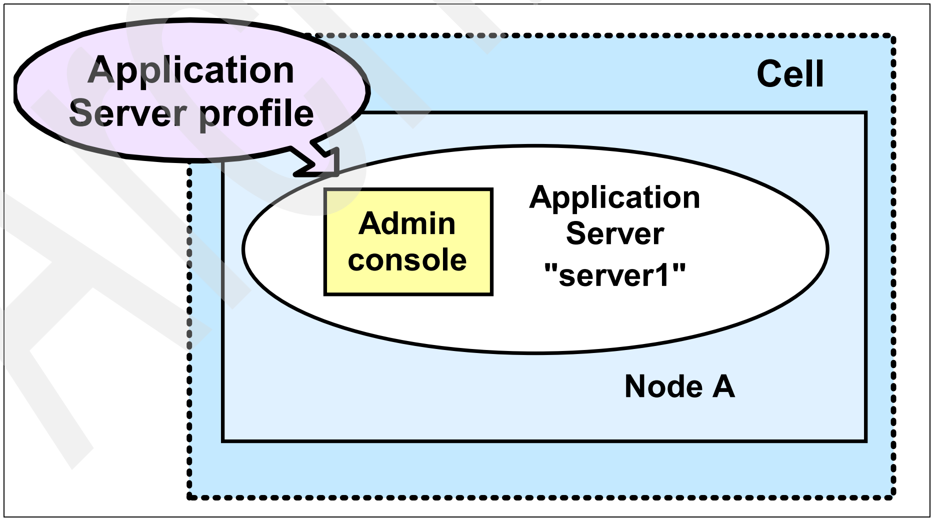


Figure 20-1 System management topology: Standalone server (Base and Express)

You can also create standalone application servers with the Network Deployment package, though you would most likely do so with the intent of federating that server into a cell for central management at some point.

With the Network Deployment package, you have the option of defining multiple application servers with central management capabilities. For more information on profiles for the IBM WebSphere Application Server V6.1 Network Deployment Edition, refer to the *WebSphere Application Server V6.1: Systems Management and Configuration*, SG247304.

Types of profiles

There are three types of profiles when defining the runtime of an application server:

- Application server profile

> **Note:** The application server profile is used by the Rational Application Developer WebSphere Application Server V6.1 Test Environment.

- Deployment manager profile
- Custom profile

Application server profile

The application server profile defines a single standalone application server. Using a profile will give you an application server that can run standalone (unmanaged) with the following characteristics:

- The profile consists of one cell, one node, and one server. The cell and node are not relevant in terms of administration, but you will see them when you administer the server through the administrative console (scopes).
- The name of the application server is `server1`.
- The WebSphere sample applications are automatically installed on the server.
- The server has a dedicated administrative console.

The primary use for this type of profile could be any of these possibilities:

- To build a server in a Base or Express installation (including a test environment within Rational Application Developer).
- To build a standalone server in a Network Deployment installation that is not managed by the deployment manager (for example, to build a test machine).

- To build a server in a distributed server environment to be federated and managed by the deployment manager. If you are new to WebSphere Application Server and want a quick way of getting an application server complete with samples, this is a good option. When you federate this node, the default cell becomes obsolete and the node is added to the deployment manager cell. The server name remains as `server1` and the administrative console is removed from the application server.

Deployment manager profile

The deployment manager profile defines a deployment manager in a Network Deployment installation. Although you could conceivably have the Network Deployment package and run only standalone servers, this would bypass the primary advantages of Network Deployment, which are workload management, failover, and central administration.

In a Network Deployment environment, you should create one deployment manager profile. This will give you:

- A cell for the administrative domain
- A node for the deployment manager
- A deployment manager with an administrative console.
- No application servers

Once you have the deployment manager, you can:

- Federate nodes built either from existing application server profiles or custom profiles.
- Create new application servers and clusters on the nodes from the administrative console.

Custom profile

A custom profile is an empty node, intended for federation to a deployment manager. This type of profile is used when you are building a distributed server environment. You would use this as follows:

- Create a deployment manager profile.
- Create one custom profile on each node on which you will run application servers.
- Federate each custom profile, either during the custom profile creation process or later using the **`addNode`** command, to the deployment manager.
- Create new application servers and clusters on the nodes from the administrative console.

WebSphere Application Server V6.1 installation

The IBM WebSphere Application Server V6.1 Integrated Test Environment is an installation option from the main Rational Application Developer Installer.

For details on how to install the WebSphere Application Server V6.1 Test Environment, refer to “Installing IBM Rational Application Developer” on page 1283 (Figure A-13 on page 1290).

Prior to the IBM WebSphere Application Server V6.1 Test Environment installation, the runtimes directory looks as follows:

```
<rad_home>\runtimes\base_v61_stub
```

The stub folder contains minimal sets of compile-time libraries that allow you to build applications for a server when it is not installed locally.

After the WebSphere Application Server V6.1 Test Environment is installed you see this directory:

```
<rad_home>\runtimes\base_v61
```

The `base_v61` folder is the Application Server installation directory.

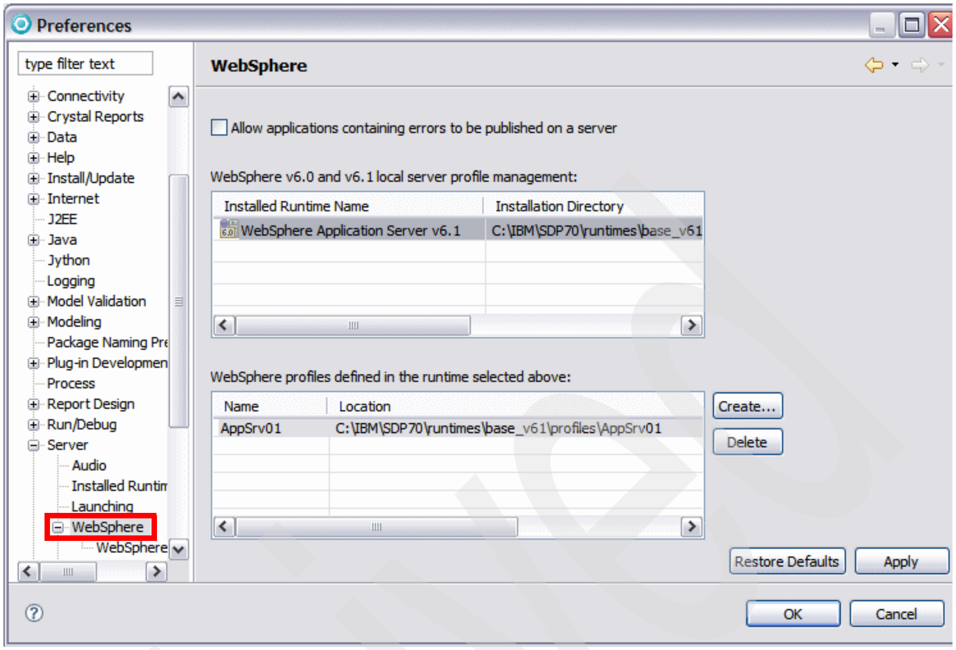
Using WebSphere Application Server V6.1 profiles

In “Understanding WebSphere Application Server V6.1 profiles” on page 966, we reviewed the concepts for WebSphere V6.1 profiles. The Profile Management tool is a WebSphere Application Server tool that creates the profile for each runtime environment. It is also a graphical user interface to the WebSphere Application Server command-line tool, `wasprofile`. You can use the development tools in this product to start WebSphere Application Server's Profile Management tool.

Creating a new profile using the WebSphere Profile wizard

To create a new WebSphere Application Server V6.1 profile using the WebSphere Profile Management Tool, do these steps:

- In the workbench, select **Window** → **Preferences**.
- In the Preferences window, expand **Server** → **WebSphere**.
- Under the **WebSphere v6.0 and v6.1 local server profile management list**, select **WebSphere Application Server v6.1** (Figure 20-2).



WebSphere

☐ Allow applications containing errors to be published on a server

WebSphere v6.0 and v6.1 local server profile management:

Installed Runtime Name	Installation Directory
WebSphere Application Server v6.1	C:\IBM\SDP70\runtimes\base_v61

WebSphere profiles defined in the runtime selected above:

Name	Location
AppSrv01	C:\IBM\SDP70\runtimes\base_v61\profiles\AppSrv01

Figure 20-2 Server Preferences page

- Click **Create** beside the **WebSphere profiles defined in the runtime selected above** list, which lists all the profiles defined for the currently selected runtime environment in the previous step.
- In the Welcome to the Profile Management Tool page, click **Next**.
- In the Environment selection page, click **Next**.
- In the Profile Creation Options page, select **Typical profile creation**, click **Next**.
- In the Administrative Security page, clear **Enable administrative security**, click **Next**.
- In the Profile Creation Summary page, review the profile settings. You can see the ports are incremented by 1 for the new profile you created so that the new profile does not conflict with an existing profile. Click **Create**.
- In the Profile Creation Complete page, clear **Launch the First steps console**. Click **Finish**.
- Back in the Preferences page, you can see the new profile is listed under the **WebSphere profiles defined in the runtime selected above** list.

Verifying the new WebSphere profile

After creating the WebSphere profile, you can verify that it was created properly and familiarize yourself with how to use it. Do these steps:

- View the directory structure and find the new profile:

  ```
  <rad_home>/runtimes/base_v61/profiles/<profile_name>
  ```

  Where *<profile_name>* is the name of the WebSphere Profile.

  This is where you will find, among other things, the `config` directory containing the application server configuration files, the `bin` directory (for entering commands), and the `logs` directory where information is recorded.

  > **Note:** For simplicity, we will refer the entire path for the profile as *<profile_home>*.

- Start the server. Open the command prompt, and issue the following commands:

  ```
  cd <profile_home>\bin
  startServer server1
  ```

- Check the server log files. Once the server is started, you can see the following message in the command prompt:

  ```
  ADMU3000I: Server server1 open for e-business; process id is 4052
  ```

  The server log files are in the `<profile_home>/logs/server1` folder. Open the following file to see the logging message:

  ```
  <profile_home>/logs/server1/SystemOut.log
  ```

- Open the WebSphere Administrative Console by accessing its URL from a Web browser:

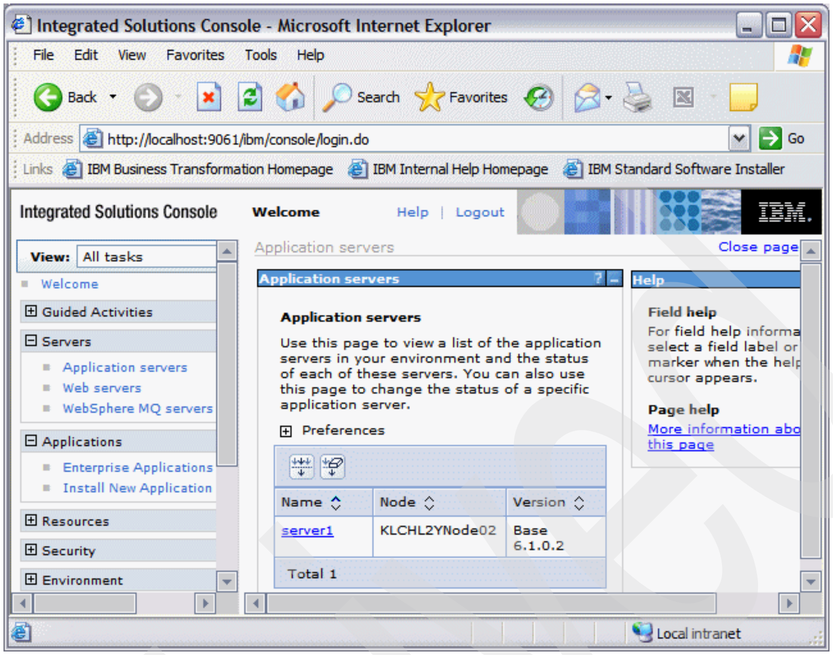
  ```
  http://<appserver_host>:<admin_console_port>/ibm/console
  http://localhost:9061/ibm/console/
  ```

  The administrative console port was listed in the Profile Creation Summary page during the profile creation.

- Click **Log in**. Because security is not active at this time, you do not have to enter a user name. If you choose to enter a name, it can be any name. If you enter a name it will be used to track changes you made to the configuration.

- Display the configuration from the console. You should be able to see the following items from the administrative console:

  – Application servers: Select **Servers** → **Application servers**. You should see `server1` (Figure 20-3). To see the configuration of this server, click the name in the list.


Integrated Solutions Console - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:9061/ibm/console/login.do Go
Links IBM Business Transformation Homepage IBM Internal Help Homepage IBM Standard Software Installer
Integrated Solutions Console Welcome Help | Logout IBM.
View: All tasks
Welcome
Guided Activities
Servers
Application servers
Web servers
WebSphere MQ servers
Applications
Enterprise Applications
Install New Application
Resources
Security
Environment
Application servers Close page
Application servers
Help
Preferences
Local intranet

Application servers

Use this page to view a list of the application servers in your environment and the status of each of these servers. You can also use this page to change the status of a specific application server.

Name	Node	Version
server1	KLCHL2YNode02	Base 6.1.0.2
Total 1		

Field help

For field help informa select a field label or marker when the help cursor appears.

Page help

More information abo this page

Figure 20-3 Application server defined by the application server profile

- Enterprise applications: Select **Applications** → **Enterprise Applications**. You should see a list of applications installed on server1.
- Click **Logout** and close the browser.

- Stop the application server using the **stopServer** command:

  ```
  stopServer server1
  ```

  You should see the following message once the server is stopped:

  ```
  ADMU4000I: Server server1 stop completed.
  ```

Deleting a WebSphere profile

We do not delete the profile we just created. For future reference, these are the instructions on how to delete the profile—but do **not** perform this now:

- From the menu bar of the workbench, select **Window** → **Preferences** → **Server** → **WebSphere**.
- Under the WebSphere v6.0 and v6.1 local server profile management list, select the installed runtime environment containing the profile which you want to delete.

- Under the WebSphere profiles defined in the runtime selected list above, select the profile which you want to delete and click **Delete**. The registry and configuration files associated to the deleted profile are removed from the file system; however, any log files remain on the file system.

Defining a new server in Application Developer

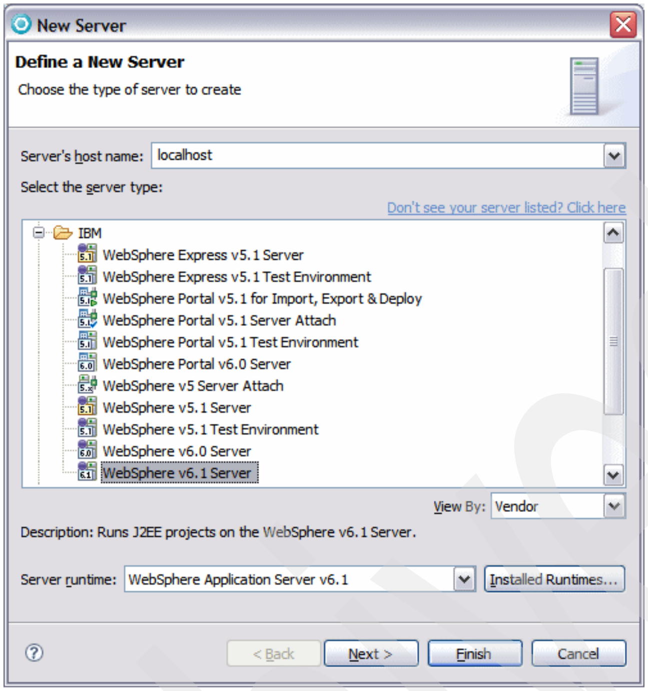
Once you have defined the WebSphere profile, you can create a server in Application Developer. The server points to the server defined within the WebSphere profile you configured. There are a few considerations:

- The profile *AppSrv01* is created when the WebSphere Application Server V6.1 Integrated Test Environment feature is selected during Rational Application Developer installation. A Rational Application Developer WebSphere Application Server V6.1 Test Environment is configured to use the `AppSrv01` profile.
- The Application Developer server configuration is essentially a pointer to the WebSphere profile.

Create a server

To create a server in Application Developer, do these steps:

- Select the **Servers** view, in the J2EE or Web perspective.
- Right-click in the Servers view and select **New** → **Server**.
- In the Define a New Server dialog (Figure 20-4), select the following items:
  - Host name: `localhost` (default)
  - Select the server type: Select **WebSphere v6.1 Server**.
  - Click **Next**.



New Server

Define a New Server

Choose the type of server to create

Server's host name: localhost

Select the server type:

Don't see your server listed? Click here

- IBM
  - WebSphere Express v5.1 Server
  - WebSphere Express v5.1 Test Environment
  - WebSphere Portal v5.1 for Import, Export & Deploy
  - WebSphere Portal v5.1 Server Attach
  - WebSphere Portal v5.1 Test Environment
  - WebSphere Portal v6.0 Server
  - WebSphere v5 Server Attach
  - WebSphere v5.1 Server
  - WebSphere v5.1 Test Environment
  - WebSphere v6.0 Server
  - WebSphere v6.1 Server

View By: Vendor

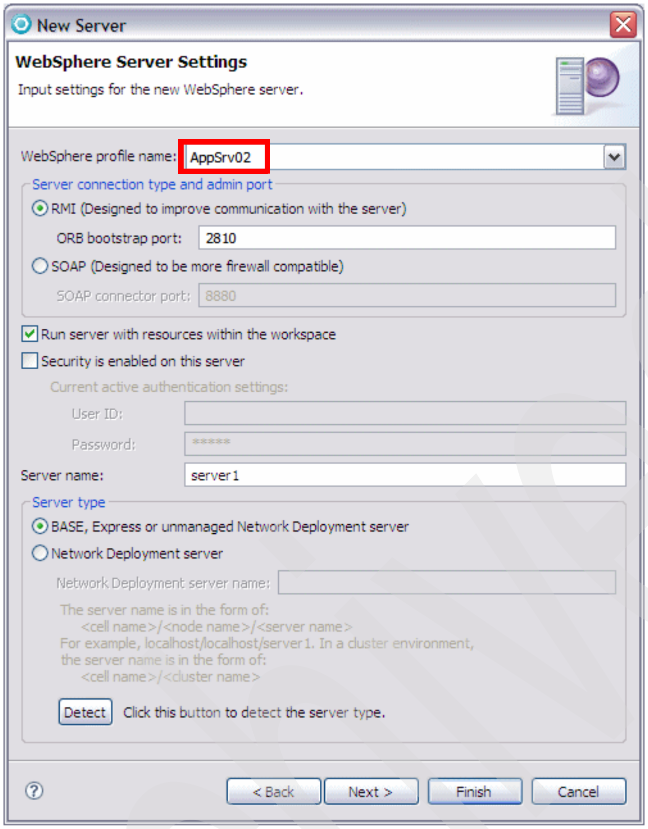
Description: Runs J2EE projects on the WebSphere v6.1 Server.

Server runtime: WebSphere Application Server v6.1

Figure 20-4 Define a New Server: Type and runtime

- In the WebSphere Server Settings dialog (Figure 20-5), do these steps:
  - WebSphere Profile name: Select **AppSrv02**.

    In our example, `AppSrv02` is the WebSphere profile we created. `AppSrv01` is the default profile.
  - Select **RMI** as the Server connection type and admin port. You can see that the ORB bootstrap port is automatically populated with the number **2810**, which is the correct number. Alternatively, select **SOAP**.
  - Select **Run server with resources within the workspace**.
  - Server name: server1
  - We left the remaining fields with their defaults.
  - Click **Next**.



New Server

WebSphere Server Settings

Input settings for the new WebSphere server.

WebSphere profile name: AppSrv02

Server connection type and admin port

● RMI (Designed to improve communication with the server)

ORB bootstrap port: 2810

○ SOAP (Designed to be more firewall compatible)

SOAP connector port: 8880

☑ Run server with resources within the workspace

□ Security is enabled on this server

Current active authentication settings:

User ID:

Password: *****

Server name: server1

Server type

● BASE, Express or unmanaged Network Deployment server

○ Network Deployment server

Network Deployment server name:

The server name is in the form of:
<cell name>/<node name>/<server name>
For example, localhost/localhost/server1. In a cluster environment,
the server name is in the form of:
<cell name>/<cluster name>

Detect Click this button to detect the server type.

Figure 20-5 Define a New Server: WebSphere Server Settings

- In the Add and Remove Projects dialog, we do not select a project at this time, and we click **Finish**.

The server will be created and is displayed in the Servers view. In our example, the server **WebSphere v6.1 @ localhost** is created.

> **Tip:** Open the new server configuration (double-click). You could change the name of the server, for example, **WebSphere v6.1 Server AppSrv02**.

Verify the server

After you have completed defining the server within Application Developer, we recommend that you perform some basic verification steps to ensure that the server is configured properly:

- In the Servers view, right-click the server (WebSphere v6.1 @ localhost) and select **Start** (or click the Start icon ▶).
- Review the server startup output in the console. Also view the server logs `startServer.log` and `SystemOut.log` for errors:

  ```
  <profile_home>/logs/server1/startServer.log
  <profile_home>/logs/server1/SystemOut.log
  ```

- Right-click the server (WebSphere v6.1 @ localhost) and select **Run administrative console**.
- Click **Log in**.
- After accessing several pages of the WebSphere Administrative Console to verify it is working properly, click **Logout** and close the browser.
- Verify that the server stops properly:
  - In the Servers view, right-click the server, and select **Stop**. The server status should change to Stopped (after a while).
  - Verify that the server has really stopped by entering the following command to verify the server status:

    ```
    cd <profile_home>/bin
    serverStatus -all
    ```

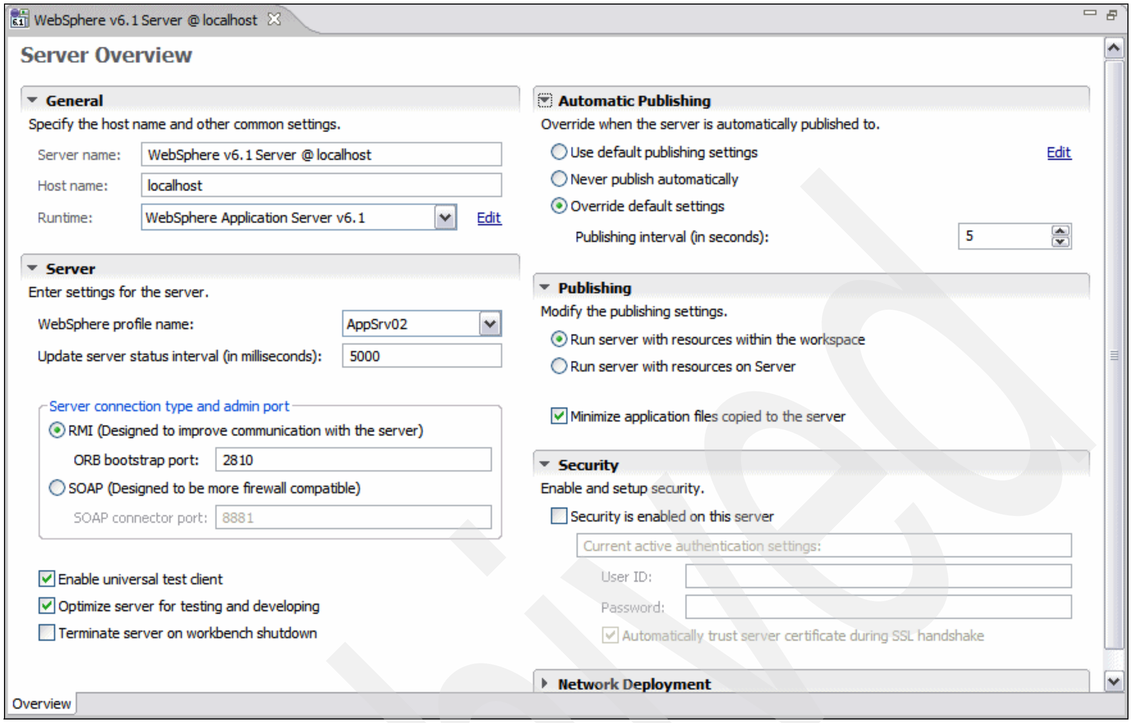
  - The server status output should show that the server has been stopped. If not, stop the server by entering the following command:

    ```
    stopServer server1
    ```

Customizing a server

Once the server has been created in Rational Application Developer, it is very easy to customize the settings:

- In the Servers view, double-click the server you want to customize.
- There are a couple of key settings to point out in the server configuration (Figure 20-6).



WebSphere v6.1 Server @ localhost

Server Overview

General

Specify the host name and other common settings.

Server name: WebSphere v6.1 Server @ localhost

Host name: localhost

Runtime: WebSphere Application Server v6.1 Edit

Server

Enter settings for the server.

WebSphere profile name: AppSrv02

Update server status interval (in milliseconds): 5000

Server connection type and admin port

● RMI (Designed to improve communication with the server)

ORB bootstrap port: 2810

○ SOAP (Designed to be more firewall compatible)

SOAP connector port: 8881

☑ Enable universal test client

☑ Optimize server for testing and developing

□ Terminate server on workbench shutdown

Automatic Publishing

Override when the server is automatically published to.

○ Use default publishing settings Edit

○ Never publish automatically

● Override default settings

Publishing interval (in seconds): 5

Publishing

Modify the publishing settings.

● Run server with resources within the workspace

○ Run server with resources on Server

☑ Minimize application files copied to the server

Security

Enable and setup security.

□ Security is enabled on this server

Current active authentication settings:

User ID:

Password:

☑ Automatically trust server certificate during SSL handshake

Network Deployment

Overview

Figure 20-6 Customize server settings

- **Server**:
  - **WebSphere Profile name**: Select the desired WebSphere profile.
  - Under the **Server connection type and admin port**, use the radio buttons to select whether to use the **RMI** or **SOAP** connection as the communication channel between the development environment and the server. By default, the RMI radio button is enabled when working with a local server. If you are working with a remote server, the SOAP radio button is enabled by default.
  - **Terminate server on workbench shutdown**

    If you want the server to be terminated after the workbench shutdown, you have to select this option. Otherwise the server continues running after you shut down the development environment. The next time you start the IDE, the server is found again in its current state.
- **Automatic publishing**:
  - **Use default publishing settings**: Specifies to use the publishing settings as defined in the Preferences page (**Window** → **Preferences** → **Server**).

  - **Never publish automatically**: Specifies that the workbench should never publish files to the server.
  - **Override default settings**: Specifies that a change to the files running on the server should automatically be published to the server in the next seconds interval, where seconds is the number of seconds you specify in the Publishing interval control. The default value for the publishing interval on a local server is 5 seconds and for a remote server is 60 seconds. If you set the publishing interval to 0 seconds, a change to the files running on the server automatically requests a publish command to occur.

- **Publishing**:
  - **Run server with resources on Server**

    This option installs and copies the full application and its server-specific configuration from the workbench into the directories of the server. The advantage of selecting the Run server with resources on Server setting is you are running your application from the directories of your server and you can edit advanced application-specific settings for your application using the WebSphere Administrative Console. However, when you select to add your application to the server using the Add and Remove Projects wizard, this option takes a longer time to complete than the Run server with resources within the workspace option, as it involves more files copied to the server.

  - **Run server with resources within the workspace**

    This option requests the server to run your application from the workspace. The Run server with resources within the workspace setting is useful when developing and testing your application, as it is designed to operate faster than the Run server with resources on Server option, as fewer files are involved when copied over to the server.

  - **Minimize application files copied to the server**

    This option is designed to optimize the publishing-time on the server by reducing the files copied to the server. In addition to the application files not getting copied into the `installedApps` directory of the server, the application also does not get copied into your server configuration directory. As a result, when configuring your application, you cannot use the WebSphere Administrative Console to edit deployment descriptor information. You are limited to editing the J2EE deployment descriptor files for your application within the workbench. Clear this option if you have to use the WebSphere Administrative Console to edit the deployment descriptor.

- The **Security** settings are discussed in “Configuring security” on page 990.

Share a WebSphere profile among developers

The configuration of a server can be time consuming and error-prone. Once you have configured the server resources, you might want to let other members of the team use the same configuration from their local environment, without duplicating the same effort.

To replicate server configurations across multiple profiles, you can use the **Import server configuration from server** wizard to import the WebSphere v6.1 server profile configuration archive file into a workspace project folder. It performs the same functionality as:

```
wsadmin AdminTask exportWasprofile
```

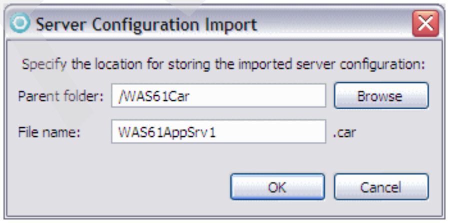
You can use the **Export server configuration to server** wizard to export the WebSphere v6.1 server profile configuration archive file from a workspace project folder to a WebSphere v6.1 server. It performs the same functionality as:

```
wsadmin AdminTask importWasprofile
```

Import the server configuration from the server

To import the server configuration from WebSphere Application Server v6.1, do these steps:

- Configure the data source using **WebSphere Application Server v6.1**, as described in “Configuring the data source in WebSphere Application Server” on page 1313.
- Create a new project to store the configuration archive file:
  - Select **File** → **New** → **Project** → **General** → **Project** and click **Next**.
  - For the project name, enter **WAS61Car**. Click **Finish**.
- In the Servers view, right-click the server for which you want to package its configuration files into a configuration archive file, and select **Import server configuration from server**.
- In the Server Configuration Import wizard (Figure 20-7), specify **/WAS61Car** as the Parent folder and **WAS61AppSrv1** as the File name. Click **OK**.



Server Configuration Import

Specify the location for storing the imported server configuration:

Parent folder: /WAS61Car

File name: WAS61AppSrv1 .car

Figure 20-7 Server Configuration Import dialog

- After the import server configuration process completes, you should see the `WAS61AppSrv1.car` under the `WAS61Car` project.

Export a server configuration to the server

To export the server configuration to WebSphere Application Server v6.1 @ localhost:

- Make sure that WebSphere Application Server v6.1 @ localhost is stopped.
- In the Servers view, right-click WebSphere Application Server v6.1 @localhost and select **Export server configuration to server**.
- In the Server Configuration Export wizard, specify **/WAS61Car** as the Parent folder and **WAS61AppSrv1.car** as the file name. Click **OK**.
- After the export server configuration process completes, start WebSphere Application Server v6.1 @ localhost.
- Right-click WebSphere Application Server v6.1 @localhost and select **Run administrative console**. You should see the data source you configured for WebSphere Application Server v6.1 also appears in the administrative console for WebSphere Application Server v6.1 @ localhost.

Defining a server for each workspace

If you want to switch workspaces and keep applications deployed in the server, you can create a new WebSphere profile for each new Application Developer workspace (see “Creating a new profile using the WebSphere Profile wizard” on page 969), and define only that server in the Servers view (see “Create a server” on page 973).

This approach requires more disk space for each WebSphere profile and server configuration, and requires additional memory if the servers run concurrently.

Adding and removing applications to and from a server

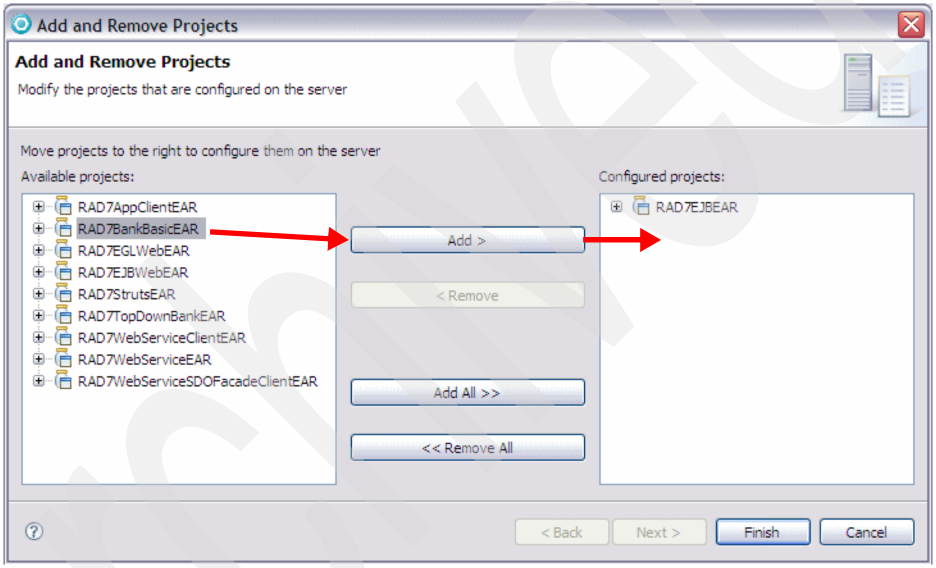
Once the server is configured, it can be further configured with server resources and be used to run applications by adding applications to it.

This section describes how to add an enterprise application to a server. Note that you cannot add Web or EJB projects to a server, only enterprise applications (EAR projects).

Adding an application to the server

To add an application to the server, do these steps:

- Verify that the server is started.
- In the Servers view, right-click a server, and select **Add and Remove Projects**.
- In the Add and Remove Projects dialog, select one of the listed EAR projects and click **Add**. The project appears in the Configured projects column (Figure 20-8).
- Click **Finish**.



Add and Remove Projects

Add and Remove Projects

Modify the projects that are configured on the server

Move projects to the right to configure them on the server

Available projects:

- RAD7AppClientEAR
- RAD7BankBasicEAR
- RAD7EGLWebEAR
- RAD7EJBWebEAR
- RAD7StrutsEAR
- RAD7TopDownBankEAR
- RAD7WebServiceClientEAR
- RAD7WebServiceEAR
- RAD7WebServiceSDOFacadeClientEAR

Configured projects:

- RAD7EJBEAR

Figure 20-8 Add and Remove Projects

- Once an application is added to the server you can run any of the HTML pages or JSPs.

Removing an application from a server

An Application Developer server configuration is essentially a pointer to a server defined in the WebSphere profile. In this section we describe two scenarios for removing published projects from the server.

Remove an application using Application Developer

In most cases, you can remove the project from the test server within Rational Application Developer as follows:

- In the Servers view, right-click the server where the application is published, and select **Add and remove projects**. In the Add and Remove Projects dialog, select the project in the Configured projects list, click **< Remove**, and then click **Finish**.
- Alternatively, expand the server, right-click the project and select **Remove**.

This operation uninstalls the application from the server.

Remove an application using the administrative console

We found it necessary in some cases to uninstall the application using the WebSphere Administrative Console.

For example, if you have published a project in Rational Application Developer to the test server, it has been deployed to the server defined in the WebSphere profile. If you then switch workspaces without first removing the project from the server, you have a broken association between the Rational Application Developer server and the server defined in the WebSphere profile.

To address issues like the scenario described, uninstall the enterprise application from the WebSphere Administrative Console as follows:

- Make sure the server is started.
- Start the WebSphere Administrative Console either by right-clicking the server and selecting **Run administrative console**, or by accessing its URL from a Web browser:

  ```
  http://<appserver_host>:<admin_console_port>/ibm/console
  http://localhost:9060/ibm/console/
  ```

- Click **Log in**.
- Select **Applications** → **Enterprise Applications**.
- Select the desired application to uninstall, and click **Uninstall**.
- When prompted, click **OK**.
- When uninstallation is complete, save the changes.

> **Note:** After you uninstall the application using the WebSphere Administrative Console, you can see that the project still appears in the Configured projects section if you start the Add and Remove Projects wizard. You have to click **Remove** to move the project to the Available projects section.

Configuring application and server resources

In WebSphere Application Server V6.1, application-related properties and data sources can be defined within an enhanced EAR file to simplify application deployment (Figure 20-9). The properties are used by the application after being deployed.

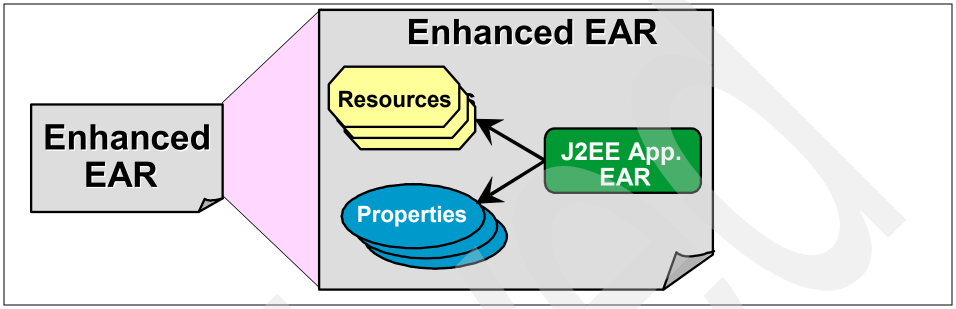


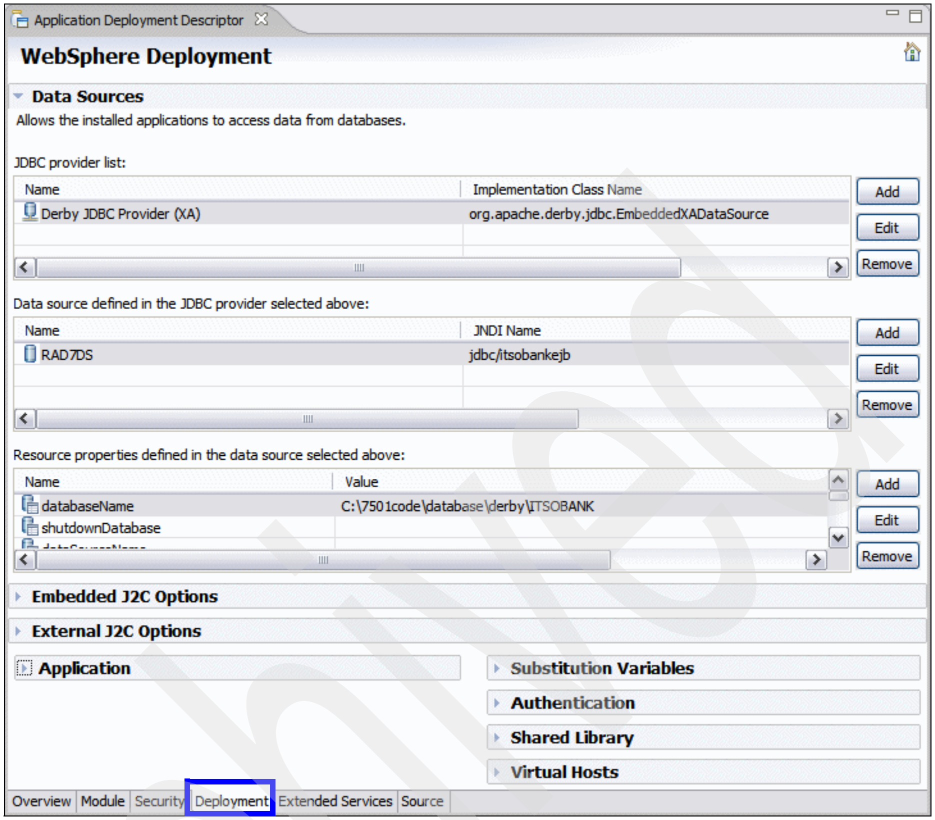
Figure 20-9 Enhanced EAR

The enhanced EAR tooling is provided from the Deployment tab of the enterprise application Deployment Descriptor editor (Figure 20-10). Deployment information is saved under the application `/META-INF/ibmconfig` directory.

Note: The Enhanced EAR editor is used to edit several WebSphere Application Server V6.1 specific configurations, such as data sources, class loader policies, substitution variables, shared libraries, virtual hosts, and authentication settings. It lets you configure these settings with little effort and publish them every time you publish the application.

The upside of the tool is that it makes the testing process simpler and easily repeatable, because the configurations it makes are saved to files that are usually shared in the team repository. Thus, even though it will not let you configure all the runtime settings, it is a good tool for development purposes because it eases the process of configuring the most common settings.

The downside is that the configurations the tool makes are attached to the EAR, and are not visible from administrative console. The console is only able to edit settings that belong to the cluster, node, and server contexts. When you change a configuration using the Enhanced EAR editor, this change is made at the application context. The deployer can still make changes to the EAR file using the Application Server Toolkit (AST), but it is a separate tool. Furthermore, in most cases these settings are dependent on the node the application server is installed in anyway, so it makes little sense to configure them at the application context for deployment purposes.



Application Deployment Descriptor

WebSphere Deployment

Data Sources

Allows the installed applications to access data from databases.

JDBC provider list:

Name	Implementation Class Name
Derby JDBC Provider (XA)	org.apache.derby.jdbc.EmbeddedXADataSource

Data source defined in the JDBC provider selected above:

Name	JNDI Name
RAD7DS	jdbc/itsobankejb

Resource properties defined in the data source selected above:

Name	Value
databaseName	C:\7501code\database\derby\ITSOBANK
shutdownDatabase	

Embedded J2C Options

External J2C Options

Application

Substitution Variables

Authentication

Shared Library

Virtual Hosts

Figure 20-10 Enterprise application deployment descriptor: Enhanced EAR

The server configuration data that you specify in this editor gets embedded within the application itself. This improves the administration process of publishing to WebSphere Application Server v6.x when installing a new application to an existing local or remote WebSphere Server by preserving the existing server configuration.

The following resource types can be added to the enhanced EAR:

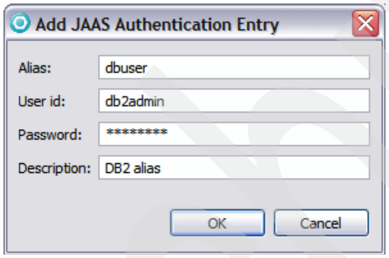
- Virtual Hosts
- JAAS Authorization entries
- Shared Library
- Application class loader settings
- JDBC resources
- Resource adapters
- Substitution variables

Creating a data source in the enhanced EAR

The data sources that support EJB entity beans must be specified before the application can be started. There are several ways to do it, but the easiest is to use the Enhanced EAR editor.

For an example of configuring the enhanced EAR against the Derby database refer to “Configuring the data source for the ITSOBANK” on page 740. In this section, we demonstrate how to create a data source against a DB2 database using enhanced EAR:

- In the Project Explorer open the **RAD7EJBEAR** enterprise application deployment descriptor.
- Select the **Deployment** page.
- Scroll down the page until you find the **Authentication** section. It allows you to define a login configuration used by JAAS.
- Click **Add** to include a new configuration (Figure 20-11).



Add JAAS Authentication Entry

Alias: dbuser

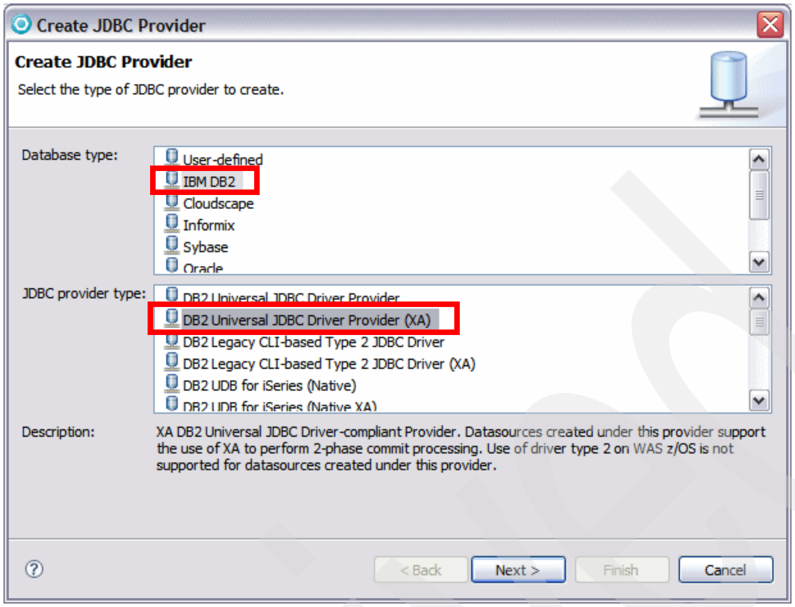
User id: db2admin

Password: ********

Description: DB2 alias

Figure 20-11 JAAS Authentication Entry

- Enter `dbuser` as the Alias, and the appropriate user ID and password for your configuration. Click **OK** to complete the configuration.
- In the Enhanced EAR editor scroll back up to the Data Sources, JDBC provider list section. By default, the Derby JDBC Provider (XA) is predefined.
- Because we are using DB2 for this example, we have to add a DB2 JDBC provider by clicking **Add** right next to the provider list. The Create JDBC Provider dialog opens (Figure 20-12).



Create JDBC Provider

Create JDBC Provider

Select the type of JDBC provider to create.

Database type:

- User-defined
- IBM DB2
- Cloudscape
- Informix
- Sybase
- Oracle

JDBC provider type:

- DB2 Universal JDBC Driver Provider
- DB2 Universal JDBC Driver Provider (XA)
- DB2 Legacy CLI-based Type 2 JDBC Driver
- DB2 Legacy CLI-based Type 2 JDBC Driver (XA)
- DB2 UDB for iSeries (Native)
- DB2 UDB for iSeries (Native XA)

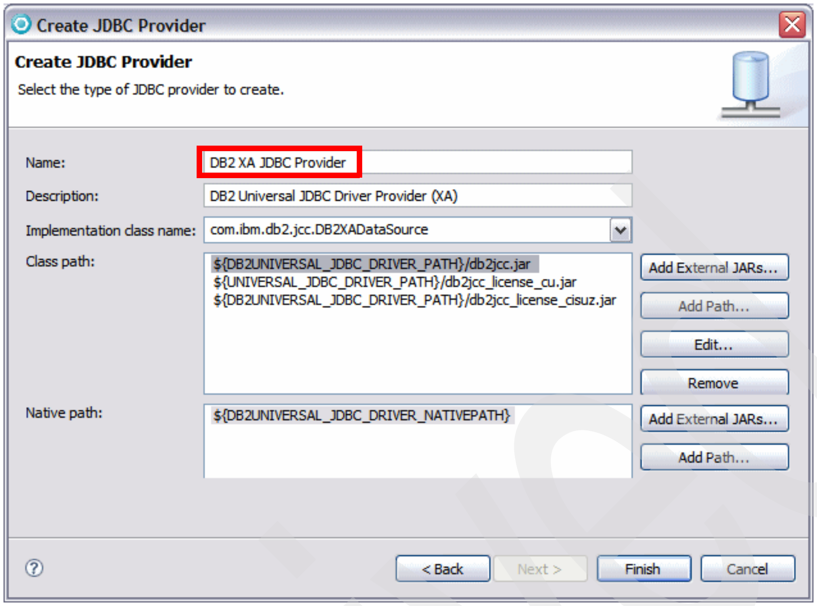
Description: XA DB2 Universal JDBC Driver-compliant Provider. Datasources created under this provider support the use of XA to perform 2-phase commit processing. Use of driver type 2 on WAS z/OS is not supported for datasources created under this provider.

Figure 20-12 Creating a JDBC provider (page 1)

- Select **IBM DB2** as the database type. Then select **DB2 Universal JDBC Driver Provider (XA)** as the provider type.

> **Note:** Note that for our development purposes, the DB2 Universal JDBC Driver Provider (non XA) would work fine, because we do not require XA (two-phase commit) capabilities.

- Click **Next** to proceed to the second page (Figure 20-13).



Create JDBC Provider

Select the type of JDBC provider to create.

Name: DB2 XA JDBC Provider

Description: DB2 Universal JDBC Driver Provider (XA)

Implementation class name: com.ibm.db2.jcc.DB2XADataSource

Class path:

```
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc.jar
${UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cu.jar
${DB2UNIVERSAL_JDBC_DRIVER_PATH}/db2jcc_license_cisuz.jar
```

Native path:

```
${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}
```

Figure 20-13 Creating a JDBC provider (page 2)

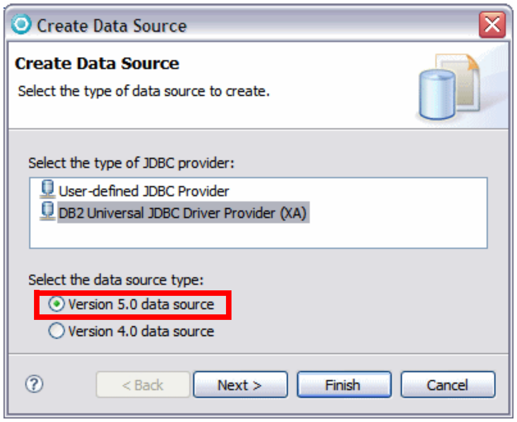
- In this page, enter DB2 XA JDBC Provider as the name. You also have to replace the following variables with the actual location of your DB2 JDBC driver path:

  ```
  ${DB2UNIVERSAL_JDBC_DRIVER_PATH}
  ${UNIVERSAL_JDBC_DRIVER_PATH}
  ${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}
  ```

  For example, you have to use `C:\Program Files\IBM\SQLLIB\Java` instead of `${DB2UNIVERSAL_JDBC_DRIVER_PATH}`.

> **Note:** Alternatively, you can also substitution variables section of the deployment page to define the value for these DB2 variables.

- Click **Finish**.
- With the new DB2 provider selected, click **Add** next to the defined data sources list (Figure 20-14). Select **DB2 Universal JDBC Driver Provider (XA)** from the JDBC provider type list, and select **Version 5.0 data source**.



Create Data Source

Create Data Source

Select the type of data source to create.

Select the type of JDBC provider:

- User-defined JDBC Provider
- DB2 Universal JDBC Driver Provider (XA)

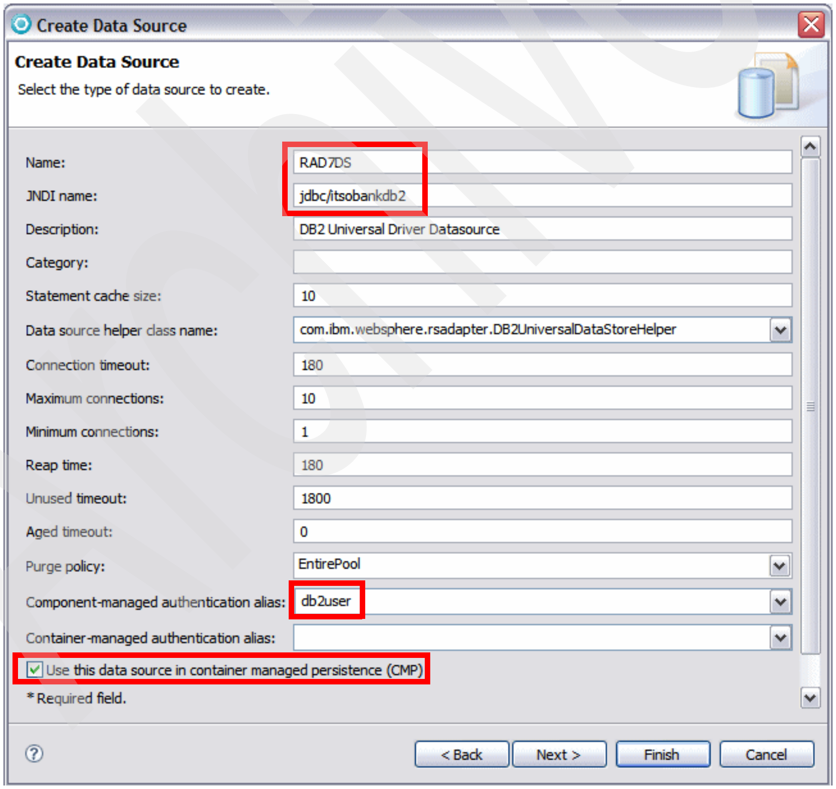
Select the data source type:

● Version 5.0 data source

○ Version 4.0 data source

Figure 20-14 Create a data source (1)

- Click **Next** to continue to the next page (Figure 20-15).



Create Data Source

Create Data Source

Select the type of data source to create.

Name: RAD7DS

JNDI name: jdbc/itsobankdb2

Description: DB2 Universal Driver Datasource

Category:

Statement cache size: 10

Data source helper class name: com.ibm.websphere.rsadapter.DB2UniversalDataStoreHelper

Connection timeout: 180

Maximum connections: 10

Minimum connections: 1

Reap time: 180

Unused timeout: 1800

Aged timeout: 0

Purge policy: EntirePool

Component-managed authentication alias: db2user

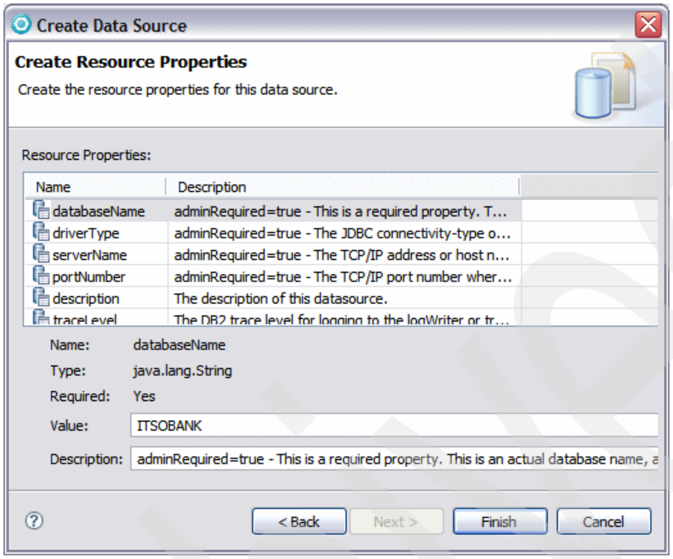
Container-managed authentication alias:

☑ Use this data source in container managed persistence (CMP)

* Required field.

Figure 20-15 Create a data source (2)

- Enter **RAD7DS** as the data source name and **jdbc/itsobankdb2** as the JNDI name, fill out a description if you want, and select the **dbuser** alias for Component-managed authentication alias. Click **Next** to proceed to next wizard's last page (Figure 20-16).



Create Data Source

Create Resource Properties

Create the resource properties for this data source.

Resource Properties:

Name	Description
databaseName	adminRequired=true - This is a required property. T...
driverType	adminRequired=true - The JDBC connectivity-type o...
serverName	adminRequired=true - The TCP/IP address or host n...
portNumber	adminRequired=true - The TCP/IP port number wher...
description	The description of this datasource.
traceLevel	The DB2 trace level for logging to the logWriter or tr...

Name: databaseName

Type: java.lang.String

Required: Yes

Value: ITSOBANK

Description: adminRequired=true - This is a required property. This is an actual database name, a

Figure 20-16 Create a data source (3)

- Finally, just set the `databaseName` property value to **ITSOBANK**, and click **Finish** to conclude the wizard.
- Save the deployment descriptor.

Configuring server resources

Within Application Developer, the WebSphere Administrative Console is the primary interface for configuring WebSphere Application Server V6.1 test servers (local and remote). Complicated resource configurations such as messaging resources can only be configured using the WebSphere Administrative Console.

There are a couple of methods for accessing the WebSphere Administrative Server. In either case the WebSphere Application Server V6.1 test server must be started.

Once the WebSphere Application Server V6.1 test server is started, you can right-click the server and select **Run administrative console**. Alternatively, once the server is started you can enter one of the following URLs in a Web browser:

```
http://localhost:9060/ibm/console
http://localhost:9060/admin
```

Port 9060 is the port defined for the WebSphere Administrative Console for the application server in the WebSphere profile.

Configuring security

If the WebSphere Application Server v6.1 runtime environment has administrative security enabled, you have to communicate the administrative settings from your development environment to the runtime server. In the Workbench you have to specify that security is enabled in the runtime environment, and provide the user ID and password in the server editor to the secured server. If you are working with a secured WebSphere Application Server Version 6.1.x, you have to establish a trust between the development workbench of this product and the server.

Note: For more information on configuring WebSphere security, refer to *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316.

In this section, we show you how to enable administrative security with local operating system registry using the WebSphere Administrative Console and how to communicate the administrative settings from the development environment to the runtime server.

Configuring security in the server

First we configure that the server runs with security enabled:

- Start the test server (WebSphere Application Server v6.1).
- Right-click the server and select **Run administrative console**.
- Click **Log in**.
- Expand **Security** → **Secure administration, applications, and infrastructure** (Figure 20-17).

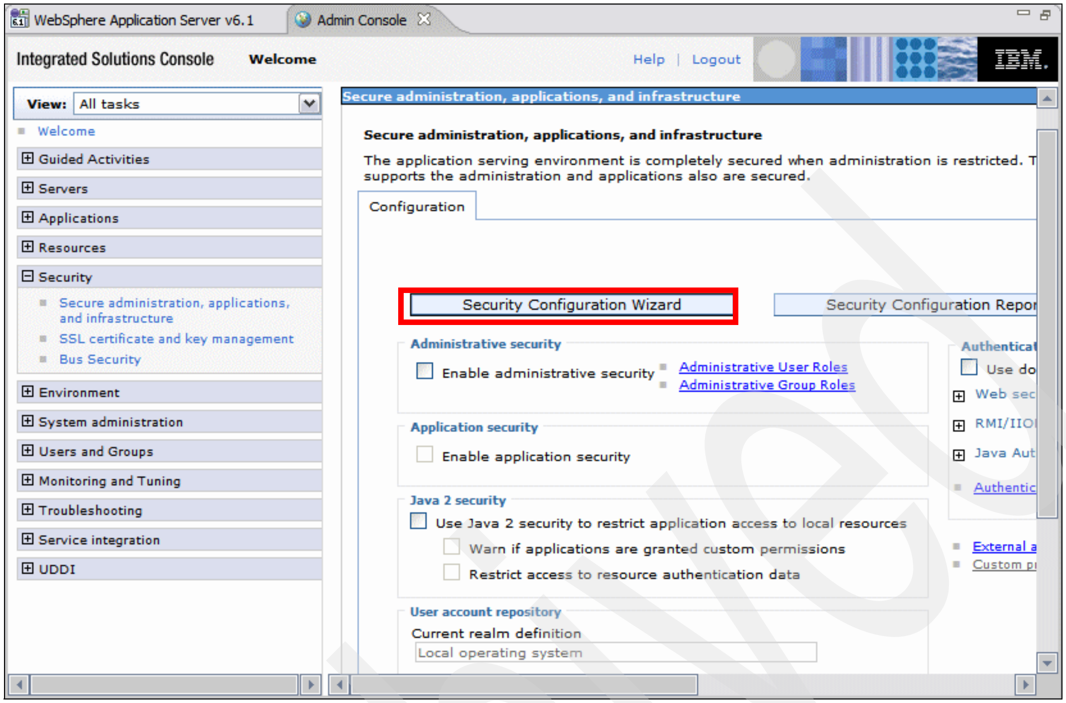

WebSphere Application Server v6.1
Admin Console
Integrated Solutions Console
Welcome
Help | Logout
IBM.
View: All tasks
Welcome
Guided Activities
Servers
Applications
Resources
Security
Secure administration, applications, and infrastructure
SSL certificate and key management
Bus Security
Environment
System administration
Users and Groups
Monitoring and Tuning
Troubleshooting
Service integration
UDDI
Secure administration, applications, and infrastructure
Secure administration, applications, and infrastructure
The application serving environment is completely secured when administration is restricted. T
supports the administration and applications also are secured.
Configuration
Security Configuration Wizard
Security Configuration Repor
Administrative security
☐ Enable administrative security
Administrative User Roles
Administrative Group Roles
Application security
☐ Enable application security
Java 2 security
☐ Use Java 2 security to restrict application access to local resources
☐ Warn if applications are granted custom permissions
☐ Restrict access to resource authentication data
User account repository
Current realm definition
Local operating system
Authenticat
☐ Use do
Web sec
RMI/IIO
Java Aut
Authentic
External a
Custom p

Figure 20-17 Administrative security

- Click **Security Configuration wizard**.
- In the Specify extent of protection page, click **Next**.
- In the Select user repository page, select **Local operating system**.
- In the Configure user repository page, enter the primary administrative user name, click **Next**.

Note: The specified user must have the required privileges in Windows, such as the permission to log on as service. For more information see section 2.3 of the *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316.

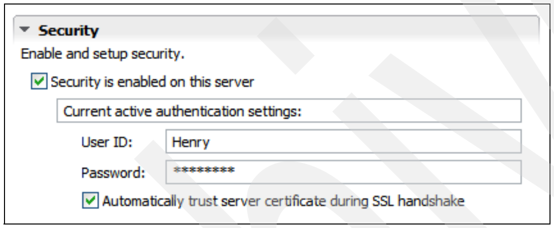
- Click **Finish**, then click **Save**, then click **Logout** to log off from administrative console.
- Stop the **WebSphere Application Server v6.1**.

The next time you start the server and open the administrative console, you are prompted for a valid user ID and password.

Configuring security in the Workbench

We have to edit the server configuration to specify that security is enabled:

- In the Servers view, double-click **WebSphere Application Server v6.1**. The server configuration editor opens.
- Expand the **Security** section.
- Select the **Security is enabled on this server**.
- The User ID and Password fields specify the administrator user of the WebSphere Administrative Console. These values must be the same as those entered in the Security Configuration wizard dialog, as described in “Configuring security in the server” on page 990.
- Make sure **Automatically trust server certificate during SSL handshake** is selected (Figure 20-18).



Security

Enable and setup security.

☑ Security is enabled on this server

Current active authentication settings:

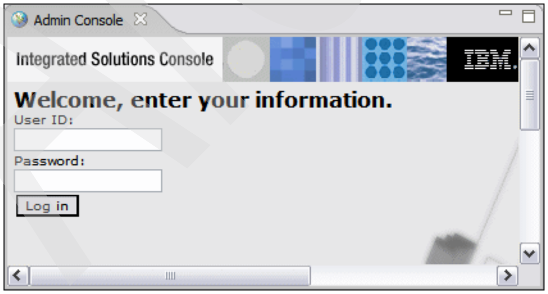
User ID: Henry

Password: ********

☑ Automatically trust server certificate during SSL handshake

Figure 20-18 Security setting in server editor

- Save and close the server configuration editor.
- Start the server and run the administrative console.
- You should see the secured administrative console (Figure 20-19). Enter your user ID and password to log in.



Welcome, enter your information.

User ID:

Password:

Figure 20-19 Secured Administrative Console

Developing automation scripts

Scripting is a non-graphical alternative that you can use to configure and manage the WebSphere Application Server. The WebSphere administrative scripting tool (wsadmin) is a non-graphical command interpreter environment enabling you to run administrative operations on a server in a scripting language.

There are five wsadmin objects available when you use scripts:

- AdminControl: Use to run operational commands.
- AdminConfig: Use to run configurational commands to create or modify WebSphere Application Server configurational elements.
- AdminApp: Use to administer applications.
- AdminTask: Use to run administrative commands.
- Help: Use to obtain general help.

The WebSphere administrative scripting program (wsadmin) supports two scripting languages: Java Tcl (Jacl) and Java Python (Jython).

With the Version 6.1 release of WebSphere Application Server, IBM announced the start of the deprecation process for the Jacl syntax associated with wsadmin.

In this chapter, we show you how to create a Jython project and Jython script file, how to edit the Jython script file, and how to run it.

Creating a Jython project

To create a Jython project, follow these instructions:

- Select **File** → **New** → **Project** → **Jython** → **Jython Project** and click **Next**.
- For the Project name, type **RAD7Jython** and click **Finish**.

Creating Jython script files

To create a Jython script file, follow these instructions:

- Select **File** → **New** → **Other** → **Jython** → **Jython Script File** and click **Next**.
- Specify **/RAD7Jython** as the Parent folder and **listJDBCProviders.py** as the File name.
- Click **Finish**.

Editing Jython script files

The Jython editor is the tool for editing Jython scripts. The Jython editor has many text editing features, such as content assist, syntax highlighting, unlimited undo or redo, and automatic tab indentation.

Type the code shown in Example 20-1. It lists all defined JDBC providers in the WebSphere Application Server. During typing, you can use content assists by pressing Ctrl+Space. You can find the code is:

```
c:\7501code\Jython\listJDBCProviders.py
```

Example 20-1 List JDBC providers using a Jython script (listJDBCProviders.py)

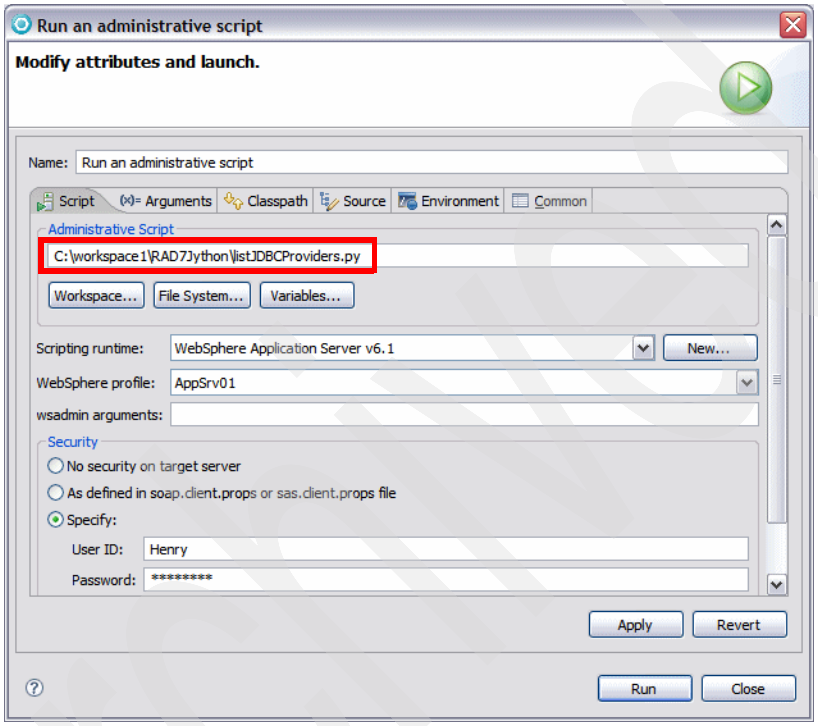
```
def showJdbcProviders():
   providerEntries = AdminConfig.list("JDBCProvider")
   # split long line of entries into individual entries in list
   providerEntryList = providerEntries.split(lf)
   # print contents of list
   for provider in providerEntryList:
      print provider

AdminConfig.reset()
cell = AdminControl.getCell()
node = AdminControl.getNode()
lf = java.lang.System.getProperty("line.separator")
slash = java.lang.System.getProperty("file.separator")
print "System information: Cell=" + cell
print "System information: Node=" + node
showJdbcProviders()
```

Running administrative script files on WebSphere Application Server

You can run administrative scripts from within Application Developer, without having to switch to the non-graphical wsadmin command. To run the Jython script, follow these instructions:

- Make sure the server is started.
- Right-click the **WebSphere Application Server v6.1** and select **Run administrative script**.
- The Script page is the main page of the WebSphere Administrative Script Launcher (Figure 20-20):
  - In the Administrative Script field click **Workspace**.
  - In the File Selection dialog, expand the **RAD7Jython** project and select **listJDBCProviders.py**, click **OK**.

- For Scripting runtime select **WebSphere Application Server v6.1**.
- For WebSphere profile select **AppSrv01**.
- In the Security section, if the administrative security is still enabled with local operating system registry, as you did in “Configuring security” on page 990, select **Specify:** and enter the User ID and Password.



Run an administrative script

Modify attributes and launch.

Name: Run an administrative script

Administrative Script

C:\workspace1\RAD7Jython\listJDBCProviders.py

Scripting runtime: WebSphere Application Server v6.1

WebSphere profile: AppSrv01

wsadmin arguments:

Security

○ No security on target server

○ As defined in soap.client.props or sas.client.props file

● Specify:

User ID: Henry

Password: *********

Figure 20-20 Run an administrative script

- Click **Apply** to save the configuration.
- Click **Run** to run the Jython script. You should see console output as listed below:

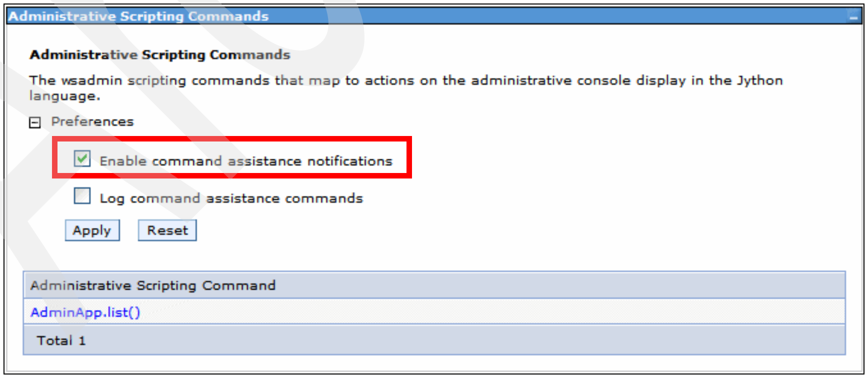
```
WASX7209I: Connected to process "server1" on node KLCHL2YNode01 using
SOAP connector;  The type of process is: UnManagedProcess
System information: Cell=KLCHL2YNode01Cell
System information: Node=KLCHL2YNode01
"Derby JDBC Provider (XA)(cells/KLCHL2YNode01Cell/nodes/KLCHL2YNode01/
      servers/server1|resources.xml#builtin_jdbcprovider)"
"Derby JDBC Provider (XA)(cells/KLCHL2YNode01Cell|resources.xml
                   #builtin_jdbcprovider)"
"Derby JDBC Provider(cells/KLCHL2YNode01Cell/nodes/KLCHL2YNode01/
      servers/server1|resources.xml#JDBCProvider_1182202633563)"
```

Generating WebSphere admin commands for Jython scripts

You can use the WebSphere Administration Command assist tool to generate WebSphere administrative (wsadmin) commands for Jython scripting language as you interact with the WebSphere Administrative Console. When you perform server operations in the WebSphere Administrative Console, the WebSphere Administration Command assist tool captures and displays the wsadmin commands issued. You can transfer the output from the WebSphere Administration Command view directly to a Jython editor, enabling you to develop Jython scripts based on actual console actions.

To generate wsadmin commands as you interact with the WebSphere Administrative Console, do these steps:

- Enable the command assistance notification option in the WebSphere Administrative Console:
  - Make sure the server is started.
  - Right-click the **WebSphere Application Server v6.1** and select **Run administrative console**.
  - Specify the User ID and the Password if the server is secured. Click **Log in**.
  - On the left-pane, expand **Applications** → **Enterprise Applications**
  - Scroll to the right of the **Enterprise Applications** page and under the **Command Assistance** section, click the **View administrative scripting command for last action** link.
  - Expand **Preferences**, select the **Enable command assistance notifications**. Click **Apply** (Figure 20-21) and close the window.



Administrative Scripting Commands

Administrative Scripting Commands

The wsadmin scripting commands that map to actions on the administrative console display in the Jython language.

⊟ Preferences

☑ Enable command assistance notifications

☐ Log command assistance commands

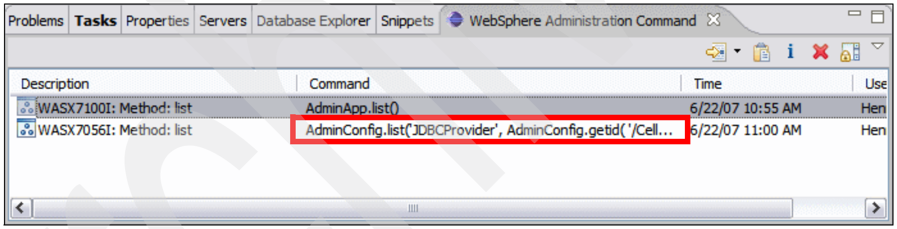
Administrative Scripting Command
AdminApp.list()
Total 1

Figure 20-21 Administrative Scripting Commands Window

- In the Servers view, right-click the server and select **WebSphere administration command assist**. The WebSphere Administration Command view opens. The view might open on the top-right, but you can move it to a better place, such as where the Servers and Console views are.
- In the Select Server to Monitor pull-down, select **WebSphere Application Server v6.1.** To make sure the sever is selected, you can click again and you should see a check mark for **WebSphere Application Server v6.1**.
- In the WebSphere Administrative Console select **Applications** → **Enterprise Applications** in the left-pane. You should see that the WebSphere Administration Command view is populated with a wsadmin command for Jython.

> **Note:** There might be a few seconds delay before you see the Jython command in the WebSphere Administration Command view.

- In the WebSphere Administrative Console, left-pane, select **Resources** → **JDBC** → **JDBC Providers.** You should see another command in the WebSphere Administration Command view (Figure 20-22):



Description	Command	Time	Use
WASX7100I: Method: list	AdminApp.list()	6/22/07 10:55 AM	Hen
WASX7056I: Method: list	AdminConfig.list('JDBCProvider', AdminConfig.getid('/Cell...	6/22/07 11:00 AM	Hen

Figure 20-22 WebSphere Administration Command Assist

- Create a Jython script file in the **RAD7Jython** project and name it **CommandAssist.py**.
- To transfer the wsadmin commands generated in the WebSphere Administration Command view to the Jython script:
  - Make sure the Jython editor for **CommandAssist.py** is open.
  - In the Jython editor place the **cursor** to the position at the bottom of the editor.
  - In the WebSphere Administration Command use the **Shift** key to select both commands. Right-click the commands in the WebSphere Administration Command view and select **Insert.**

- In the editor, you should see commands similar to these:

```
AdminApp.list()
AdminConfig.list(\
   'JDBCProvider', AdminConfig.getid( \
   '/Cell:KLCHL2YNode01Cell/'))
```

- Add print method in front of these two commands so that it looks as follows:

```
print AdminApp.list()
print AdminConfig.list(\
   'JDBCProvider', AdminConfig.getid( \
   '/Cell:KLCHL2YNode01Cell/'))
```

- Save the file.
- Right-click **CommandAssist.py** → **Run As** → **Administrative Script**.
- In the Script page of the WebSphere Administrative Script Launcher, select the following items:
  - For Scripting runtime, select **WebSphere Application Server v6.1**.
  - For WebSphere profile, select **AppSrv01**.
  - In the Security section, if administrative security is enabled, enter the User ID and Password.
- Click **Apply** and then click **Run** to run the script. You should see console output as listed below:

```
WASX7209I: Connected to process "server1" on node KLCHL2YNode01 using
SOAP connector;  The type of process is: UnManagedProcess
DefaultApplication

IBMUTC

ivtApp

query
"Derby JDBC Provider
(XA)(cells/KLCHL2YNode01Cell/nodes/KLCHL2YNode01/servers/
         server1|resources.xml#builtin_jdbcprovider)"

"Derby JDBC Provider (XA)(cells/KLCHL2YNode01Cell|resources.xml#
         builtin_jdbcprovider)"

"Derby JDBC
Provider(cells/KLCHL2YNode01Cell/nodes/KLCHL2YNode01/servers/
         server1|resources.xml#JDBCProvider_1182202633563)"
```

The command assist feature is great when you are learning Jython to create scripts for future use.

Debugging Jython scripts

The Jython debugger enables you to detect and diagnose errors in Jython scripts that are run on WebSphere Application Server.

For an example of debugging the sample `listJDBCProviders` script (Example 20-1 on page 994), refer to “Jython debugger” on page 1068.

Jython script for application deployment

For a complete Jython script that creates a JDBC provider and a data source, and installs and starts an enterprise application, refer to “Automated deployment using Jython based wsadmin scripting” on page 1141.

More information

For more information, consult these IBM Redbooks publications:

- *WebSphere Application Server V6.1: Systems Management and Configuration*, SG24-7304
- *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316

For more information on Jython, refer to these documents:

- Get to know Jython:

  http://www.ibm.com/developerworks/java/library/j-alj07064/
- WebSphere InfoCenter scripting:

  http://publib.boulder.ibm.com/infocenter/wasinfo/v6r0/index.jsp?topic=/com.ibm.websphere.express.doc/info/exp/ae/cxml_jython.html

21

Test using JUnit

The Application Developer test framework is built on the Eclipse Test & Performance Tools Platform (TPTP), which extends the Eclipse Hyades Tool Project. It contains monitoring, tracing, profiling, and testing tools. *JUnit* is one of the testing tools and can be used for automated component testing. TPTP also includes profiling capabilities for memory, performance, and other execution time code analysis. We explore profiling in Chapter 25, “Profile applications” on page 1155.

In this chapter we introduce application testing concepts, and provide an overview on TPTP and JUnit, as well as the features of Application Developer for testing. In addition, we include working examples to demonstrate how to create, and run component tests using JUnit, as well as demonstrate how to test Web applications.

The chapter is organized into the following sections:

- Introduction to application testing
- JUnit testing [without using TPTP]
- JUnit testing using TPTP
- Web application testing

Note: In the previous version of Application Developer there was a feature called *automated component testing*. This feature has been removed in Application Developer V7, as it was rarely used by customers.

Introduction to application testing

Altough the focus of this chapter is on component testing, we have included an introduction to testing concepts, such as test phases and environments, to put into context where component testing fits within the development cycle. Next, we provide an overview on the TPTP and JUnit testing frameworks. The remainder of the chapter provides a working example of using the features of TPTP and JUnit within Application Developer.

Test concepts

Within a typical development project, there are various types of testing performed during the different phases of the development cycle. Project requirements based on size, complexity, risks, and costs determine the levels of testing to be performed. The focus of this chapter is on component testing and unit testing.

Test phases

In this section we outline the key test phases and categorize them.

Unit test

Unit tests are informal tests that are generally executed by the developers of the application code. They are often quite low-level in nature, and test the behavior of individual software components, such as individual Java classes, servlets, or EJBs.

Because unit tests are usually written and performed by the application developer, they tend to be white-box in nature—that is, they are written using knowledge about the implementation details and test-specific code paths. This is not to say that all unit tests have to be written this way; one common practice is to write the unit tests for a component based on the component specification, before developing the component itself. Both approaches are valid, and you might want to make use of both when defining your own unit testing policy.

Component test

Component tests are used to verify particular components of the code before they are integrated into the production code base. Component tests can be performed on the development environment. Within the context of Application Developer, a developer configures a test environment and supporting testing tools such as JUnit. Using the test environment, you can test customized code including JavaBeans, Enterprise JavaBeans, and JavaServer Pages without having to deploy this code to a runtime system.

Build verification test (BVT)

Members of the development team check their source code into the source control tool, and mark the components as part of a build level. The build team is responsible for building the application in a controlled environment, based on the source code available in the source control system repository. The build team extracts the source code from the source control system, executes scripts to compile the source code, packages the application, and tests the application build.

The test run on the application of the build produced is called a *build verification test* (BVT). BVT is a predefined and documented test procedure to ensure that basic elements of the application are working properly, before accepting the build and making it available to the test team for function verification test (FVT) and/or system verification test (SVT).

Function verification test (FVT)

These tests are used to verify individual functions of an application. For example, you can verify if the taxes are being calculated properly within a banking application.

> **Note:** Within the Rational product family, the **IBM Rational Function Tester** is an ideal choice for this type of testing.

System verification test (SVT)

System verification tests are used to test a group of functions. A dedicated test environment should be used with the same system and application software as the target production environment. To get the best results from such tests, you have to find the most similar environment and involve as many components as possible, and verify that all functions are working properly in an integrated environment.

> **Note:** Within the Rational product family, the **IBM Rational Manual Tester** is an ideal choice for this type of testing.

Performance test

Performance tests simulate the volume of traffic that you expect to have for the application(s) and ensure that the system will support this stress, and to determine if the system performance is acceptable.

Note: Within the Rational product family, the **IBM Rational Performance Tester** is an ideal choice for this type of testing.

Customer acceptance test

This is a level of testing in which all aspects of an application or system are thoroughly and systematically tested to demonstrate that it meets business and non-functional requirements. The scope of a particular acceptance test is defined in the acceptance test plan.

Test environments

When sizing a project, it is important to consider the system requirements for the test environments. Here is a list of some common test environments that are used.

- **Component test environment**: This is often the development system and the focus of this chapter. In larger projects, we recommend that development teams have a dedicated test environment to be used as a sandbox to integrate the components of the team members, before putting the code into the application build.
- **Build verification test environment**: This test environment is used to test the application produced from a controlled build. For example, a controlled build should have source control, build scripts, and packaging scripts for the application. The build verification team runs a subset of tests, often known as regression tests, to verify basic functionality of the system that is representative to a wider scale of testing.
- **System test environment**: This test environment is used for FVT and SVT to verify the functionality of the application and integrate it with other components. There can be many test environments with teams of people focused on different aspects of the system.
- **Staging environmen**t: The staging environment is critical for all sizes of organizations. Prior to deploying the application to production, the staging environment is used to simulate the production environment. This environment can be used to perform customer acceptance tests.
- **Production environment**: This is the live runtime environment that customers will use to access the e-commerce Web site. In some cases, customer acceptance testing might be performed on the production environment. Ultimately, the customers test the application. You must have a process to track customer problems and to implement fixes to the application within this environment.

Calibration

By definition, *calibration* is a set of gradations that show positions or values. When testing, it is important to establish a base line for such things as performance and functionality for regression testing. For example, when regression testing, you have to provide a set of tests that have been exercised on previous builds of the application, before you test the new build. This is also very important when setting entrance and exit criteria.

Test case execution and recording results

When trying to determine why a piece of functionality of a component within an application has become broken, it is useful to know when the test case last executed successfully. Recording the successes and failures of test cases for a designated application build is essential to having an accountable test organization and a quality application.

Benefits of unit and component testing

It might seem obvious as to why we want to test our code. Unfortunately, many people do not understand the value of testing. Simply put, we test our code and applications to find defects in the code, and to verify that changes we have made to existing code do not break that code. In this section, we highlight the key benefits of unit and component testing.

Perhaps it is more useful to look at the question from the opposite perspective, that is, why developers *do not* perform unit tests. In general, the simple answer is because it is too hard or because nobody forces them to. Writing an effective set of unit tests for a component is not a trivial undertaking. Given the pressure to deliver that many developers find themselves subjected to, the temptation to postpone the creation and execution of unit tests in favor of delivering code fixes or new functionality is often overwhelming.

In practice, this usually turns out to be a false economy, because developers very rarely deliver bug-free code, and the discovery of code defects and the costs associated with fixing them are simply pushed further out into the development cycle, which is inefficient. The best time to fix a code defect is immediately after the code has been written, while it is still fresh in the developer's mind.

Furthermore, a defect discovered during a formal testing cycle must be written up, prioritized, and tracked. All of these activities incur cost, and might mean that a fix is deferred indefinitely, or at least until it becomes critical.

Based on our experience, we believe that encouraging and supporting the development and regular execution of unit test cases ultimately leads to significant improvements in productivity and overall code quality. The creation of unit test cases does not have to be a burden. If done properly, developers can find the intellectual challenge quite stimulating and ultimately satisfying. The thought process involved in creating a test can also highlight shortcomings in a design, which might not otherwise have been identified when the main focus is on implementation.

We recommend that you take the time to define a unit testing strategy for your own development projects. A simple set of guidelines, and a framework that makes it easy to develop and execute tests, pays for itself surprisingly quickly.

Once you have decided to implement a unit testing strategy for your project, the first hurdles to overcome are the factors that dissuade developers from creating and running unit tests in the first place. A testing framework can help by making it easier to:

- Write tests
- Run tests
- Rerun a test after a change

Tests are easier to write, because a lot of the infrastructure code that you require to support every test is already available. A testing framework also provides a facility that makes it easier to run and re-run tests, perhaps via a GUI. The more often a developer runs tests, the sooner the problems can be located and fixed, because the difference between the code that last passed a unit test, and the code that fails the test, is smaller.

Benefits of testing frameworks

Testing frameworks also provide other benefits such as these:

- **Consistency**: Every developer is using the same framework, all of your unit tests work in the same way, can be managed in the same way, and report results in the same format.
- **Maintenance**: A framework has already been developed and is already in use in a number of projects, and you spend less time maintaining your testing code.
- **Ramp-up time**: If you select a popular testing framework, you might find that new developers coming into your team are already familiar with the tools and concepts involved.
- **Automation**: A framework can offer the ability to run tests unattended, perhaps as part of a daily or nightly build.

Note: A common practice in many development environments is the use of daily builds. These automatic builds are usually initiated in the early hours of the morning by a scheduling tool.

Test & Performance Tools Platform (TPTP)

TPTP provides an open platform supplying powerful frameworks and services that allow software developers to build unique test and performance tools.

TPTP addresses the entire test and performance life cycle, from early testing to production application monitoring, including test editing and execution, monitoring, tracing and profiling, and log analysis capabilities.

Within the scope of Application Developer, it includes the following types of testing:

- JUnit testing
- Manual testing
- Performance testing of Web applications

Although each of these areas of testing has its own unique set of tasks and concepts, two sets of topics are common to all three types:

- Providing tests with variable data
- Creating a test deployment

Further information about TPTP can be found here:

```
http://www.eclipse.org/tptp
```

JUnit testing [without using TPTP]

This section provides JUnit fundamentals as well as a working example of how to create and run a JUnit test within Application Developer.

The JUnit home page is located at:

```
http://junit.sourceforge.net/
```

JUnit fundamentals

A unit test is a collection of tests designed to verify the behavior of a single unit. A unit is always the smallest testable part of an application. In object-oriented programming, the smallest unit is always a class.

JUnit tests your class by scenario, and you have to create a testing scenario that uses the following elements:

- Instantiate an object
- Invoke methods
- Verify assertions

> **Note:** An assertion is a statement that allows you to test the validity of any assumptions made in your code.

What is new in JUnit 4.x?

For the first time in its history, JUnit 4 has significant changes to previous releases. It simplifies testing by using the annotation feature, which was introduced in Java 5 (JDK 1.5). One helpful feature is that tests no longer rely on subclassing, reflection, and naming conventions. JUnit 4 allows you to mark any method in any class as an executable test case, just by adding the `@Test` annotation in front of the method.

Test case class

Example 21-1 lists a simple JUnit 3.8.1 test case class.

Example 21-1 Simple JUnit 3.8.1 test case class

```
package itso.rad7.bank.test.junit;

import itso.rad7.bank.exception.InvalidAccountException;
import itso.rad7.bank.exception.InvalidCustomerException;
import itso.rad7.bank.ifc.Bank;
import itso.rad7.bank.impl.ITSOBank;
import itso.rad7.bank.model.Account;
import itso.rad7.bank.model.Customer;
import junit.framework.TestCase;

public class ITSOBank381Test extends TestCase {

   private Bank bank = null;
   private static final String ACCOUNT_NUMBER = "001-999000777";
   private static final String CUSTOMER_SSN = "111-11-1111";
   public ITSOBank381Test(String s) {
      super(s);
   }

   protected void setUp() throws Exception {
      super.setUp();
      // Instantiate objects
      this.bank = ITSOBank.getBank();
   }

   protected void tearDown() throws Exception {
      super.tearDown();
   }

   public final void testSearchAccountByAccountNumber() {
      // Invoke a method
      try {
         Account bankAccount = this.bank
               .searchAccountByAccountNumber(ITSOBank381Test.ACCOUNT_NUMBER);
         // Verify an assertion
         assertEquals(bankAccount.getAccountNumber(),
               ITSOBank381Test.ACCOUNT_NUMBER);
      } catch (InvalidAccountException e) {
         e.printStackTrace();
      }
   }

   public final void testSearchCustomerBySsn() {
      // Invoke a method
      try {
         Customer bankCustomer = this.bank
               .searchCustomerBySsn(ITSOBank381Test.CUSTOMER_SSN);
         // Verify an assertion
         assertEquals(bankCustomer.getSsn(), ITSOBank381Test.CUSTOMER_SSN);
      } catch (InvalidCustomerException e) {
         e.printStackTrace();
      }
   }
}
```

Example 21-2 lists the same test case class again, but this time using JUnit 4.x.

Example 21-2 Simple JUnit 4.x test case class

```
package itso.rad7.bank.test.junit;

import static org.junit.Assert.assertEquals;
import org.junit.After;
import org.junit.Before;
import org.junit.Test;

import itso.rad7.bank.exception.InvalidAccountException;
import itso.rad7.bank.exception.InvalidCustomerException;
import itso.rad7.bank.ifc.Bank;
import itso.rad7.bank.impl.ITSOBank;
import itso.rad7.bank.model.Account;
import itso.rad7.bank.model.Customer;

public class ITSOBank4Test {

   private Bank bank = null;
   private static final String ACCOUNT_NUMBER = "001-999000777";
   private static final String CUSTOMER_SSN = "111-11-1111";

   @Before
   public void setUp() {
      // Instantiate objects
      this.bank = ITSOBank.getBank();
   }

   @After
   public void tearDown() {}

   @Test
   public final void testSearchAccountByAccountNumber() {
      try {
         // Invoke a method
         Account bankAccount = this.bank
               .searchAccountByAccountNumber(ITSOBank4Test.ACCOUNT_NUMBER);
         // Verify an assertion
         assertEquals(bankAccount.getAccountNumber(),
               ITSOBank4Test.ACCOUNT_NUMBER);
      } catch (InvalidAccountException e) {
         e.printStackTrace();
      }
   }

   @Test
   public final void testSearchCustomerBySsn() {
      // Invoke a method
      try {
         Customer bankCustomer = this.bank
               .searchCustomerBySsn(ITSOBank4Test.CUSTOMER_SSN);
         // Verify an assertion
         assertEquals(bankCustomer.getSsn(), ITSOBank4Test.CUSTOMER_SSN);
      } catch (InvalidCustomerException e) {
         e.printStackTrace();
      }
```

```
    }
}
```

In the foregoing example, the differences are marked in **bold**. In JUnit, each test is implemented as a Java method that should be declared as `public void` and should take no parameters. This method is then invoked from a test runner. In previous JUnit releases, all the test method names had to begin with `test...`, so the test runner could find them automatically and run them. In JUnit 4.x, this is no longer required, because we mark the test methods with the `@Test` annotation.

Important: The package structure **`junit.framework`** used in JUnit 3.8.1 has been changed to **`org.junit`** in JUnit 4.x.

JUnit Assert class

JUnit provides a number of static methods in the `org.junit.Assert` class that can be used to assert conditions and fail a test if the condition is not met. Table 21-1 summarizes the provided static methods.

Table 21-1 JUnit Assert class: Static methods overview

Method name	Description
`assertEquals`	Asserts that two objects or primitives are equal. Compares objects using `equals` method, and compares primitives using `==` operator.
`assertFalse`	Asserts that a boolean condition is false.
`assertNotNull`	Asserts that an object is not null.
`assertNotSame`	Asserts that two objects do not refer the same object. Compares objects using `!= operator`.
`assertNull`	Asserts that an object is null.
`assertSame`	Asserts that two objects refer to the same object. Compares objects using `== operator`.
`assertTrue`	Asserts that a boolean condition is true.
`fail`	Fails the test.

All of these methods include an optional `String` parameter that allows the writer of a test to provide a brief explanation of why the test failed. This message is reported along with the failure when the test is executed. The full JUnit 4 API documentation can be found here:

http://junit.sourceforge.net/javadoc_40/index.html

Test suite class

Test cases can be organized into test suites. There are two ways how you can add a `TestCase` class to the test suite in JUnit 3.8.1:

- Pass a `TestCase` object as an argument to the constructor of the `TestSuite` class, as shown in Example 21-3. The `TestSuite` class can extract the tests to be run automatically.
- Add single tests with the `addTest` method of the `TestSuite` class, as shown in Example 21-4.

Example 21-3 Simple JUnit 3.8.1 test suite class: Using constructor

```
package itso.rad7.bank.test.junit;

import junit.framework.Test;
import junit.framework.TestSuite;

public class All381Tests_Example1 {

   public static Test suite() {
      // Automatically - runs all test...() methods
      TestSuite suite = new TestSuite(ITSOBank381Test.class,
            "All test cases of itso.rad7.bank.test.junit.ITSOBank381Test");
      return suite;
   }
}
```

Example 21-4 Simple JUnit 3.8.1 test suite class: Using addTest method

```
package itso.rad7.bank.test.junit;

import junit.framework.Test;
import junit.framework.TestSuite;

public class All381Tests_Example2 {

   public static Test suite() {
      TestSuite suite = new TestSuite();

      // Add manually single tests to the suite
      suite.addTest(new ITSOBank381Test("testSearchAccountByAccountNumber"));
      return suite;
   }
}
```

In Junit 4.x, the way to build test suites has been completely replaced and no longer uses subclassing, reflection, and naming conventions. Example 21-5 shows how to build a test suite class in JUnit 4.x.

Example 21-5 Simple JUnit 4.x test suite class

```
package itso.rad7.bank.test.junit;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ITSOBank4Test.class})
public class All4Tests { }
```

The `All4Tests` class is a simple placeholder for the `@RunWith` and `@SuiteClasses` annotations, and does not require a static `suite` method. The `@RunWith` annotation tells the JUnit 4 test runner to use the `org.junit.runners.Suite` class for running the `All4Tests` class. The `@SuiteClasses` annotation allows you to define which test classes to include in this suite and in which order. If you add more than one test class, the syntax is:

```
@SuiteClasses({TestClass1.class, TestClass2.class})
```

> **Note:** We are using JUnit 4.x in this chapter, because it is our goal to make you familiar with the new features and supported technologies of Application Developer V7.

Prepare the JUnit sample

We use the ITSO Bank application created in “Developing the ITSO Bank application” on page 236 for the JUnit test working example.

Import the `c:\7501code\junit\RAD7JUnit.zip` project interchange file with the **RAD7JUnit** project to your workspace.

The `RAD7JUnit` project is a copy of the `RAD7Java` project developed in Chapter 7, “Develop Java applications” on page 227, and we do not provide step-by-step guidance for topics that have been already described there. After importing the project, verify that it runs by executing the `BankClient` class.

The completed code for this section can also be imported from the `c:\7501code\zInterchangeFiles\junit\RAD7JUnit.zip` project interchange file.

Open the Java perspective.

Creating a JUnit test case

Application Developer provides wizards to help you build JUnit test cases and test suites. The following step-by-step guide leads you through the example, so that you get familiar with the JUnit tooling within Application Developer:

- Create a new package called `itso.rad7.bank.test.junit` in the `RAD7JUnit` project (under `src`).
- Add the JUnit library to the Java project so that the classes from the JUnit framework can be resolved:
  - Right-click the `RAD7JUnit` project and select **Properties**, or press **Alt+Enter**.
  - Select **Java Build Path** and select the **Libraries** tab. Click **Add Library**.
  - In the Add Library dialog, select **JUnit** and click **Next**.
  - Select **JUnit 4** in the JUnit library version field and click **Finish**.
  - Click **OK** to close the Properties dialog.

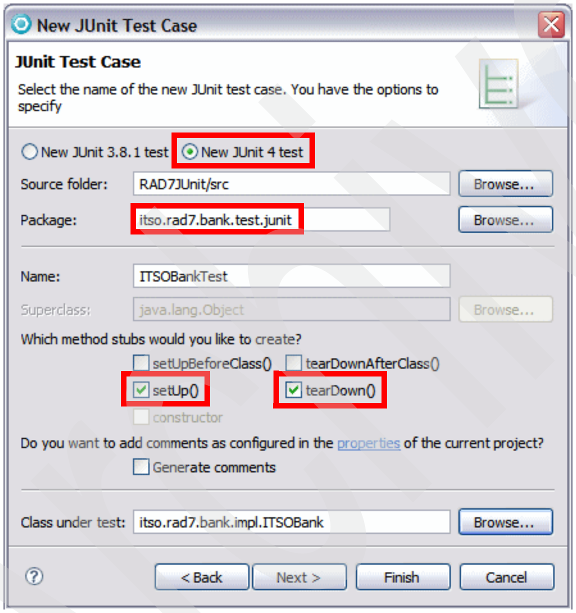
> **Note:** Instead of adding a library, you can add a variable. Classpath variables are an indirection to JARs with the benefit of avoiding local file system paths in a classpath. This is needed when projects are shared in a team. To add a variable to the `RAD7JUnit` project, do these steps:
>
> - Right-click the `RAD7JUnit` project and select **Properties**, or press **Alt+Enter**.
> - Select **Java Build Path** and select the **Libraries** tab. Click **Add Variable**.
> - In the New Variable Classpath Entry dialog, there is a predefined variable called `JUNIT_HOME`. This variable links to the JUnit 3.8.1 directory. But we want to use JUnit 4.x. Therefore, we have to configure the variable. Click **Configure Variables**.
> - Select the `JUNIT_HOME` variable and click **Edit**.
> - In the Edit Variable Entry dialog, click **Folder,** and navigate to the folder .../plugins/**org.junit4_4.1.0.1** and click **OK**.
>
>   Note that you could also create a new variable (`JUNIT4_HOME`) pointing to the JUnit4 folder (`../plugins/org.junit4_4.1.0.1`).
> - Click **OK** to close the dialog.
> - In the New Variable Classpath Entry dialog, select the `JUNIT_HOME` (or `JUNIT4_HOME`) variable and click **Extend**. Select the `junit-4.1.jar` file and click **OK**.
> - Click **OK** to close the Properties dialog.

- Import the examples shown in "What is new in JUnit 4.x?" on page 1008 to the package `itso.rad7.bank.test.junit`. The source files can be loaded from c:\7501code\junit\examples.

Create a JUnit test case

To create a test case for the `transfer` method of the `ITSOBank` class, do these steps:

- Right-click the `itso.rad7.bank.impl.ITSOBank` class and select **New** → **JUnit Test Case** (only available in the Java perspective), or select **New** → **Other** → **Java** → **JUnit** → **JUnit Test Case**, or click the arrow in the icon in the toolbar and select JUnit Test Case.
- In the JUnit Test Case dialog, enter the following data (Figure 21-1).



New JUnit Test Case

JUnit Test Case

Select the name of the new JUnit test case. You have the options to specify

○ New JUnit 3.8.1 test ● New JUnit 4 test

Source folder: RAD7JUnit/src [Browse...]

Package: itso.rad7.bank.test.junit [Browse...]

Name: ITSOBankTest

Superclass: java.lang.Object [Browse...]

Which method stubs would you like to create?

□ setUpBeforeClass() □ tearDownAfterClass()

☑ setUp() ☑ tearDown()

□ constructor

Do you want to add comments as configured in the properties of the current project?

□ Generate comments

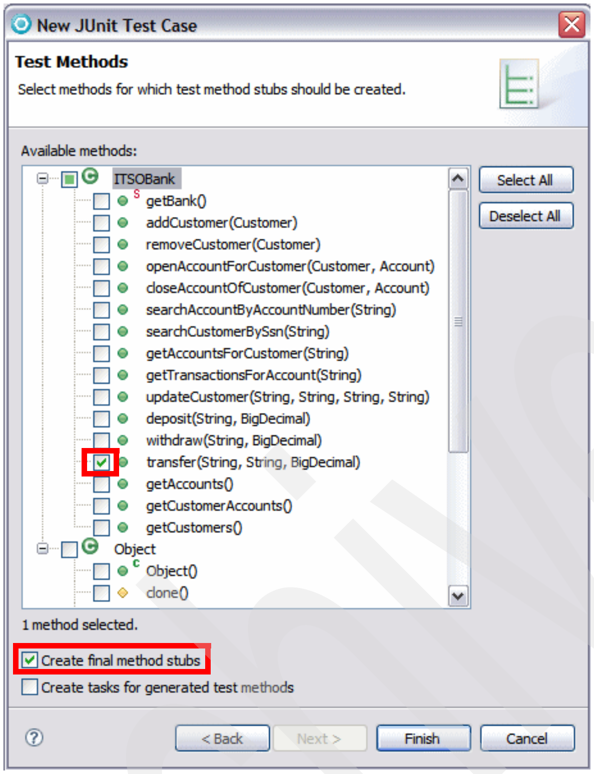
Class under test: itso.rad7.bank.impl.ITSOBank [Browse...]

Figure 21-1 New JUnit Test Case wizard

- Select **New JUnit 4 test**.
- For Package, enter **`itso.rad7.bank.test.junit`**.
- For Name, accept: `ITSOBankTest`
- Select **`setUp()`** and **`tearDown()`** methods.
- Clear Generate comments (default).
- Click **Next**.

> **Note:** A *stub* is a skeleton method so that you can add the body of the method yourself.

- In the Test Methods dialog, select the **transfer** method and **Create final method stubs** (Figure 21-2) and then click **Finish**.



New JUnit Test Case

Test Methods

Select methods for which test method stubs should be created.

Available methods:

- ■ ITSOBank
  - □ getBank()
  - □ addCustomer(Customer)
  - □ removeCustomer(Customer)
  - □ openAccountForCustomer(Customer, Account)
  - □ closeAccountOfCustomer(Customer, Account)
  - □ searchAccountByAccountNumber(String)
  - □ searchCustomerBySsn(String)
  - □ getAccountsForCustomer(String)
  - □ getTransactionsForAccount(String)
  - □ updateCustomer(String, String, String, String)
  - □ deposit(String, BigDecimal)
  - □ withdraw(String, BigDecimal)
  - ☑ transfer(String, String, BigDecimal)
  - □ getAccounts()
  - □ getCustomerAccounts()
  - □ getCustomers()
- □ Object
  - □ Object()
  - □ clone()

1 method selected.

☑ Create final method stubs

□ Create tasks for generated test methods

Figure 21-2 Select test methods

The wizard generates the `ITSOBankTest.java` and opens the file in the Java editor.

Complete the test class

Typically, you run several tests in one test case. To make sure there are no side effects between test runs, the JUnit framework provides the `setUp` and `tearDown` methods. Every time the test case is run, `setUp` is called at the start and `tearDown` is called at the end of the run.

- Add three variables to `ITSOBankTest` class:

```
private Bank bank = null;
private static final String ACCOUNT_NUMBER_1 = "001-999000777";
private static final String ACCOUNT_NUMBER_2 = "002-999000777";
```

Remember that you can add missing imports by selecting **Source** → **Organize Imports**, or by pressing **Ctrl+Shift+O**.

The bank variable is instantiated in the setUp method before starting each test and is available for use in the test methods.

- Add the code to the setUp method:

```
@Before
public void setUp() throws Exception {
   /* Instantiate objects
    * The getBank method returns the initialized (containing Customers
    * and Accounts) ITSOBank instance.
    */
   this.bank = ITSOBank.getBank();
}
```

- Keep the generated code of the tearDown method unchanged.

Note: The JUnit framework calls the setUp method before each test method. The ITSOBank class is implemented as a *singleton* (only one object of this class exists). Therefore, you get always the same ITSOBank object, when you call the static getBank method. When the setUp method gets called a second time, you would get the same ITSOBank instance as in the first call.

For example, if you have removed all the customers in the first test method, the ITSOBank object that you get in the second call of the setUp method would be empty, because removing a customer from the bank closes automatically all of his accounts. Therefore, it can be useful to call a kind of clean up service in the tearDown method to reset the ITSOBank instance. In our example this is not needed.

Complete the test methods

When the ITSOBankTest is generated, the stub of the testTransfer method is added. This section describes the steps to implement this test method and add a second method.

- Complete the testTransfer method (Example 21-6).

Example 21-6   ITSOBankTest class: testTransfer method

```
@Test
public final void testTransfer() {
   try {
      BigDecimal account1AmountBeforeTransfer = this.bank
            .searchAccountByAccountNumber(
                  ITSOBankTest.ACCOUNT_NUMBER_1).getBalance();
      BigDecimal account2AmountBeforeTransfer = this.bank
            .searchAccountByAccountNumber(
                  ITSOBankTest.ACCOUNT_NUMBER_2).getBalance();
      BigDecimal transferAmount = new BigDecimal(20.00D);
      // Invoke a method
      this.bank.transfer(ITSOBankTest.ACCOUNT_NUMBER_1,
            ITSOBankTest.ACCOUNT_NUMBER_2, transferAmount);

      // Verify assertions
      Assert.assertEquals(this.bank.searchAccountByAccountNumber(
            ITSOBankTest.ACCOUNT_NUMBER_1).getBalance().doubleValue(),
            account1AmountBeforeTransfer.subtract(transferAmount)
                  .doubleValue(), 0.00D);
      Assert.assertEquals(this.bank.searchAccountByAccountNumber(
            ITSOBankTest.ACCOUNT_NUMBER_2).getBalance().doubleValue(),
            account2AmountBeforeTransfer.add(transferAmount)
                  .doubleValue(), 0.00D);
   } catch (ITSOBankException e) {
      e.printStackTrace();
      Assert.fail("Transfer failed: " + e.getMessage());
   }
}
```

Note: Make sure you import java.math.BigDecimal and org.junit.Assert.

This test method transfers an amount from one account to another one. After the credit and debit transactions have completed, the method verifies that the balances of the two involved accounts have changed accordingly.

- If you really want to test the method completely, you have to write a test method for each possible outcome of that method. In our example we just add another test method called testInvalidTransfer, as shown in Example 21-7. This method also calls the transfer method, but this time we make a transfer where the debit account does not have enough funds. The methods verifies that you receive an InvalidTransactionException.

Example 21-7 ITSOBankTest class: testInvalidTransfer

```
@Test
public final void testInvalidTransfer() {
   try {
      BigDecimal transferAmount =
         this.bank.searchAccountByAccountNumber(
            ITSOBankTest.ACCOUNT_NUMBER_1).getBalance().multiply(
            new BigDecimal(2.00D));

      // Invoke a method
      this.bank.transfer(ITSOBankTest.ACCOUNT_NUMBER_1,
            ITSOBankTest.ACCOUNT_NUMBER_2, transferAmount);
   } catch (InvalidAccountException e) {
      e.printStackTrace();
      Assert.fail("Transfer failed: " + e.getMessage());
   } catch (InvalidTransactionException e) {
      e.printStackTrace();
      Assert.assertTrue(true);
   }
}
```

Creating a JUnit test suite

A JUnit test suite is used to run one or more test cases at once. Application Developer contains a simple wizard to create a test suite for JUnit 3.8.1 test cases.

> **Important:** We found that the New JUnit Test Suite wizard does not allow us to select between JUnit 3.8.1 and JUnit 4.x test suites. The wizard can only be used for JUnit 3.8.1 test suites and therefore, it only lets you add JUnit 3.8.1 or lower JUnit version test cases. This is a known Eclipse bug:
>
> `http://bugs.eclipse.org/bugs/show_bug.cgi?id=155828`
>
> We had to create the JUnit test suite for the example manually. We provide below a step-by-step guide to create a test suite for JUnit 3.8.1 test cases. This guide is just given for completeness and cannot be used for the example.

To create a JUnit 4.x test suite manually, do these steps:

- Create a new class called `AllTests` in the same package. The class extends the default superclass `java.lang.Object`, and does not require any interfaces or method stubs.
- Add the import statements and annotations to the `AllTests` class, as shown in Example 21-8. The structure of that class is described in “Test suite class” on page 1012.

Example 21-8 AllTests class: A JUnit 4.x test suite class

```
package itso.rad7.bank.test.junit;

import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ITSOBankTest.class})
public class AllTests {}
```

In our example we only have added a single test class, and thus a test suite is not required. However, as you add more and more test cases, a test suite quickly becomes a more practical way to manage your unit testing.

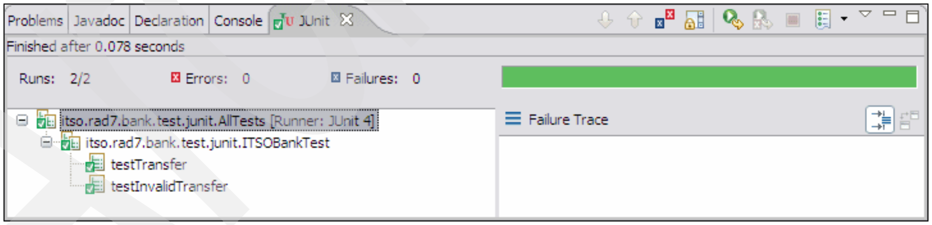
To create a JUnit 3.8.1 test suite using the New JUnit Test Suite wizard, do these steps:

- Right-click the junit package and select **New** → **Other** → **Java** → **JUnit** → **JUnit Test Suite**.
- In the JUnit Test Suite dialog, enter the following data:
  - Name: `All3Tests`
  - Test classes to include in suite: select all test classes which you want to include in this suite (for example, `ITSOBank381Test`).
  - Click **Finish**.

Running the JUnit test case or JUnit test suite

To run a JUnit test case or JUnit test suite, do these steps:

- Select the `ITSOBankTest` or `AllTests` class and click the arrow of ▶ ▾ in the toolbar and select **Run As** → **JUnit Test** (or right-click the class and select **Run As** → **JUnit Test**.
- In our example, Application Developer runs the two test methods defined in `ITSOBankTest` class. Do not worry about the `InvalidTransactionException` that gets thrown and displayed in the Console view, because it is expected in the `testInvalidTransfer` method.
- The JUnit view opens. You might want to move the JUnit view on top of the Console view.
- Notice that the two test methods passed the asserts verification (Figure 21-3).



Finished after 0.078 seconds

Runs: 2/2 Errors: 0 Failures: 0

```
itso.rad7.bank.test.junit.AllTests [Runner: JUnit 4]
  itso.rad7.bank.test.junit.ITSOBankTest
    testTransfer
    testInvalidTransfer
```

Figure 21-3 JUnit view: Both test methods passed the assert verifications

Tip: To run the same test again, click ▶ in the JUnit view toolbar.

Modify and run the JUnit test case with assert failures

In our example we test only for success (even one method throws an exception, but that was what we had expected). A test is considered to be *successful*, if the test method returns normally. A test *fails*, if one of the methods could not pass all assert verifications. An *error* indicates that an unexpected exception is raised by any test, setUp, or tearDown method. The JUnit view is more interesting when an error or failure occurs.

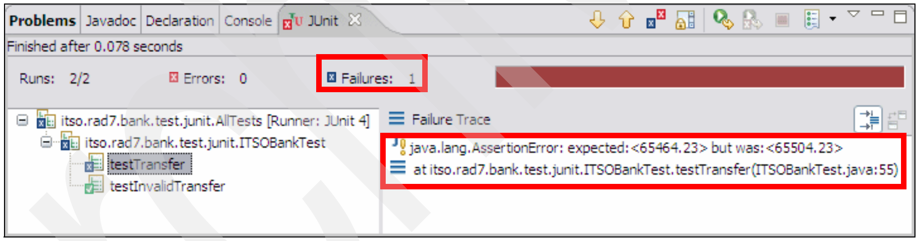
- Modify the process method of the itso.rad7.bank.model.Credit class:

```
public BigDecimal process(BigDecimal accountBalance)
        throws InvalidTransactionException {
   if ((this.getAmount() != null)
        && (this.getAmount().compareTo(new BigDecimal(0.00D)) > 0)) {
      return accountBalance.subtract(this.getAmount());
   } else {
      ...... // rest unchanged
   }
```

  We change **add** to **subtract**, which is obviously an error in the business logic.

- Run the ITSOBankTest class again as a JUnit Test.

  This time, a failure and its trace information is displayed for the testTransfer method in the JUnit view (Figure 21-4).



Finished after 0.078 seconds

Runs: 2/2 Errors: 0 Failures: 1

```
itso.rad7.bank.test.junit.AllTests [Runner: JUnit 4]
   itso.rad7.bank.test.junit.ITSOBankTest
      testTransfer
      testInvalidTransfer
```

Failure Trace

```
java.lang.AssertionError: expected:<65464.23> but was:<65504.23>
at itso.rad7.bank.test.junit.ITSOBankTest.testTransfer(ITSOBankTest.java:55)
```

Figure 21-4 JUnit view with failure

Double-clicking the entry in the Failure Trace list takes you to the specified line in the specified Java source file. This is the line where to set a breakpoint and start debugging the application. For details how to debug an application, refer to Chapter 22, “Debug local and remote applications” on page 1041.

- Correct the process method of the Credit class, by undoing the change we made before.

JUnit testing using TPTP

TPTP JUnit test generates an execution history from which a report can be generated. In this section we discuss the following topics:

- Creating the TPTP JUnit sample
- Importing an existing JUnit test case
- Running the TPTP JUnit test
- Analyzing the test results
- Generating test reports

Note: TPTP JUnit Tests are based on JUnit version 3.8.1.

Creating the TPTP JUnit sample

In this section we continue with the `RAD7JUnit` project that we created in “JUnit testing [without using TPTP]” on page 1007.

Note: If you imported the final `RAD7JUnit` project from the sample code and you want to create this example on your own, delete the `itso.rad7.bank.test.tptp` package.

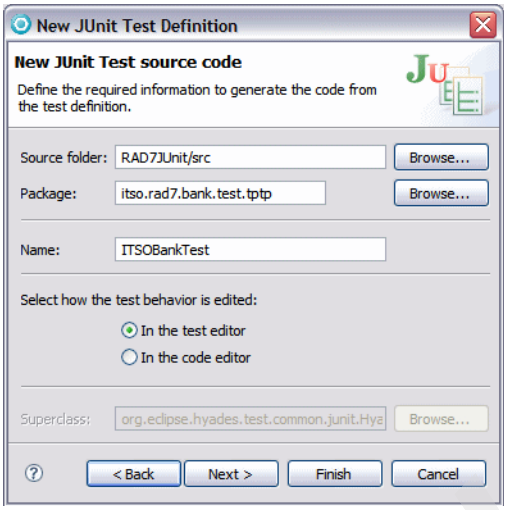
Create new package

Add a new package called `itso.rad7.bank.test.tptp` to the `RAD7JUnit` project.

Create a TPTP JUnit test manually

To create a TPTP JUnit test manually, do these steps:

- Right-click the `itso.rad7.bank.test.tptp` package and select **New** → **Other** → **Test** → **TPTP JUnit Test** (select **Show All Wizards**).
- In the Confirm Enablement dialog, click **OK** to enable the Core Testing capability of Application Developer.
- Click **Yes** to allow Application Developer to add the libraries `common.runner.jar` and `java.runner.jar` to the build path.
- In the New JUnit Test Definition dialog, enter the following data (Figure 21-5):
  - Source folder: `RAD7JUnit/src` {default}
  - Package: `itso.rad7.bank.test.tptp` {default}
  - Name: **ITSOBankTest**
  - Select **In the test editor** {default}



New JUnit Test Definition

New JUnit Test source code

Define the required information to generate the code from the test definition.

Source folder: RAD7JUnit/src

Package: itso.rad7.bank.test.tptp

Name: ITSOBankTest

Select how the test behavior is edited:

● In the test editor

○ In the code editor

Superclass: org.eclipse.hyades.test.common.junit.Hya

Figure 21-5 New JUnit Test source code dialog

- In the JUnit Test Definition dialog, enter the following data and click **Finish**.
  - Enter or select the parent folder:

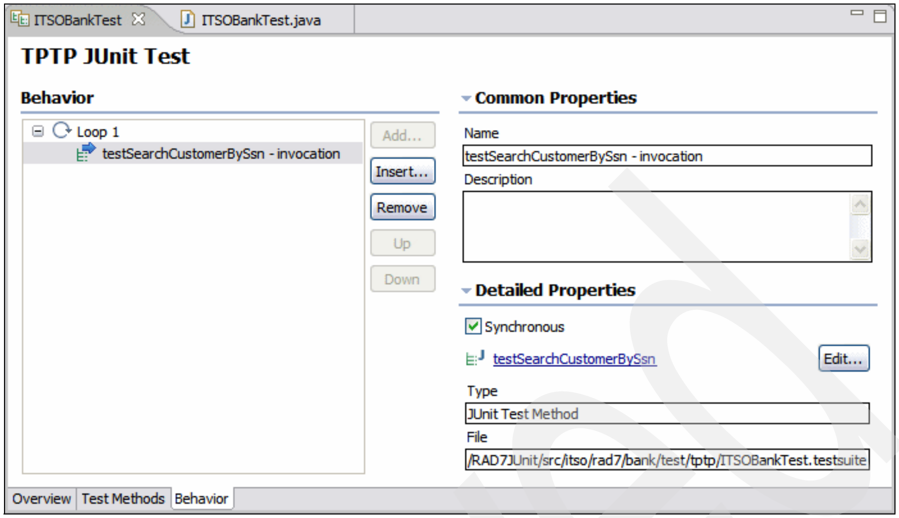
    `RAD7JUnit/src/itso/rad7/bank/test/tptp`

  - Name: `ITSOBankTest`

> **Note:** In the previous step we enter the name and location of the source code, while in this step we enter the name and location of the TPTP Test (the model). By default, they are identical.

- The JUnit Test Suite editor opens and you can create and remove methods on a JUnit test, and control how those methods are invoked. Three tabs are visible: Overview, Test Methods, and Behavior.
- In the **Test Methods** tab, click **Add** and enter **testSearchCustomerBySsn** in the name field.
- In the **Behavior** tab, click **Add** and select **Loop**. Select the **Loop 1**, click **Add** again, select **invocation,** select the `testSearchCustomerBySsn` method and click **OK**.

  At this point the Behavior tab looks as shown in Figure 21-6.



TPTP JUnit Test

Behavior

- Loop 1
  - testSearchCustomerBySsn - invocation

Common Properties

Name: testSearchCustomerBySsn - invocation

Description:

Detailed Properties

☑ Synchronous

testSearchCustomerBySsn

Type: JUnit Test Method

File: /RAD7JUnit/src/itso/rad7/bank/test/tptp/ITSOBankTest.testsuite

Figure 21-6 TPTP JUnit Test editor [Behavior tab]

> **Note:** In the Overview tab:
>
> - If **Implement Test Behavior as code** is **selected**, the behavior is purely code-based, that is, the test methods are executed exactly as presented in the Test Methods view.
> - If **Implement Test Behavior as code** is **cleared**, then the Behavior tab becomes available. The behavior feature should be used only for TPTP JUnit tests that have been created manually.

- Save the TPTP JUnit Test editor.
- Open the generated `itso.rad7.bank.test.tptp.ITSOBankTest` class in the Java editor and add the highlighted code, as shown in Example 21-9.

Example 21-9 ITSOBankTest class

```
package itso.rad7.bank.test.tptp;

import junit.framework.Test;
import .....;

import itso.rad7.bank.exception.InvalidCustomerException;
import itso.rad7.bank.ifc.Bank;
import itso.rad7.bank.impl.ITSOBank;
import itso.rad7.bank.model.Customer;

/**
```

```
 * Generated code for the test suite <b>ITSOBankTest</b> located at
 * <i>/RAD7JUnit/src/itso/rad7/bank/test/tptp/ITSOBankTest.testsuite</i>.
 */
public class ITSOBankTest extends HyadesTestCase {

   private Bank bank = null;
   private static final String CUSTOMER_SSN = "111-11-1111";

   /**
    * Constructor for ITSOBankTest.
    * @param name
    */
   public ITSOBankTest(String name) {
      super(name);
   }

   /**
    * Returns the JUnit test suite that implements the <b>ITSOBankTest</b>
    * definition.
    */
   public static Test suite() {
      HyadesTestSuite iTSOBankTest = new HyadesTestSuite("ITSOBankTest");
      iTSOBankTest.setArbiter(DefaultTestArbiter.INSTANCE).setId(
            "DDB9E1EA6622C91ECCFC7050406611DC");

      HyadesTestSuite loop1 = new HyadesTestSuite("Loop 1");
      iTSOBankTest.addTest(new RepeatedTest(loop1, 1));
      loop1.setId("DDB9E1EA6622C91E00B65230406711DC");

      loop1.addTest(new ITSOBankTest("testSearchCustomerBySsn").setId(
            "DDB9E1EA6622C91EEFA8C810406611DC").setTestInvocationId(
            "DDB9E1EA6622C91E109E6F20406711DC"));
      return iTSOBankTest;
   }

   /**
    * @see junit.framework.TestCase#setUp()
    */
   protected void setUp() throws Exception {
      this.bank = ITSOBank.getBank();
   }

   /**
    * @see junit.framework.TestCase#tearDown()
    */
   protected void tearDown() throws Exception {}

   /**
    * testSearchCustomerBySsn
    * @throws Exception
    */
   public void testSearchCustomerBySsn() throws Exception {
      // Enter your code here
      try {
         Customer bankCustomer = this.bank
               .searchCustomerBySsn(ITSOBankTest.CUSTOMER_SSN);
         assertEquals(bankCustomer.getSsn(), ITSOBankTest.CUSTOMER_SSN);
      } catch (InvalidCustomerException e) {
         e.printStackTrace();
      }
   }
}
```

Note: The long hexadecimal IDs might be different in your workspace.

Importing an existing JUnit test case

To create a TPTP JUnit Test by importing an existing JUnit test case, do these steps:

- Right-click in the Package Explorer, select **Import** → **Test** → **JUnit tests to TPTP**, and click **Next**.
- In the Import JUnit tests to TPTP dialog, select **RAD7JUnit** → **itso.rad7.bank.test.junit** → **ITSOBank381Test.java**, and click **Finish**.

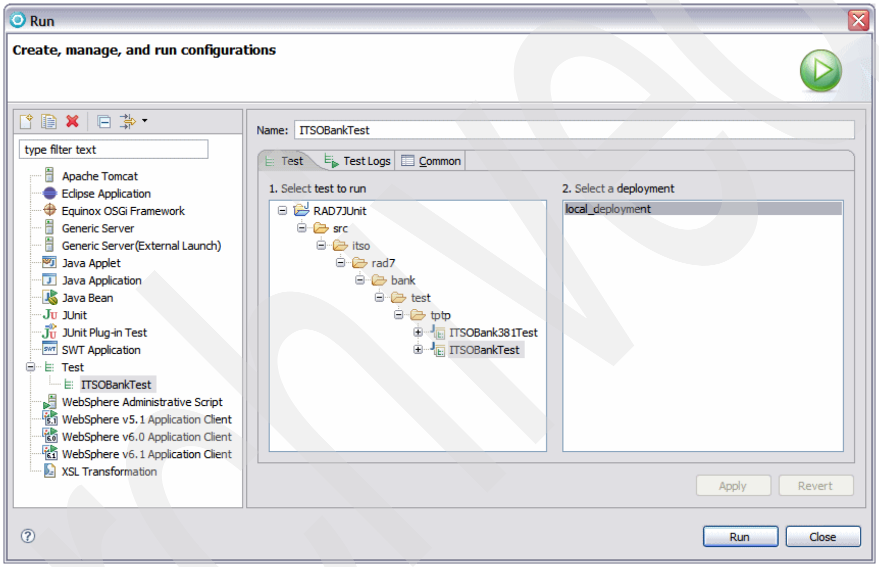
> **Note:** Notice that the Import JUnit test to TPTP dialog does not allow us to select the `ITSOBankTest.java` or `ITSOBankTest.java` test cases that are also in the `itso.rad7.bank.test.junit` package. The reason is that those test cases are JUnit 4.x based and TPTP cannot handle JUnit 4.x.

- Copy the `ITSOBank381Test` class and move the newly generated `ITSOBank381Test.testsuite` from the `itso.rad7.bank.test.junit` package to the package `itso.rad7.bank.test.tptp` by using Application Developer's refactor tooling.
- Open the `ITSOBank381Test.testsuite` in the TPTP JUnit Test editor. In the Overview tab change the package name from `itso.rad7.bank.test.junit` to `itso.rad7.bank.test.tptp`.
- Save and close the editor.

Running the TPTP JUnit test

To run a TPTP JUnit test, do these steps:

- Right-click **ITSOBankTest.testsuite** and select **Run As** → **Run**.
- In the Create, manage, and run configurations dialog, right-click **Test** and select **New**. Enter the following data (Figure 21-7):
  - Name: `ITSOBankTest`
  - Select test to run: `ITSOBankTest` (preselected)
  - Select a deployment: `local_deployment` {default}



Name: ITSOBankTest

1. Select test to run: ITSOBankTest

2. Select a deployment: local_deployment

Figure 21-7 TPTP JUnit Test Run Configuration dialog

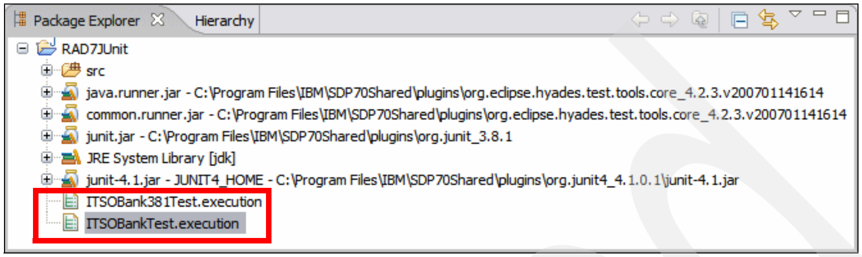
- Click **Apply** and then **Run**.

There is a faster way to run a test suite with default values:

- Right-click **ITSOBank381Test.testsuite** and **Run As** → **Test**. This creates the configuration automatically and runs the test.
- You can verify the configuration by selecting **Run** → **Run**. A second configuration named **ITSBank381Test** has be created under Test.

Analyzing the test results

When the test run is finished, the execution result is generated in the `RAD7JUnit` project (Figure 21-8).

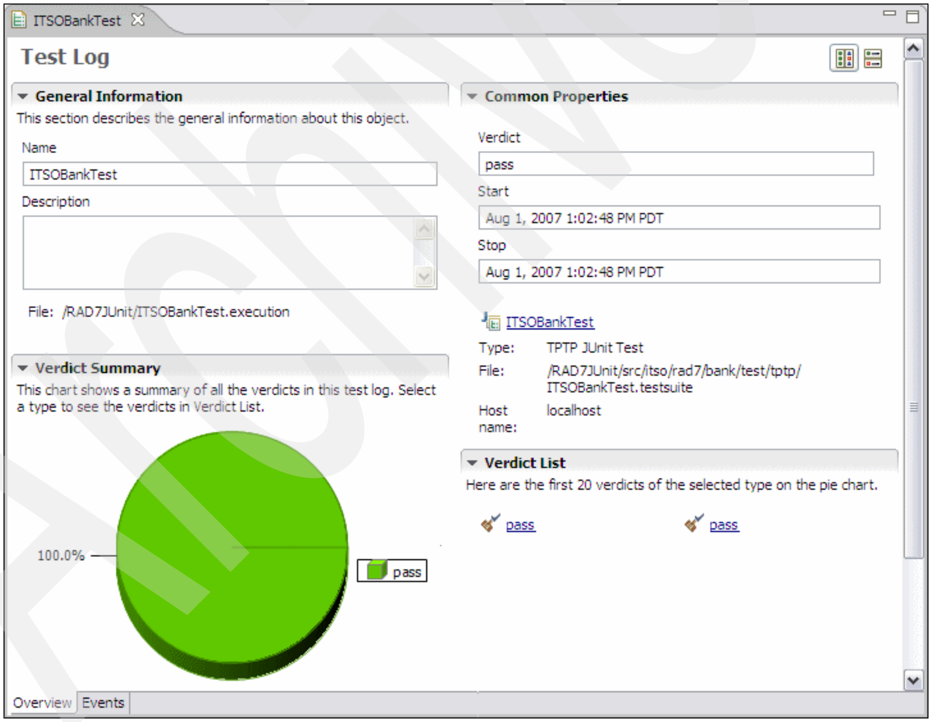


Package Explorer | Hierarchy

- RAD7JUnit
  - src
  - java.runner.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.eclipse.hyades.test.tools.core_4.2.3.v200701141614
  - common.runner.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.eclipse.hyades.test.tools.core_4.2.3.v200701141614
  - junit.jar - C:\Program Files\IBM\SDP70Shared\plugins\org.junit_3.8.1
  - JRE System Library [jdk]
  - junit-4.1.jar - JUNIT4_HOME - C:\Program Files\IBM\SDP70Shared\plugins\org.junit4_4.1.0.1\junit-4.1.jar
  - ITSOBank381Test.execution
  - ITSOBankTest.execution

Figure 21-8 Package Explorer view containing test execution results

To analyze the test results, double-click the test execution result (Figure 21-9).



ITSOBankTest

Test Log

General Information

This section describes the general information about this object.

Name: ITSOBankTest

Description:

File: /RAD7JUnit/ITSOBankTest.execution

Verdict Summary

This chart shows a summary of all the verdicts in this test log. Select a type to see the verdicts in Verdict List.

Common Properties

Verdict: pass

Start: Aug 1, 2007 1:02:48 PM PDT

Stop: Aug 1, 2007 1:02:48 PM PDT

ITSOBankTest

Type: TPTP JUnit Test

File: /RAD7JUnit/src/itso/rad7/bank/test/tptp/ITSOBankTest.testsuite

Host name: localhost

Verdict List

Here are the first 20 verdicts of the selected type on the pie chart.

pass pass

Figure 21-9 TPTP JUnit test execution result

- In Test Log Overview tab, you can see the test verdict (**pass**) and the starting and stopping time of the test run.
- The Test Log Events tab lists each single step of the test run with further information about it.

Generating test reports

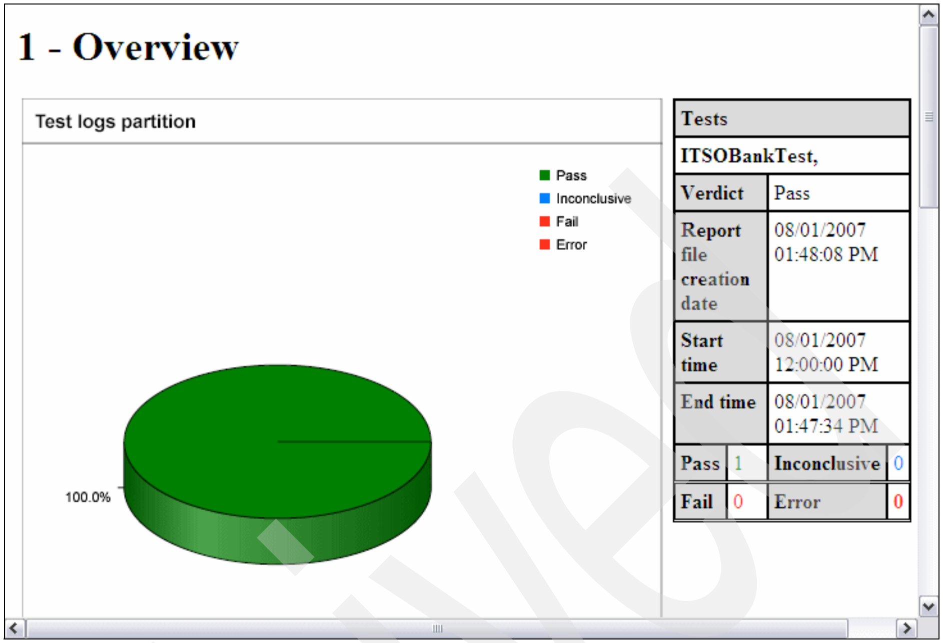
Based on a test execution results file, you can generate analysis reports:

- **Test Pass report**: Summary of the latest test execution result with graphical presentation of test success.
- **Time Frame Historic report**: Summary of all test execution result within a time frame with graphical presentation of the test success.

> **Note:** To view the reports, you have to install the Scalable Vector Graphics (SVG) browser plug-in. You can get this free viewer from the Adobe® Web site:
>
> `http://www.adobe.com/svg/viewer/install/main.html`

To generate a Test Pass Report, do these steps:

- Open the Test perspective.
- Expand **RAD7JUnit** → **src** → **itso** → **rad7** → **bank** → **test** → **tptp**.
- In the Test Navigator, right-click **`ITSOBankTest`** and select **Report**.
- In the New Report dialog, select **Test Pass Report** and **Open Report Automatically**, then click **Next**.
- In the Test Pass Report dialog, enter **ITSOBankTest_TestPass** as name (the folder is preselected), and click **Next**.
- In the Select a time frame dialog, enter the start and end date and time:
  - The end time is set as the current time.
  - For the start time, enter a time before you started testing.
  - Click **Finish** and a Test Pass report is generated.
- The report file `ITSOBankTest_TestPass` appears in the `tptp` folder and the report is displayed (Figure 21-10).
- If you did not select **Open Report Automatically**, then right-click the **tptp** folder (where the `ITSOBankTest` is located) and select **Refresh**. The generated file `ITSOBankTest_TestPass` is now visible. Right-click **ITSOBankTest_TestPass** and select **Open With** → **Web Browser**.

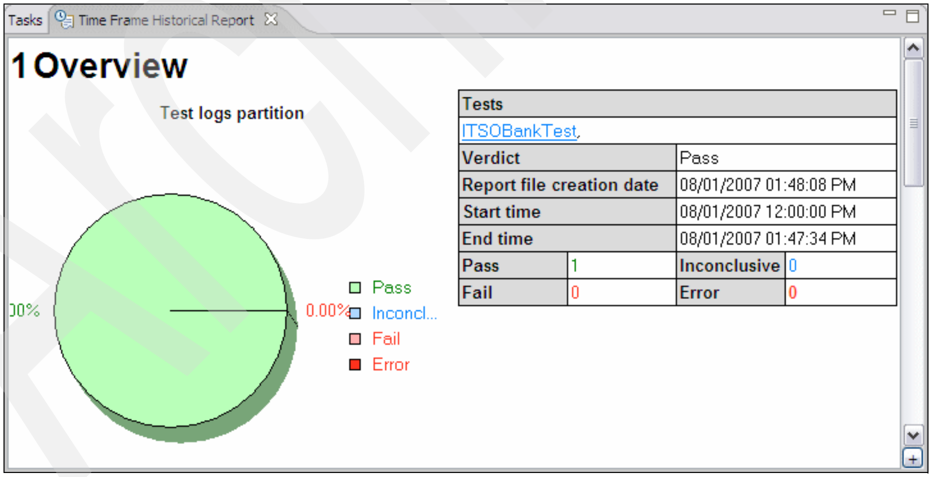


1 - Overview

Tests			
ITSOBankTest,			
Verdict	Pass		
Report file creation date	08/01/2007 01:48:08 PM		
Start time	08/01/2007 12:00:00 PM		
End time	08/01/2007 01:47:34 PM		
Pass	1	Inconclusive	0
Fail	0	Error	0

Figure 21-10 Test Pass report

The Time Frame Historic report is also open (Figure 21-11).



1 Overview

Tests			
ITSOBankTest,			
Verdict		Pass	
Report file creation date		08/01/2007 01:48:08 PM	
Start time		08/01/2007 12:00:00 PM	
End time		08/01/2007 01:47:34 PM	
Pass	1	Inconclusive	0
Fail	0	Error	0

Figure 21-11 Time Frame Historic report

Web application testing

You can also create test cases that run against one of the Web projects, `RAD7BankBasicWeb` or `RAD7StrutsWeb`. However, when testing anything that runs inside a servlet container, a testing framework like *Cactus* could make the testing much easier.

> **Note:** Cactus is an open source sub-project in the Apache Software Foundation's Jakarta Project. It is a simple framework for unit testing server-side Java code, such as servlets, EJBs, tag libraries, and filters.
>
> The objective of Cactus is to lower the cost of writing tests for server-side code. Cactus supports so-called white box testing of server-side code. It extends and uses JUnit.

In addition to providing a common framework for test tools and support for JUnit test generation, TPTP includes features allowing you to test Web applications.

TPTP provides the following Web testing tasks:

- Recording a test—The test creation wizard starts the Hyades proxy recorder, which records your interactions with a browser-based application. When you stop recording, the wizard starts a test generator, which creates a test from the recorded session.
- Editing a test—You can inspect and modify a test prior to compiling and running it.
- Generating an executable test—Follow this procedure to generate an executable test. Before a test can be run, the Java source code of the test must be generated and compiled. This process is called code generation.
- Running a test—Run the generated test.
- Analyzing test results—All the conclusion of a test run you see an execution history, including a test verdict, and you can request two graphical reports showing a page response time and a page hit analysis.

Preparing for the sample

As a prerequisite to the Web application testing sample, you must have the WebSphere Application Server V6.1 test environment installed and running. We use the `RAD7BankBasicWeb` application created in Chapter 12, “Develop Web applications using JSPs and servlets” on page 465, for the Web application testing sample.

If you do not have the RAD7BankBasicWeb application in the workspace, import the c:\7501code\zInterchangeFiles\webApps\RAD7BankBasic.zip project interchange file into the workspace (select all three projects).

The completed code for this section can also be imported from the C:\7501code\zInterchangeFiles\junit\RAD7JUnitWebTest.zip project interchange file.

To the verify that the Web application runs, do these steps:

- Switch to the Web perspective.
- Right-click the RAD7BankBasicWeb project in the Project Explorer and select **Run As** → **Run on Server**.
- Verify that the Web browser starts and the welcome page of ITSO RedBank is shown. Close the page.

Create a Java project

Create a new Java project called **RAD7JUnitWebTest**.

Recording a test

To create a simple HTTP test, do these steps:

> **Note:** To make sure your recording accurately captures HTTP traffic, clear the browser cache.

- Open the Test perspective.
- Right-click RAD7JUnitWebTest in the Test Navigator view, select **New** → **Test Element** → **Recording** → **HTTP Recording**, and click **Next**.
- In the HTTP Recording dialog, select the RAD7JUnitWebTest project and enter RAD7BankBasicWebTest.rec in the Recording file name field, and then click **Finish**.

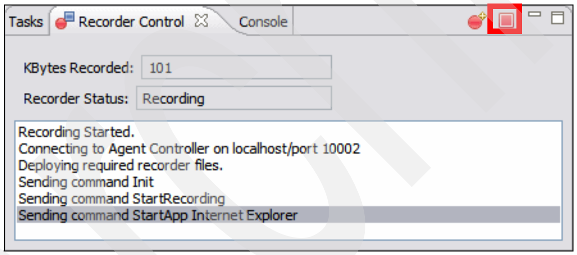
  A progress dialog box opens while your browser starts. Your browser settings are updated and a local proxy is enabled. If you are using a browser other than Microsoft Internet Explorer, see the online help for detailed instructions on how to configure the proxy.

> **Tip:** The browser used for recording can be set in **Window** → **Preferences** → **Test** → **HTTP Recording**.

- Recording has now started.
- Start the RAD7BankBasicWeb application by entering the following URL in the browser:

  ```
  http://localhost:9080/RAD7BankBasicWeb/
  ```

- We record a money transfer from a customer's account to another one, and verify that the required transactions have been created.
  - Select the **redbank** link on the ITSO RedBank welcome page, enter 111-11-1111 as customer ID (SSN), and click **Submit**.
  - Click account number 001-999000777 and on the next page select **Transfer**, enter 500 in the Amount field and 001-999000888 in the To account field, and click **Submit**.
  - Verify that **List transactions** is selected and click **Submit**. One DEBIT transaction is listed.
  - Click **Account Details** and then click **Customer Details**.
  - Click account number 001-999000888, verify that **List transactions** is selected and click **Submit**. One CREDIT transaction is listed.
  - Click **Account Details,** click **Customer Details**, and finally click **Logout**.
- Close the browser to stop recording, or click **Stop Recording** in the toolbar of the Recorder Control view (Figure 21-12).



KBytes Recorded: 101

Recorder Status: Recording

```
Recording Started.
Connecting to Agent Controller on localhost/port 10002
Deploying required recorder files.
Sending command Init
Sending command StartRecording
Sending command StartApp Internet Explorer
```

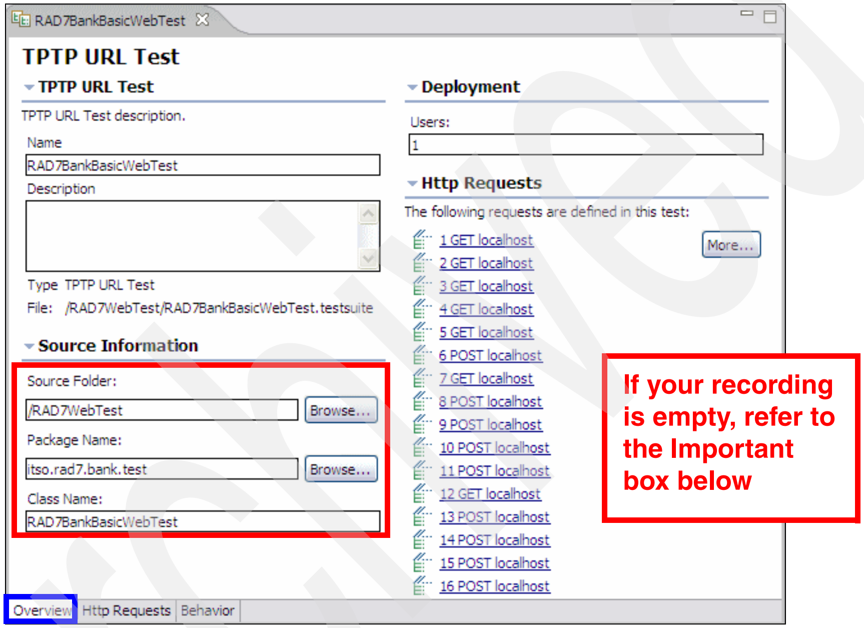
Figure 21-12 Recorder Control view

- Notice the messages Recording completed in the Recorder Control view after closing the browser.
- The wizard generates automatically a TPTP URL Test. The Recorder Control view informs you with the message Test generation completed when the test is successfully generated.

Editing the test

The TPTP URL Test appears under the `RAD7JUnitWebTest` project and is open in the editor. We can inspect and modify it before compiling and running it. The test is not Java code yet, but we can check the requests and modify them.

- Enter the following data in the **Overview** tab (Figure 21-13):
  - Source Folder: `/RAD7JUnitWebTest`
  - Package Name: `itso.rad7.bank.test`
  - Class Name: `RAD7BankBasicWebTest` {default}



TPTP URL Test

TPTP URL Test description.

Name: RAD7BankBasicWebTest

Description:

Type TPTP URL Test

File: /RAD7WebTest/RAD7BankBasicWebTest.testsuite

Source Information

Source Folder: /RAD7WebTest

Package Name: itso.rad7.bank.test

Class Name: RAD7BankBasicWebTest

Deployment

Users: 1

Http Requests

The following requests are defined in this test:

1 GET localhost
2 GET localhost
3 GET localhost
4 GET localhost
5 GET localhost
6 POST localhost
7 GET localhost
8 POST localhost
9 POST localhost
10 POST localhost
11 POST localhost
12 GET localhost
13 POST localhost
14 POST localhost
15 POST localhost
16 POST localhost

If your recording is empty, refer to the Important box below

Figure 21-13 TPTP URL Test dialog: Overview tab

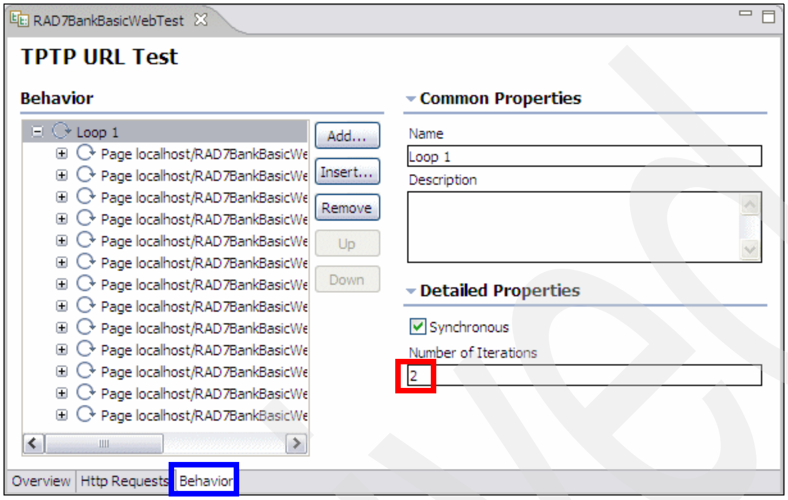
Important: For Internet Explorer 7.0 you must use the IP-address to run the Web application: `http://`**`<ip-address>`**`:9080/RAD7BankBasicWeb/`. No recording is produced when using `http://localhost:9080/`...

For Firefox 2.0, you must first configure the network settings: Select **Tools** → **Options** → **Advanced** → **Network**. Click **Settings** and select **Auto-detect proxy settings for the network**.

You can import the recording files into the `RAD7JUnitWebTest` project from:

```
C:\7501code\junit\webtestrecording
```

- Select the **Behavior** tab of the TPTP URL Test editor.
- Change the behavior of the test. For example, we adjust the number of iterations (Figure 21-14). Save the Test Editor.


RAD7BankBasicWebTest
TPTP URL Test
Behavior
Loop 1
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Page localhost/RAD7BankBasicWe
Add...
Insert...
Remove
Up
Down
Common Properties
Description
Detailed Properties
Overview
Http Requests
Behavior

Name: Loop 1

☑ Synchronous

Number of Iterations: 2

Figure 21-14 TPTP URL Test dialog: Behavior tab

Generating an executable test

Before a test can be run, the Java source code for the test must be generated and compiled. This process is called *code generation*. The compiled code is stored in the RAD7JUnitWebTest project.

To start the code generation of the RAD7BankBasicWeb, do these steps:

- Right-click **RAD7BankBasicWebTest** in the RAD7JUnitWebTest project and select **Generate**.
- In the TPTP URL Test Definition Code Generation dialog, select the RAD7JUnitWebTest project and click **Finish** to start the code generation.
- To examine the generated Java code, switch to the Web (or Java) perspective and open the itso.rad7.bank.test.RAD7BankBasicWebTest class.

Running the test

To run the RAD7BankBasicWebTest, do these steps:

- Switch to Test perspective.

- Right-click **RAD7BankBasicWebTest** in the RAD7JUnitWebTest project and select **Run As** → **Run**.
- In the Create, manage, and run configurations dialog, right-click **Test** and select **New**. Enter the following data (similar to Figure 21-6 on page 1024):
  - Name: RAD7BankBasicWebTest
  - Select test to run: Select RAD7JUnitWebTest → RAD7BankBasicWebTest
  - Select a deployment: local_deployment {default}
- Click **Apply** and then **Run**.

Important: We found a problem in the generated test class. At some point the RAD7BankBasicWeb application is using HttpSession to store the customer id. The session ID is normally stored as a transient cookie in the browser. Because the generated RAD7BankBasicWebTest class is acting as a browser, it also has to care about the session ID. But as you can see in the code, the session ID being used is hard coded. This concept does not work because the server gives the session ID after the first request from the client. Therefore, there is a mismatch between the session IDs of the client and the server and that is the reason the server is not able to find the customer id in the session. An InvalidCustomerException is thrown while running the test. **We assume that is also the reason why running the test can take up to 10 minutes.**

The problem is a known Eclipse issue and there are a few discussions in the following two defects:

```
https://bugs.eclipse.org/bugs/show_bug.cgi?id=128613
https://bugs.eclipse.org/bugs/show_bug.cgi?id=139699
```

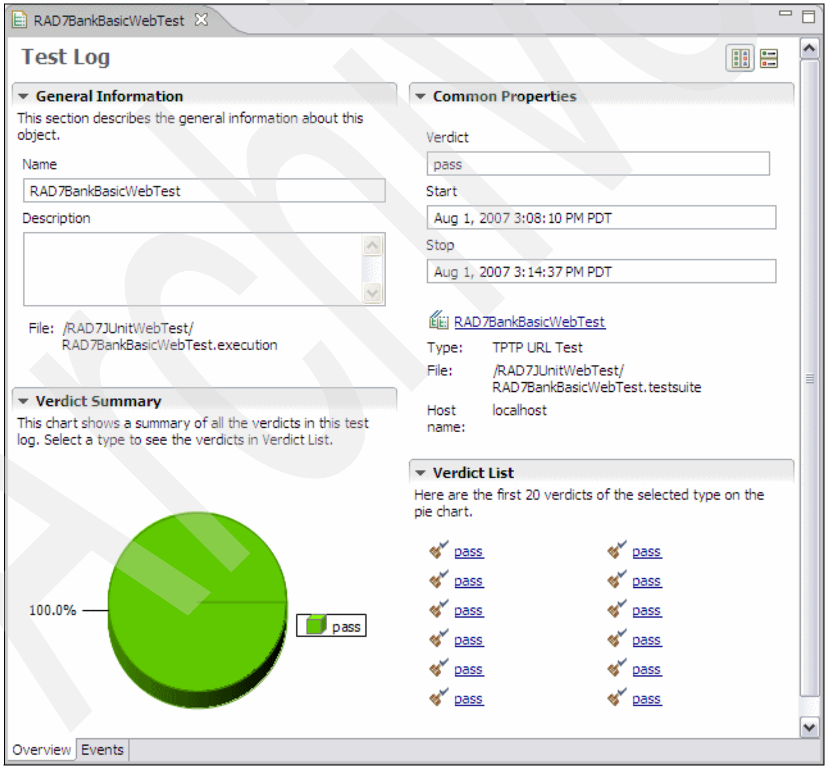
Do not get confused: As you can see when we analyze the test results, 100% of the tests passed. That is because none of the requests returned an HTTP code of 400 or greater—even those which threw an exception. But in the console you can see that an unexpected call to the /showException.jsp has occurred. If you want to see the exception message on the console, you have to modify the constructor of the itso.rad7.bank.exception.ITSOBankException class in the RAD7Java project:

```
public ITSOBankException(String message) {
    super(message);
    printStackTrace();
    System.err.println(message);
}
```

Analyzing the test results

When the test run is finished, the execution result appears in the Test Navigator view. To analyze the test results, do these steps:

- Double-click the execution result `RAD7BankBasicWebTest[<timestamp>]` file in the Test Navigator view. The Test Log Overview tab is displayed (Figure 21-15).
- The test log gives the test verdict and the starting and stopping time of the test run. The verdict can be one of the following:
  - *fail*: One or more requests returned a HTTP code of 400 or greater, or the server could not be reached during playback.
  - *pass*: No request returned a code of 400 or greater.
  - *inconclusive*: The test did not run to completion.
  - *error*: The test itself contains an error.



RAD7BankBasicWebTest

Test Log

General Information

This section describes the general information about this object.

Name: RAD7BankBasicWebTest

Description:

File: /RAD7JUnitWebTest/RAD7BankBasicWebTest.execution

Verdict Summary

This chart shows a summary of all the verdicts in this test log. Select a type to see the verdicts in Verdict List.

Common Properties

Verdict: pass

Start: Aug 1, 2007 3:08:10 PM PDT

Stop: Aug 1, 2007 3:14:37 PM PDT

RAD7BankBasicWebTest

Type: TPTP URL Test

File: /RAD7JUnitWebTest/RAD7BankBasicWebTest.testsuite

Host name: localhost

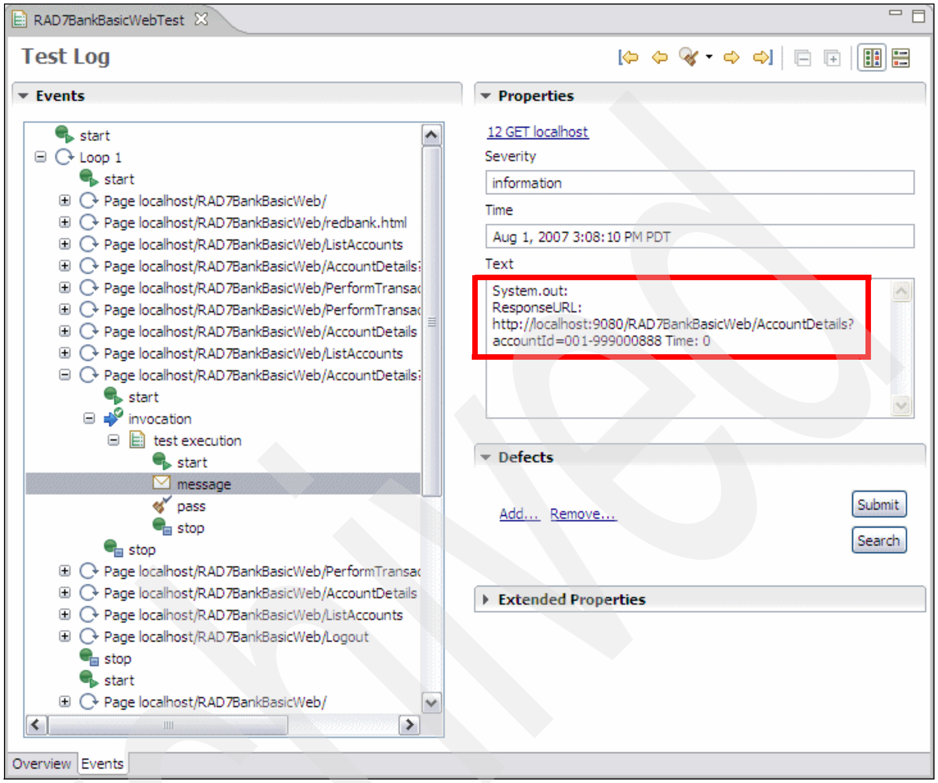
Verdict List

Here are the first 20 verdicts of the selected type on the pie chart.

pass	pass
pass	pass
pass	pass
pass	pass
pass	pass
pass	pass

Figure 21-15 Test Log Overview tab

- Click the **Events** tab to get detailed information about each single HTTP request (Figure 21-16).



```
start
Loop 1
  start
  Page localhost/RAD7BankBasicWeb/
  Page localhost/RAD7BankBasicWeb/redbank.html
  Page localhost/RAD7BankBasicWeb/ListAccounts
  Page localhost/RAD7BankBasicWeb/AccountDetails?
  Page localhost/RAD7BankBasicWeb/PerformTransac
  Page localhost/RAD7BankBasicWeb/PerformTransac
  Page localhost/RAD7BankBasicWeb/AccountDetails
  Page localhost/RAD7BankBasicWeb/ListAccounts
  Page localhost/RAD7BankBasicWeb/AccountDetails?
    start
    invocation
      test execution
        start
        message
        pass
        stop
    stop
  Page localhost/RAD7BankBasicWeb/PerformTransac
  Page localhost/RAD7BankBasicWeb/AccountDetails
  Page localhost/RAD7BankBasicWeb/ListAccounts
  Page localhost/RAD7BankBasicWeb/Logout
  stop
  start
  Page localhost/RAD7BankBasicWeb/
```

12 GET localhost

Severity: information

Time: Aug 1, 2007 3:08:10 PM PDT

Text:

```
System.out:
ResponseURL:
http://localhost:9080/RAD7BankBasicWeb/AccountDetails?
accountId=001-999000888 Time: 0
```

Figure 21-16 Test Log Events tab

Generating test reports

Based on a test execution results file, you can generate different kinds of analysis reports:

- **HTTP Page Response Time report**: Bar graph showing the seconds required to process each page in the test and the average response time for all pages.
- **HTTP Page Hit Rate report**: Bar graph showing the hits per second to each page and the total hit rate for all pages.

You can also generate the Test Pass report in the same way as for the basic JUnit tests.

> **Note:** To view the reports, you have to install the Scalable Vector Graphics (SVG) browser plug-in.

HTTP Page Response Time report

To generate an HTTP Page Response Time report, do these steps:

- Right-click the `RAD7BankBasicWebTest` test suite in the Test Navigator view and select **Report**.
- In the New Report dialog, select **HTTP Page Response Time**, clear **Open Report Automatically**, then click **Next**.
- In the New Report dialog, accept the parent folder (`RAD7JUnitWebTest`), enter `RAD7BankBasicWebTest_HTTPPageResponseTime` in the name field, and click **Finish**.
- If you have multiple test execution results, in the HTTP Report Generator dialog, you have to select the test execution result for which the report is generated and click **Finish**.
- An HTTP Page Response Time report is generated. Select the generated file and **Open With** → **Web Browser** (Figure 21-17).

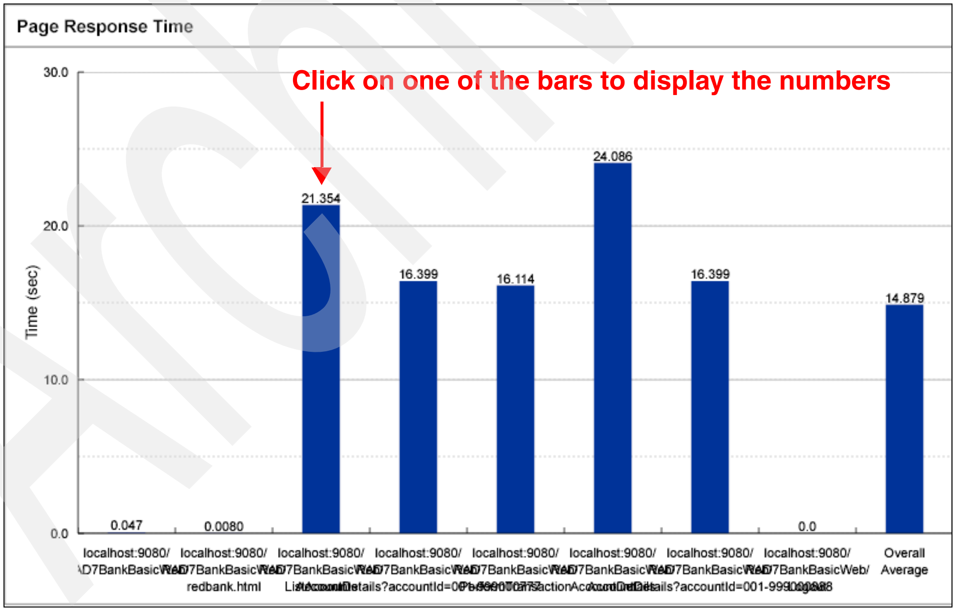


Figure 21-17 HTTP Page Response Time report

HTTP Page Hit Rate report

To generate a HTTP Page Hit Rate report, do these steps:

- Right-click the `RAD7BankBasicWebTest` test suite in the Test Navigator view and select **Report**.
- In the New Report dialog, select **HTTP Page Hit Rate** and click **Next**.
- In the New Report dialog, accept the parent folder (`RAD7JUnitWebTest`), enter `RAD7BankBasicWebTest_HTTPPageHitRate` in the name field, and click **Finish**.
- If you have multiple test execution results, in the HTTP Report Generator dialog, you have to select the test execution result for which the report is generated and click **Finish**.
- An HTTP Page Hit Rate report is generated. Select the generated file and **Open With** → **Web Browser** (Figure 21-18).

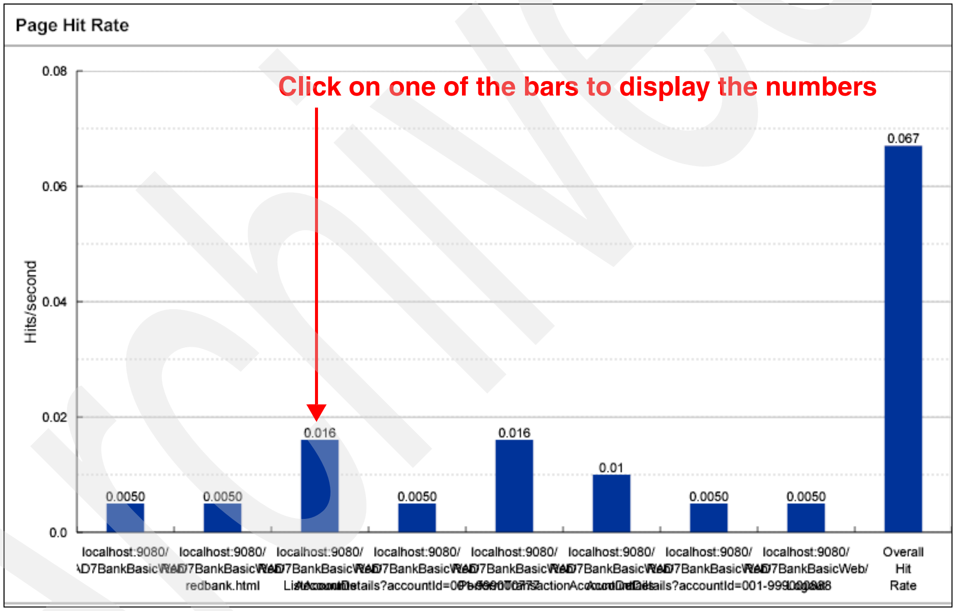


Figure 21-18 HTTP Page Hit Rate report

More information

For more information, consult the Application Developer Help facility and expand **Testing functionality and performance** → **Testing applications (Eclipse Hyades project)**.

22

Debug local and remote applications

The debugging features included with IBM Rational Application Developer V7.0 can be used to debug a wide range of applications in several languages running either on local test environments (including local Web applications) or on remote servers running systems such as WebSphere Application Server or WebSphere Portal.

In this chapter, the main features available to developers for debugging are described, the new debug tooling features included with Rational Application Developer v7.0 are introduced, and two examples of how to use the debugger are provided. The first demonstrates debugging a Web application running on a local test environment, and the second shows how to debug an application running on a remote Web Application Server server. Finally, a list of potential sources for further information is supplied.

The chapter is organized into the following sections:

- Overview of Application Developer debugging tools
- Summary of new features in V7.0
- Debugging a Web application on a local server
- Debugging a Web application on a remote server
- Jython debugger
- More information

Overview of Application Developer debugging tools

This section provides an overview of the basic debug tooling features included in Application Developer v7.0.

The topics covered here are:

- Supported languages and environments
- Basic Java debugging features
- XSLT debugging
- Remote debugging

Supported languages and environments

Application Developer includes support for debugging many different languages and environments. These are as follows:

- Java
- JavaScript
- DB2 Stored Procedures (either in Java or SQL)
- EGL
- XSL Transformations (XSLT)
- SQLJ
- Jython Scripts for WebSphere Application Server administration
- Mixed language applications (for example XSLT called from Java)
- WebSphere Application Server (servlets, JSPs, EJBs, Web services)
- WebSphere Portal (portlets)

Applications in all these languages and environments can be debugged within Application Developer using a similar process of setting breakpoints, running the application in debug mode and within the Debug perspective stepping through the code to track variables and logic and hopefully find and fix problems. Furthermore, the interface for debugging within the Debug perspective is intended to be as consistent as possible across all these languages and environments.

Basic Java debugging features

The following section gives a brief description of the main debugging features available within Application Developer for Java applications and explains how they would typically be used. Although this description focuses mainly on the tools available for Java, most of the features are available when debugging other languages.

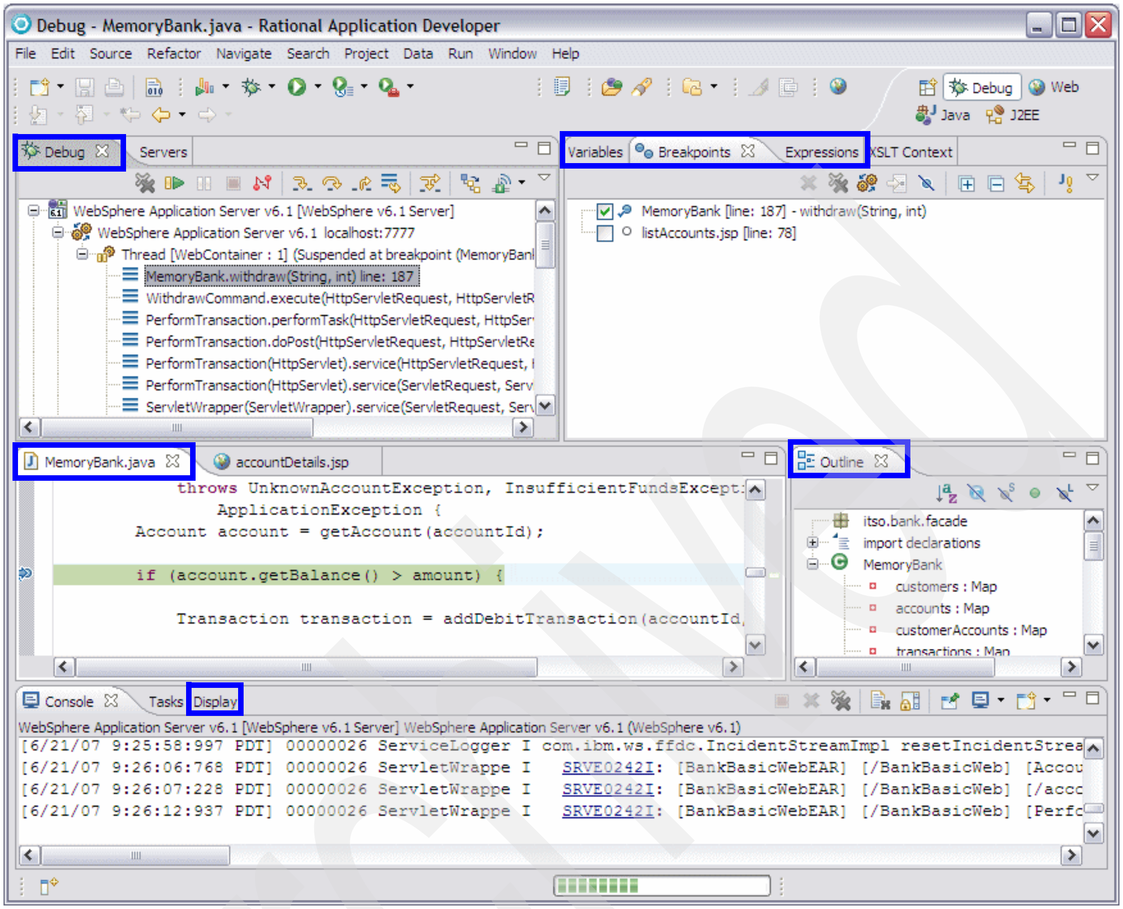
Views within the Debug perspective

When running an application in debug mode and a breakpoint is reached, the user is prompted to see if they want to move to the Debug perspective. Note that it is possible to do debugging in other perspectives, however the Debug perspective provides the full set of views available for debugging.

By default, when debugging Java, the views shown in the Debug perspective are as follows:

- **Source view**—Shows the file of the source code that is being debugged, highlighting the current line being executed.
- **Outline view**—Contains a list of variables and methods for the code listing shown in the display view.
- **Debug view**—Shows a list of all active threads, and a stack trace of the thread that is currently being debugged.
- **Servers view**—Useful if the user wants to start or stop test servers while debugging.
- **Variables view**—Given the selected source code file shown in the Debug view, the Variables view shows all the variables available to that class and their values.
- **Breakpoints view**—Shows all breakpoints in the current workspace and gives a facility to activate/de-activate them, remove them, change their properties, and to import/export a set of them to other developers.
- **Display view**—Allows the user to execute any Java command or evaluate an expression in the context of the current stack frame.
- **Expressions view**—During debugging, the user has the option to inspect or display the value of expressions from the code or even valuate new expressions. The Expressions view contains a list of expressions and values which the user has evaluated and then selected to track.
- **Console view**—Shows the output to `System.out`.
- **Tasks view**—Shows any outstanding source code errors, warnings or informational messages for the current workspace.
- **Error Log**—Shows all errors and warnings generated by plug-ins running in the work space.

Figure 22-1 shows a workspace in the Debug perspective.



```
WebSphere Application Server v6.1 [WebSphere v6.1 Server]
  WebSphere Application Server v6.1  localhost:7777
    Thread [WebContainer : 1] (Suspended at breakpoint (MemoryBanl
      MemoryBank.withdraw(String, int) line: 187
      WithdrawCommand.execute(HttpServletRequest, HttpServletR
      PerformTransaction.performTask(HttpServletRequest, HttpSer
      PerformTransaction.doPost(HttpServletRequest, HttpServletRe
      PerformTransaction(HttpServlet).service(HttpServletRequest, I
      PerformTransaction(HttpServlet).service(ServletRequest, Serv
      ServletWrapper(ServletWrapper).service(ServletRequest, Serv
```

```
☑ MemoryBank [line: 187] - withdraw(String, int)
□ listAccounts.jsp [line: 78]
```

```
                throws UnknownAccountException, InsufficientFundsExcept:
                    ApplicationException {
            Account account = getAccount(accountId);

            if (account.getBalance() > amount) {

                Transaction transaction = addDebitTransaction(accountId,
```

```
WebSphere Application Server v6.1 [WebSphere v6.1 Server] WebSphere Application Server v6.1 (WebSphere v6.1)
[6/21/07 9:25:58:997 PDT] 00000026 ServiceLogger I com.ibm.ws.ffdc.IncidentStreamImpl resetIncidentStrea
[6/21/07 9:26:06:768 PDT] 00000026 ServletWrappe I   SRVE0242I: [BankBasicWebEAR] [/BankBasicWeb] [Accou
[6/21/07 9:26:07:228 PDT] 00000026 ServletWrappe I   SRVE0242I: [BankBasicWebEAR] [/BankBasicWeb] [/accc
[6/21/07 9:26:12:937 PDT] 00000026 ServletWrappe I   SRVE0242I: [BankBasicWebEAR] [/BankBasicWeb] [Perfc
```

Figure 22-1 Typical application running in the Debug perspective

Debug functions

From the Debug view, you can use the functions available from the icon bar to control the execution of the application. The following icons are available:

- **Resume** (F8): Runs the application to the next breakpoint.
- **Suspend**: Suspends a running thread.
- **Terminate**: Terminates a process.
- **Disconnect**: Disconnects from the target when debugging remotely.
- **Remove All Terminated Launches**: Removes terminated executions from the Debug view.
- **Step Into** (F5): Steps into the highlighted statement.

- **Step Over** (F6): Steps over the highlighted statement.
- **Step Return** (F7): Steps out of the current method.
- **Drop to Frame**: Provides the facility to reverse back to a higher method call in the current stack frame.
- **Use Step Filters/Step Debug** (Shift-F5): Enable/disable the filtering for the *step debug* functions.
- **Step-By-Step Mode**: Once the step-by-step debug feature is enabled in the Run/Debug preferences, this icon can also be used to toggle the feature.
- **Show Qualified Names** (from drop-down menu): Toggle option to show the full package name.
- or **Debug UI demon**: Provides a drop-down list for controlling the debugging of XSL transforms that are invoked by WebSphere applications.

Using these buttons and the information shown in the various views available in the Debug perspective, it should be possible to debug most problems.

Enable/disable Step Filter/Step Debug in the Debug view

In the Debug view toolbar, there is the Use Step Filters icon. This feature allows users to filter out Java classes that do not have to be stepped into while debugging. For example, usually it is not necessary for programers to step into code from the classes within the Sun and IBM Java libraries when step-by-step debugging, and by default these classes are in the step filter list. The facility is provided to add any class or package to this list.

To add new Java package to the Step Filter/Step Debug feature in the Debug view, do these steps:

- Select **Window** → **Preferences**.
- Expand **Run/Debug** → **Java and Mixed Language Debug** → **Step Filters**.
- Click **Add Filter**.
- Enter the new package or class you want to filter out and click **OK**.

The Step Filter/Step Debug feature can be toggled on and off by clicking **Step Filter** ( ) in the Debug view.

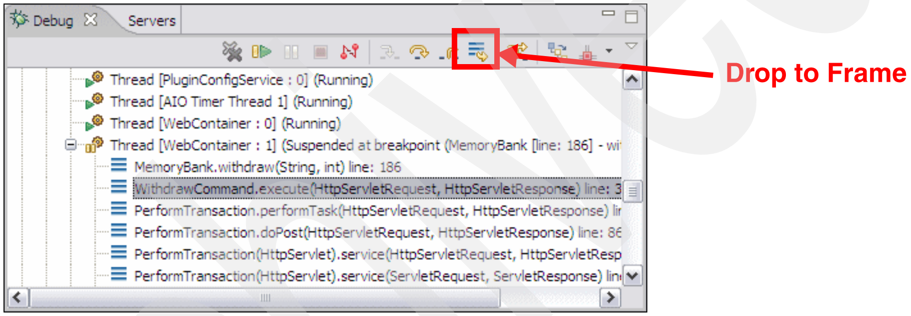
Drop to Frame

The drop-to-frame feature allows the control of an application to be reversed back to a previously executed line of code. This feature is available when debugging Java applications and Web applications running on WebSphere

Application Server and is useful when you want to test a block of code using a range of values.

For example, if a developer wants to test a method with the minimum and maximum permitted values for a given parameter, a breakpoint can be added at the end of the method, and the drop-to-frame feature can be used to back up the control of the application to the start of the method, change the parameters, and run it again.

When running an application in the Debug perspective, the stack frame can be seen in the Debug view (Figure 22-2). Drop to frame allows you to back up your application's execution to previous points in the call stack by selecting the desired method level from within the Debug view and then clicking **Drop To Frame** . This moves the control of the application to the top of the method selected in the Debug view.



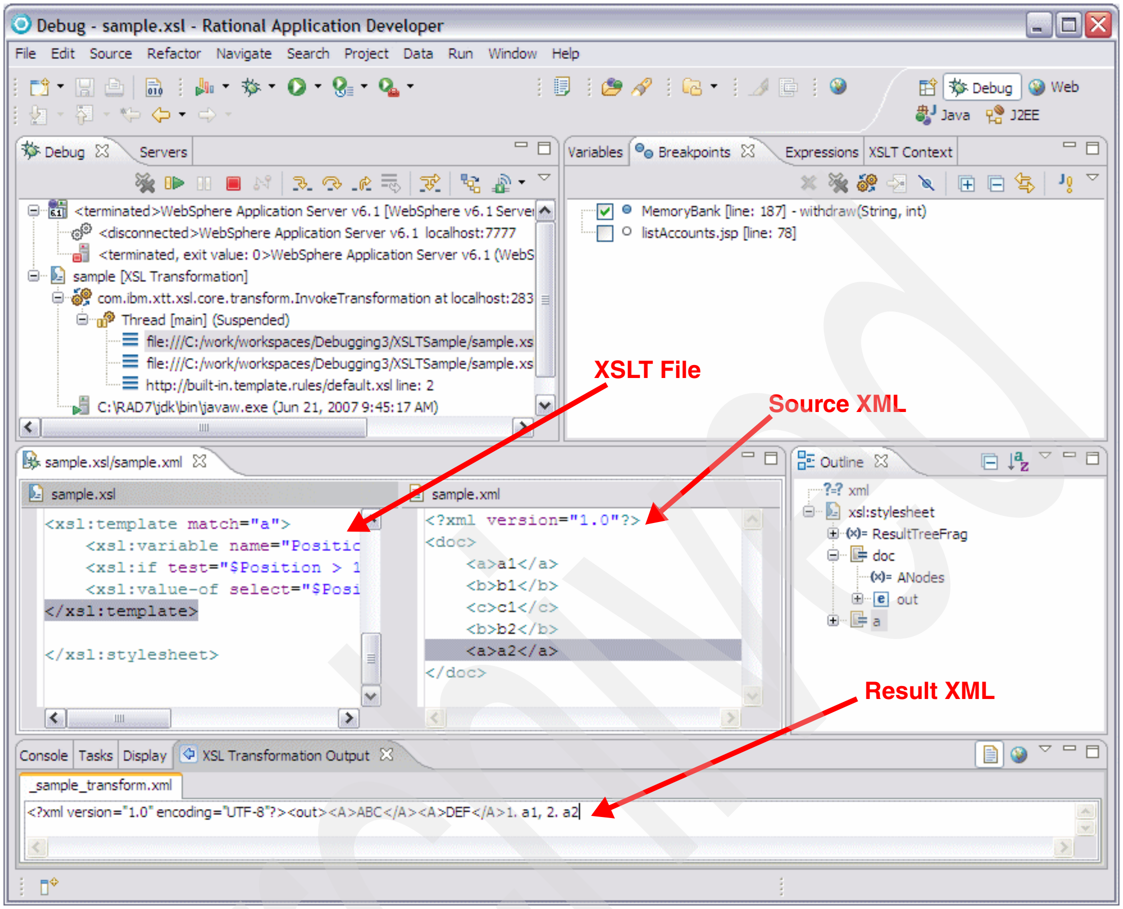
```
Thread [PluginConfigService : 0] (Running)
Thread [AIO Timer Thread 1] (Running)
Thread [WebContainer : 0] (Running)
Thread [WebContainer : 1] (Suspended at breakpoint (MemoryBank [line: 186] - wi
    MemoryBank.withdraw(String, int) line: 186
    WithdrawCommand.execute(HttpServletRequest, HttpServletResponse) line: 3
    PerformTransaction.performTask(HttpServletRequest, HttpServletResponse) lir
    PerformTransaction.doPost(HttpServletRequest, HttpServletResponse) line: 86
    PerformTransaction(HttpServlet).service(HttpServletRequest, HttpServletResp
    PerformTransaction(HttpServlet).service(ServletRequest, ServletResponse) lin
```

Figure 22-2 Drop to Frame Button in the Debug view.

XSLT debugging

XSL (EXtensible Stylesheet Language) Transformations (XSLT) is a language for transforming XML documents into XHTML or other XML documents. It is a declarative language expressed in XML and provides mechanisms to match tags from the source document to templates (similar to methods), store values in variables, basic looping/code branching and invoke other templates in order to build up the result document.

Also, XSLT files can use templates stored in other files and it is possible for XSLT files to become complicated. Application Developer features a full-featured XSL transformation debugger, and XSLT files can be debugged using a similar set of tools to that available for Java. Figure 22-3 shows the Debug perspective when debugging XSLT. From the Debug perspective, the user can step through the XSLT file and the source XML and watch the result XML being built element by element.


Debug - sample.xsl - Rational Application Developer
File Edit Source Refactor Navigate Search Project Data Run Window Help
Debug Web Java J2EE
Debug Servers
Variables Breakpoints Expressions XSLT Context
sample.xsl/sample.xml
sample.xsl
sample.xml
Outline
Console Tasks Display XSL Transformation Output
_sample_transform.xml
XSLT File
Source XML
Result XML

```
<terminated>WebSphere Application Server v6.1 [WebSphere v6.1 Serve
  <disconnected>WebSphere Application Server v6.1 localhost:7777
  <terminated, exit value: 0>WebSphere Application Server v6.1 (WebS
sample [XSL Transformation]
  com.ibm.xtt.xsl.core.transform.InvokeTransformation at localhost:283
    Thread [main] (Suspended)
      file:///C:/work/workspaces/Debugging3/XSLTSample/sample.xs
      file:///C:/work/workspaces/Debugging3/XSLTSample/sample.xs
      http://built-in.template.rules/default.xsl line: 2
  C:\RAD7\jdk\bin\javaw.exe (Jun 21, 2007 9:45:17 AM)
```

```
☑ ● MemoryBank [line: 187] - withdraw(String, int)
□ ○ listAccounts.jsp [line: 78]
```

```
<xsl:template match="a">
    <xsl:variable name="Positic
    <xsl:if test="$Position > 1
    <xsl:value-of select="$Posi
</xsl:template>

</xsl:stylesheet>
```

```
<?xml version="1.0"?>
<doc>
    <a>a1</a>
    <b>b1</b>
    <c>c1</c>
    <b>b2</b>
    <a>a2</a>
</doc>
```

```
?-? xml
xsl:stylesheet
  (x)= ResultTreeFrag
  doc
    (x)= ANodes
    e out
  a
```

```
<?xml version="1.0" encoding="UTF-8"?><out><A>ABC</A><A>DEF</A>1. a1, 2. a2
```

Figure 22-3 Debugging XSLT

The following views are useful when debugging XSLT:

- **Debug view**—Shows the XSL transformation running as an application with the stack frame for the current execution point.
- **XSLT Context view**—Shows the current context of the XML input for the selected stack frame.
- **Expressions view**—Can be used show the value of XSL expressions, including XPath expressions in the current context.
- **Variables view—**Currently visible XSLT variables.
- **XSL Transformation Output view**—Shows the serialized output of the transformation as it is produced.

- **Display view**—Shows the XSL file on the left side and the input XML file on the right. The line being executed for each is highlighted.
- **Breakpoints view**—Shows all breakpoints in the workspace, including those placed in an XSLT file.

To launch a debugging session for a XSLT file and its source XML file, simply highlight both files on the Project Explorer, and from the context menu, select **Debug As** → **XSLT Transformation.** This action steps into the first line of the XSL file, and from there, debugging can continue. The debug launch configuration window (from the Context Window use **Debug As** → **Debug**) allows the user to configure dependent files (both XSLT and Java) and other settings to guide the XSLT debugging. These configurations can then be saved to make launching the debugging quicker in the future.

It is also possible to debug a XSLT transformation called from a Java application. The easiest way to do this is to add a breakpoint in the Java code before the XSL transformation is called (typically from the `javax.com.transform.Transformer transform` method). Then use the **Debug** → **Java And Mixed Language Application** option for the Java class from the context menu. When the debugger stops at the breakpoint, use the Step Into option, and the debugger moves to debugging the XSL file and stops at the first line in the transformation.

Finally, it is also possible to debug Java code called from the XSL file. To do this you must use the launch configuration window and make sure that the Class path tab includes the projects that contain the Java code to be debugged. When stepping through the XSLT debugger, it will be possible to Step Into the call to the Java method.

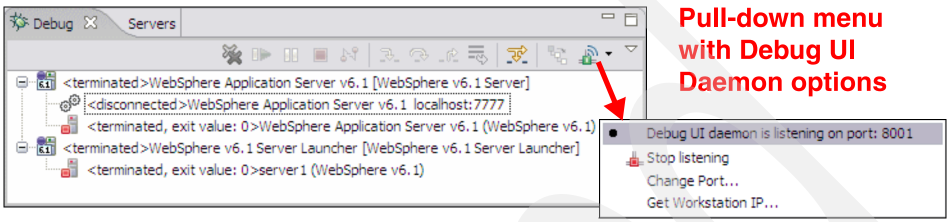
Remote debugging

The debugger provided with Application Developer allows a programmer to attach to a Java application running on a remote machine and control its execution. This is particularly useful when debugging an application that cannot be run on the development machine.

The application that is debugged remotely must be compiled with debug information on (within Application Developer see the **Java** → **Compiler** and the Classfile Generation attributes), and when the application is launched, the appropriate JVM parameters must be supplied to configure the IP address and port for the debugger machine. This s setting varies between different JVMs.

When starting the debug session, use **Debug As** → **Debug** (from the Project Explorer context menu) and create a new configuration for Remote Java Application. Make sure the Host and Port are configured for the target machine and that the source tab is filled out with the appropriate source code, then click Debug and the remote debugging session will start.

The **Debug UI daemon** is a feature available from the toolbar of the Debug view which allows developers to debug XSL transformations which are invoked by a WebSphere application.



<terminated>WebSphere Application Server v6.1 [WebSphere v6.1 Server]
<disconnected>WebSphere Application Server v6.1 localhost:7777
<terminated, exit value: 0>WebSphere Application Server v6.1 (WebSphere v6.1)
<terminated>WebSphere v6.1 Server Launcher [WebSphere v6.1 Server Launcher]
<terminated, exit value: 0>server1 (WebSphere v6.1)

Pull-down menu with Debug UI Daemon options

Figure 22-4 Setting Debug UI Daemon options

When enabled, the daemon listens on a port (which you can configure) and if during the WebSphere debug session the application invokes an XSL transformation, Application Developer will step into the XSLT file and debugging continues. You can disable the feature by selecting the appropriate option from the drop-down menu in the Debug view.

Summary of new features in V7.0

The debugging facilities available with Rational Application Developer Version 7.0 are very similar to those available in the previous version. The main areas of enhancements are as follows:

- **New Debugger for WebSphere Jython scripts**—Using the Debug Launch configuration, you can launch a debugging session for a given Jython script (with extension .py or .jy) on either the local test server or a server running on a remote machine. If the target environment is on a remote machine, then the host and port numbers must be configured in the **wsadmin arguments** field. Refer to the Application Developer online help for details on this feature.
- **New Show Running Threads filter**—When debugging, there are often many extra threads shown in the debug view that are not useful for finding a fault in an application under development. This is especially the case when debugging Web applications, where the Application Server starts several threads that are very unlikely to be the cause of an application problem. To show only threads that are suspended, bring up the context menu of the thread being debugged in the Debug view and toggle the **Show Running Threads** filter.

- **Stored procedure debugging for DB2 V9**—This feature allows a user to debug Java and DB2 stored procedures running on a local or remote DB2 server. The Debug Launch configuration editor provides fields to specify a stored procedure on a DB2 database to debug, arguments to pass to the procedure and the associated source code. If the database server is configured correctly, the debugger will launch and allow debugging to continue. Application Developer has a detailed help chapter on this feature.
- **Ability to step into JVM classes**—Previously the `java.*` and `javax.*` packages were not visible from the Step Filters preferences (**Window → Preferences → Run/Debug → Java and Mixed Language Debug → Step Filters**) and were always filtered. Now these packages appear in the step filer list and it is possible to de-select them and therefore, when debugging, it is possible to step into classes from this package. The default setting is to leave them remaining filtered.
- **New checks for invalid XSL breakpoints**—In an XSL file there are several places where a breakpoint has no effect, for example blank lines, `<xsl:output>` lines and XML declarations. When the user attempts to add a breakpoint to an invalid line they now receive the message *Cannot Add Breakpoint*.
- **Variables view structure**—The variables view is now, by default, structured into columns. The use of columns can be toggled from the **Layout → Show Columns** menu option from the drop-down arrow menu in the Variables view. Also, step-by-step debugging variables that change value are highlighted in a different color.
- **Performance improvements for JSP breakpoints**—The performance of JSP breakpoint creation and modification while connected to WebSphere has been improved.

Debugging a Web application on a local server

This section steps through a Web application scenario where a sample application is run on the local Application Developer test server and the debugging facilities are used to step through the code and watch the behavior of the application.

The debug example includes the following tasks to demonstrate the debug tooling:

- Importing the sample application
- Running the sample application in debug mode
- Setting breakpoints in a Java class
- Watching variables

- Evaluating and watching expressions
- Working with breakpoints
- Set breakpoints in a JSP
- Debugging a JSP

Importing the sample application

The following instructions show how to set up your workspace for the sample application. We use the ITSO RedBank Web application sampled developed in Chapter 12, “Develop Web applications using JSPs and servlets” on page 465 to demonstrate the debug facilities.

If you have the necessary projects (`RAD7BankBasicEAR`, `RAD7BankBasicWeb`, `RAD7Java`) in the workspace, you can skip this step.

To import the ITSO RedBank JSP and servlet Web application project interchange file, do these steps:

- In the Web perspective select **File** → **Import** → **Project Interchange** and click **Next**.
- In the Import Projects screen click **Browse** to locate the file:

  ```
  c:\7501code\zInterchangeFiles\webapps\RAD7BankBasic.zip
  ```

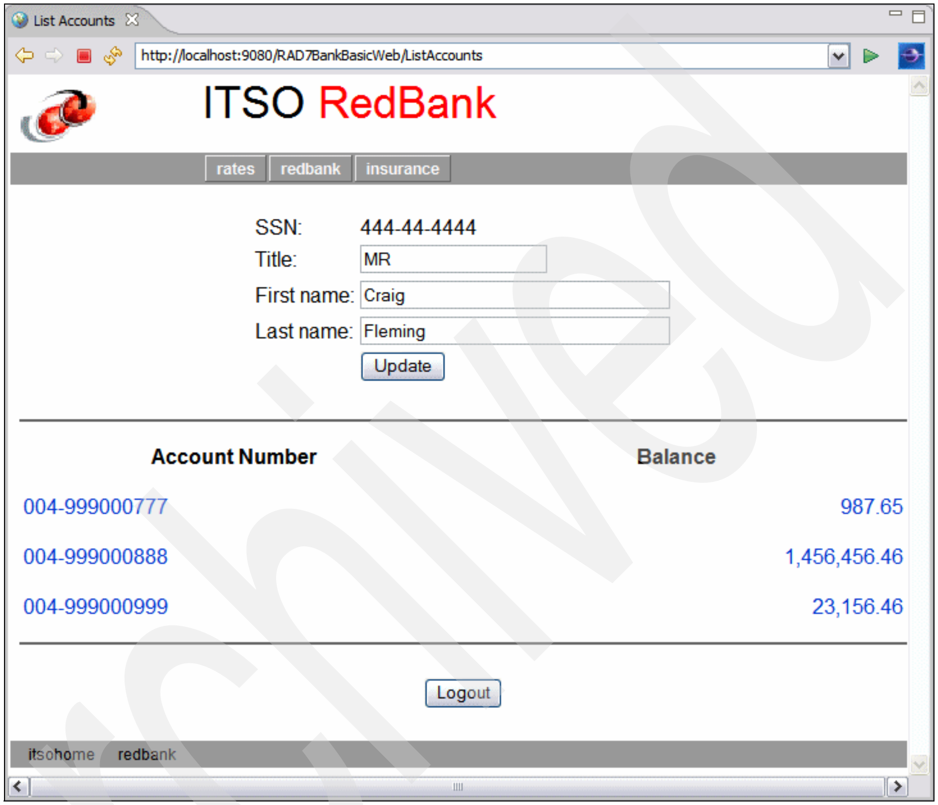
- Select the projects that are not in the workspace, and click **Finish**.

Running the sample application in debug mode

To verify that the sample application was imported properly, run the sample Web application on the WebSphere Application Server V6.1 test server in debug mode as follows:

- In the Project Explorer, expand **RAD7BankBasicWeb** → **WebContent**.
- Right-click **index.html** and select **Debug As** → **Debug on Server**. Note that no database configuration is required here, as the sample Bank Basic application used here stores the accounts on start-up in a `HashMap` from the constructor for the `MemoryBank` class.
- If the Server Selection dialog opens, select **Choose an existing server**, select **WebSphere Application Server v6.1**, and click **Finish**.This will start the server, publish the application to the server, and bring up a browser showing the Index page.
- If the server is already running in normal (non-debug) mode, you are prompted to switch mode. Click **OK**, and the server restarts in debug mode.

- When the Index page is displayed, select the **redbank** tab, enter `444-44-4444` in the Customer SSN field, and click **Submit**.
- The list of accounts for that customer are displayed (Figure 22-5). If you can see these results, then the application is working fine in debug mode.



SSN: 444-44-4444
Title: MR
First name: Craig
Last name: Fleming

Account Number	Balance
004-999000777	987.65
004-999000888	1,456,456.46
004-999000999	23,156.46

Figure 22-5 Customer details for the RedBank application

Setting breakpoints in a Java class

Breakpoints are indicators to the debugger that it should stop execution at that point in the code and let the user inspect the current state and step through through the code. Breakpoints can be set to always trigger when the execution point reaches it or when a certain condition has been met.

In the ITSO RedBank sample application, before the balance of an account is updated after withdrawal of funds from an account, the new balance is compared to see if it goes below zero.

If there are adequate funds, the withdrawal will complete. If there are not enough funds in the account, an `InvalidTransactionException` is thrown from the `Account` class and the `showException.jsp` is displayed to the user showing an appropriate message.

In this example, we set a breakpoint where the logic tests that the amount to withdraw does not exceed the amount that exists in the account.

To add a breakpoint in the code, do these steps:

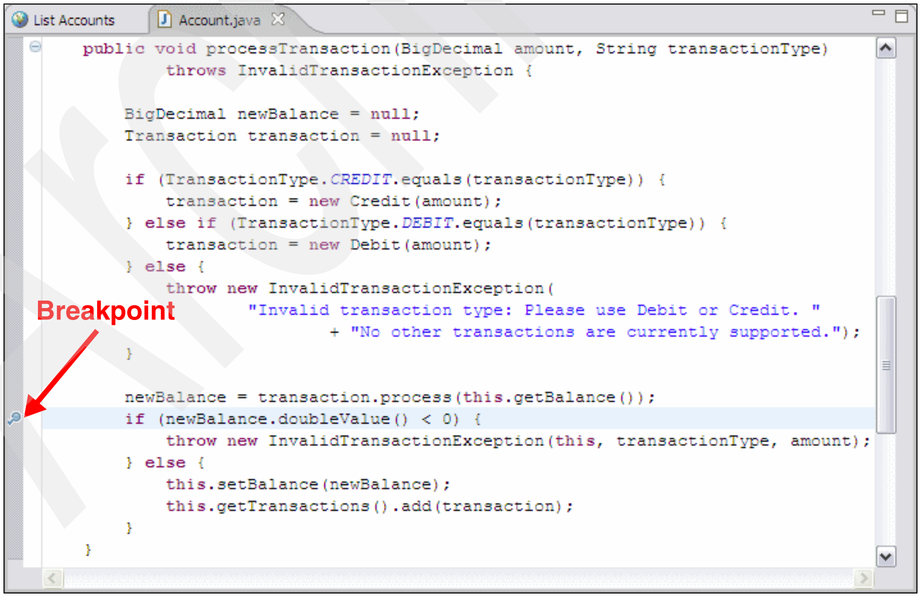
- In the Project Explorer select and expand **RAD7Java** → **src** → **itso.rad7.bank.model**, and open **Account.java** in the Java editor.
- Locate the `processTransaction` method.

> **Tip:** You can use the Outline view or expand `Account.java` in the Project Explorer to find the `processTransaction` method quickly in the source code.

- Place the cursor in the gray bar (along the left edge of the editor area) on the following line of code in the `processTransaction` method:

```
if (newBalance.doubleValue() < 0) {
```

- Double-click to set a breakpoint marker (Figure 22-6).

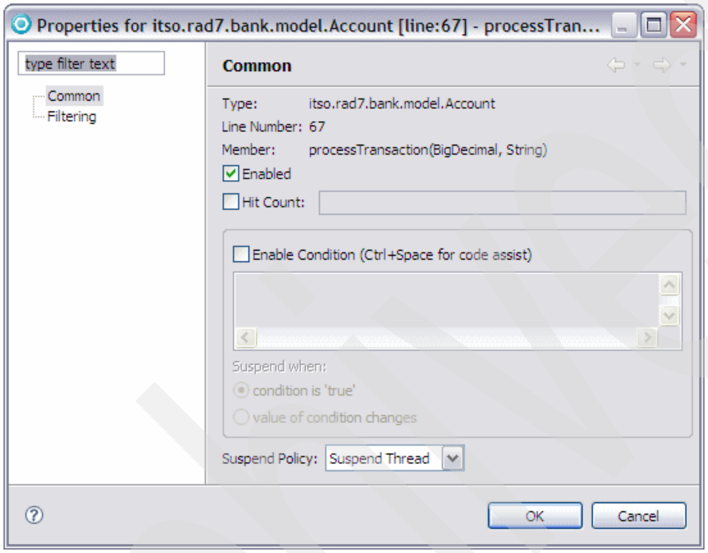


```
public void processTransaction(BigDecimal amount, String transactionType)
        throws InvalidTransactionException {

    BigDecimal newBalance = null;
    Transaction transaction = null;

    if (TransactionType.CREDIT.equals(transactionType)) {
        transaction = new Credit(amount);
    } else if (TransactionType.DEBIT.equals(transactionType)) {
        transaction = new Debit(amount);
    } else {
        throw new InvalidTransactionException(
                "Invalid transaction type: Please use Debit or Credit. "
                        + "No other transactions are currently supported.");
    }

    newBalance = transaction.process(this.getBalance());
    if (newBalance.doubleValue() < 0) {
        throw new InvalidTransactionException(this, transactionType, amount);
    } else {
        this.setBalance(newBalance);
        this.getTransactions().add(transaction);
    }
}
```

Figure 22-6 Setting a breakpoint in Java

> **Note:** Enabled breakpoints are indicated with a blue circle. If the enabled breakpoint is successfully placed, it is indicated with a select mark overlay.

- Right-click the breakpoint and select **Breakpoint Properties**.
- In the Breakpoint Properties window, you can change the details of the breakpoint (Figure 22-7).



Properties for itso.rad7.bank.model.Account [line:67] - processTran...

type filter text

Common
Filtering

Common

Type: itso.rad7.bank.model.Account
Line Number: 67
Member: processTransaction(BigDecimal, String)
☑ Enabled
□ Hit Count:

□ Enable Condition (Ctrl+Space for code assist)

Suspend when:
● condition is 'true'
○ value of condition changes

Suspend Policy: Suspend Thread

Figure 22-7 Breakpoint properties

- If the **Hit Count** property is set, it causes the breakpoint to be triggered only when the line has been executed as many times as the hit count specified. Once triggered, the breakpoint is disabled.
- Selecting **Enable Condition** allows breakpoints to trigger only when the condition specified in the entry field evaluates to *true*. This condition is a Java expression. When this is enabled, the breakpoint is marked with a question mark on the breakpoint, which indicates that it is a conditional breakpoint.

  For example, select **Enable Condition**, enter the expression `amount.doubleValue() >= 50.00`, select **condition is 'true'**. Now the breakpoint will only trigger on transactions over $50.00

- Click **OK** to close the breakpoint properties.

It should now be possible to run the application and trigger the breakpoint. Note that it is not necessary to restart the Application Server for the new breakpoint to work.

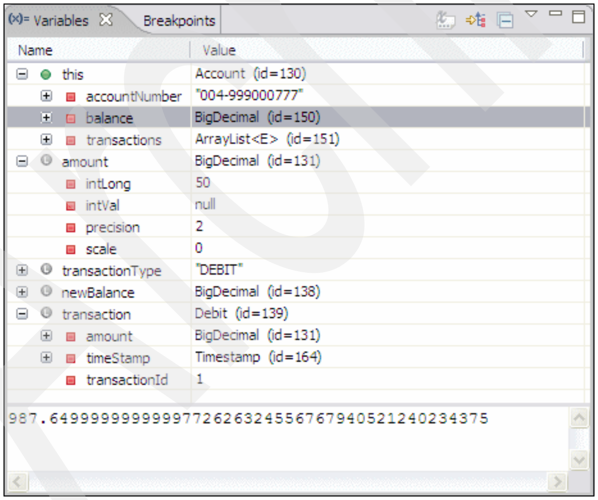
To trigger the breakpoint, perform the following steps:

- In the home page, click on the **redbank** tab, enter `444-44-4444` as the customer number and click **Submit**.
- In the List Accounts page, select the top account (`004-999000777`).
- In the Account Details page, select **Withdraw** and enter 50 in the Amount field.
- Click **Submit**. The `processTransaction` method is executed and the new breakpoint triggered. Application Developer prompts you to enter the Debug perspective (click **Yes**), where it is possible to watch and edit variables and step through code.

> **Note:** Within the Debug view, it is now possible to show only the suspended threads, as described in “Summary of new features in V7.0” on page 1049. To do this, toggle the **Show Running Threads** option on the context menu of the Debug view.

Watching variables

The Variables view displays the current values of the variables in the selected stack frame (Figure 22-6).



Name	Value
⊟ ● this	Account (id=130)
⊞ ■ accountNumber	"004-999000777"
⊞ ■ balance	BigDecimal (id=150)
⊞ ■ transactions	ArrayList<E> (id=151)
⊟ ○ amount	BigDecimal (id=131)
■ intLong	50
■ intVal	null
■ precision	2
■ scale	0
⊞ ○ transactionType	"DEBIT"
⊞ ○ newBalance	BigDecimal (id=138)
⊟ ○ transaction	Debit (id=139)
⊞ ■ amount	BigDecimal (id=131)
⊞ ■ timeStamp	Timestamp (id=164)
■ transactionId	1

```
987.649999999999772626324556767940521240234375
```

Figure 22-8   Displaying variables

In the variables view expand **this** and select **balance**. Although balance is of type `BigDecimal`, a string representation of its value is shown in the bottom section of the window.

The plus sign (+) next to a variable indicates that it is an object. By clicking on the plus sign, it is possible to look into instance variables associated with the object, for example, clicking on the plus sign next to the `transaction` object shows the `amount`, `timeStamp`, and `transactionId` variables. Select **timeStamp** to see today's date and time.

Follow these steps to see how you can track the state of a variable, while debugging the method:

- Click **Step Over** in the Debug view (or press F6) to execute the current statement.
- Click **Step Over** again and the balance is updated. Note that the color of balance changes in the Variables view.

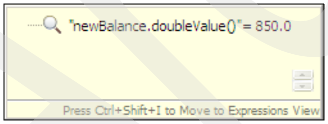
It is possible to test the code with some other value for any of these instance variables; a value can be changed by selecting **Change Value** from its context menu. An entry field opens where the value can be changed.

Note that to change a `BigDecimal`, you have to enter a constructor, such as `new BigDecimal(900.00)`.

Evaluating and watching expressions

When debugging, it is often useful to be evaluate an expression made up of several variables within the current application context.

To view the value of an expression within the code, select the expression (for example, `newBalance.doubleValue()` in the breakpoint line), right-click, and select **Inspect**. The result opens in a pop-up window showing the value (Figure 22-9).



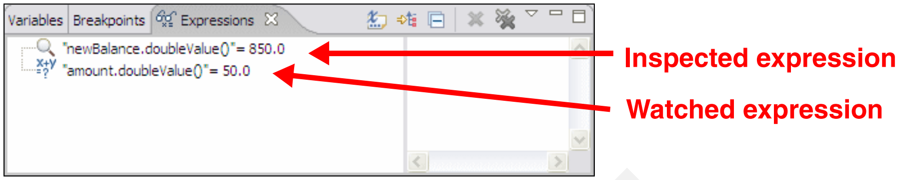
"newBalance.doubleValue()"= 850.0

Press Ctrl+Shift+I to Move to Expressions View

Figure 22-9 Inspect Pop-up window

To move the results to the Expressions view (Figure 22-10) so that the value of the expression can continue to be monitored, press Ctrl+Shift+I.

To watch an expression, right-click in the Expressions view and select **Add Watch Expression**. In the Add Watch Expression dialog, enter an expression, such as, `amount.doubleValue()`.



"newBalance.doubleValue()"= 850.0
"amount.doubleValue()"= 50.0

Figure 22-10 Inspecting a variable in Expressions view

> **Note:** The Expressions view contains a list of watched expressions (marked by a X+Y =? symbol) and a list of inspected expressions marked by a a 🔍 symbol (Figure 22-10). The difference between these is that the value shown for watched expressions changes with the underlying value as the user steps through the code, while an inspected expression will remain showing the value it held when it was first inspected.

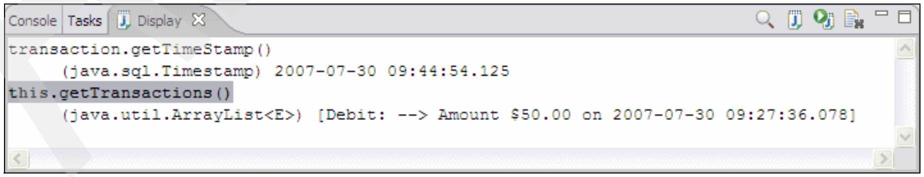
Using the Display view

To evaluate an expression in the context of the currently suspended thread which does not come from the source code, use the Display view.

- From the Workbench, select **Windows** → **Show View** → **Display**.
- Enter the expression `transaction.getTimeStamp()` in the Display View, then highlight the expression, right-click, and select **Display**.

> **Tip:** When entering an expression in the Display Panel, it is possible to use code assist (Ctrl+Spacebar).

Each expression is executed, and the result is displayed as shown in Figure 22-11. This is a useful way to evaluate Java expressions or even call other methods during debugging, without having to make changes in your code and recompile.



```
transaction.getTimeStamp()
     (java.sql.Timestamp) 2007-07-30 09:44:54.125
this.getTransactions()
     (java.util.ArrayList<E>) [Debit: --> Amount $50.00 on 2007-07-30 09:27:36.078]
```

Figure 22-11 Expression and evaluated result in display view

> **Note:** To move the results of the display to the Expression view, press Ctrl+Shift+I.

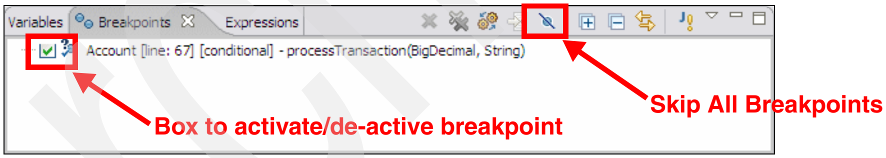
- You can also highlight any expression in the source code, right-click, and select **Watch** (or **Inspect**). The result is shown in the Expressions view.
- Select **Remove** from the context menu to remove expressions or variables from the Expressions views. In the Display view, just select the text and delete it.

Working with breakpoints

To enable a breakpoint in the code, double-click in the grey area of the left frame (or use the context menu on the left side of the frame) for the line of code the breakpoint is required for. To remove double-click on it again, and to disable the breakpoint, select **Disable Breakpoint** from its context menu.

Alternatively, once they have been created, the breakpoints can be enabled and disabled from the Breakpoints view (Figure 22-12). If the breakpoint is un-selected in the Breakpoints view, it is skipped during execution.

To disable or enable all breakpoints, click the **Skip All Breakpoints** icon, as seen in Figure 22-12.If this option is selected then all breakpoints are skipped during execution.



☑ Account [line: 67] [conditional] - processTransaction(BigDecimal, String)

Figure 22-12 Enabling/Disabling Breakpoints

It is possible to export a set of breakpoints, including the conditions and hit count properties so that they can be shared across a development team. To do this, from the context menu of the Breakpoints view select **Export Breakpoints** and the breakpoints are saved as a `bkpt` file to the selected location.

Before continuing with debugging of JSPs in this example, click **Resume** to continue execution.

Note that you do not see the Web page automatically in the Debug perspective. Click the view with the World icon (in the same pane as the source code) to see the resulting Web page. Alternatively, switch to the Web perspective.

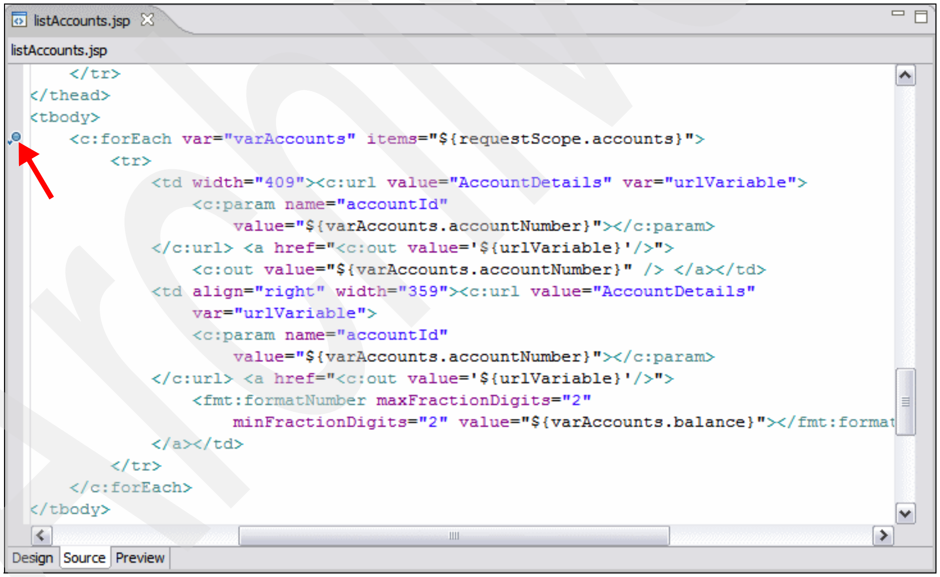
Tip: Java exception breakpoints are a different kind of breakpoint that are triggered when a particular exception is thrown. These breakpoints can be set by selecting **Run** → **Add Java Exception Breakpoint**.

Set breakpoints in a JSP

You can also set breakpoints in for JSPs. Within the source view of a JSP page, you can set breakpoints inside JSP scriptlets, JSP directives, and lines which use JSP tag libraries. You cannot set breakpoints in lines with only HTML code.

In the following example, we set a breakpoint in the `listAccounts.jsp` at the point where the JSP displays a list of accounts for the customer.

- In the Web Perspective expand **RAD7BankBasicWeb** → **WebContent** and open the `listAccounts.jsp` in the editor. Select the **Source** tab.
- Set a breakpoint by double-clicking in the grey area next to the desired line of code (Figure 22-13).



```
    </tr>
</thead>
<tbody>
    <c:forEach var="varAccounts" items="${requestScope.accounts}">
        <tr>
            <td width="409"><c:url value="AccountDetails" var="urlVariable">
                <c:param name="accountId"
                    value="${varAccounts.accountNumber}"></c:param>
            </c:url> <a href="<c:out value='${urlVariable}'/>">
                <c:out value="${varAccounts.accountNumber}" /> </a></td>
            <td align="right" width="359"><c:url value="AccountDetails"
                var="urlVariable">
                <c:param name="accountId"
                    value="${varAccounts.accountNumber}"></c:param>
            </c:url> <a href="<c:out value='${urlVariable}'/>">
                <fmt:formatNumber maxFractionDigits="2"
                    minFractionDigits="2" value="${varAccounts.balance}"></fmt:format
            </a></td>
        </tr>
    </c:forEach>
</tbody>
```

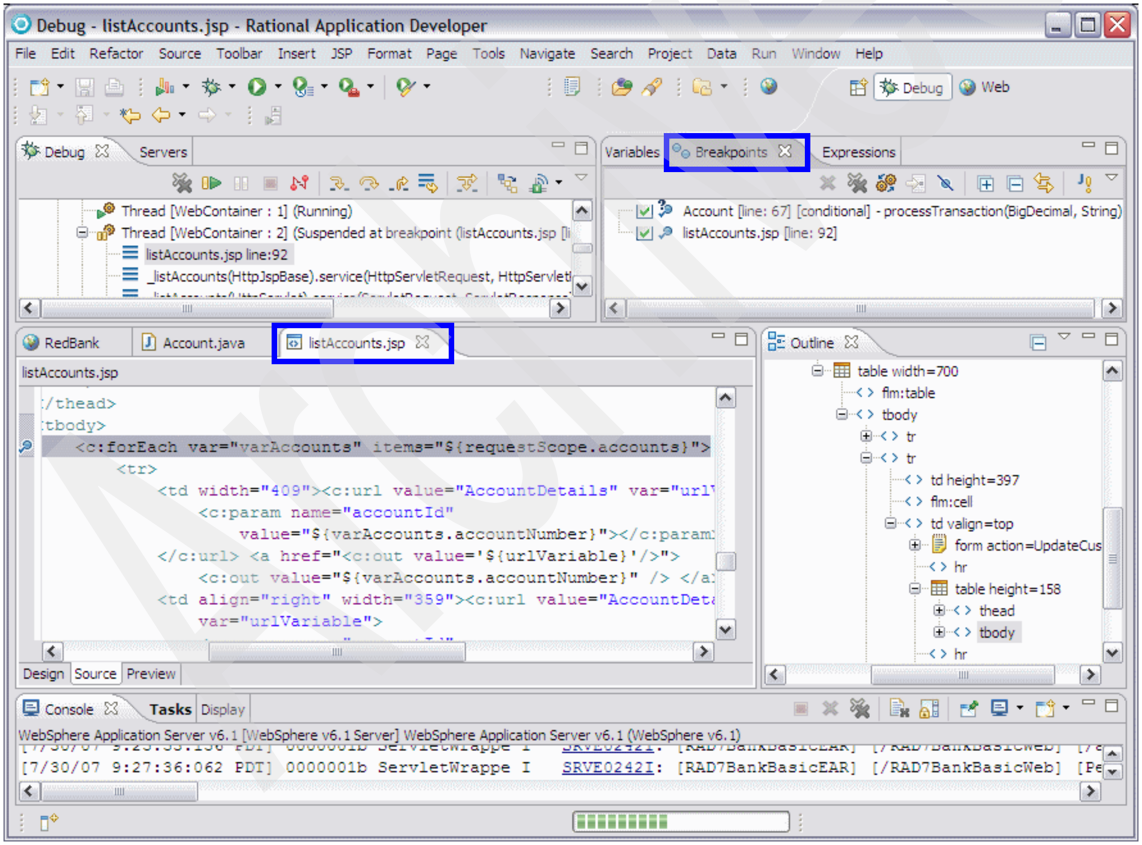
Figure 22-13 Adding a breakpoint to a JSP page

Note that the Breakpoint properties are also available for JSPs from the context menu. These share the same features as Java breakpoints with the exception that content assist is not available in the breakpoint condition field.

Debugging a JSP

When a new breakpoint is added, it is not necessary to redeploy the Web application:

- From the RedBank index page, select the **redbank** tab.
- On the redbank page, enter a Customer ID of 444-44-4444, and click **Submit**. This executes the `listAccounts.jsp` file and hits the new breakpoint.
- In the Confirm Perspective Switch dialog, click **Yes** to switch to the Debug perspective.
- Execution should stop at the breakpoint set in the `listAccounts.jsp`, because clicking **Submit** in the application attempts to display the accounts by executing this JSP. The thread is suspended in debug, but other threads might still be running (Figure 22-14).



```
Thread [WebContainer : 1] (Running)
Thread [WebContainer : 2] (Suspended at breakpoint (listAccounts.jsp [li
    listAccounts.jsp line:92
    _listAccounts(HttpJspBase).service(HttpServletRequest, HttpServlet
```

```
Account [line: 67] [conditional] - processTransaction(BigDecimal, String)
listAccounts.jsp [line: 92]
```

listAccounts.jsp

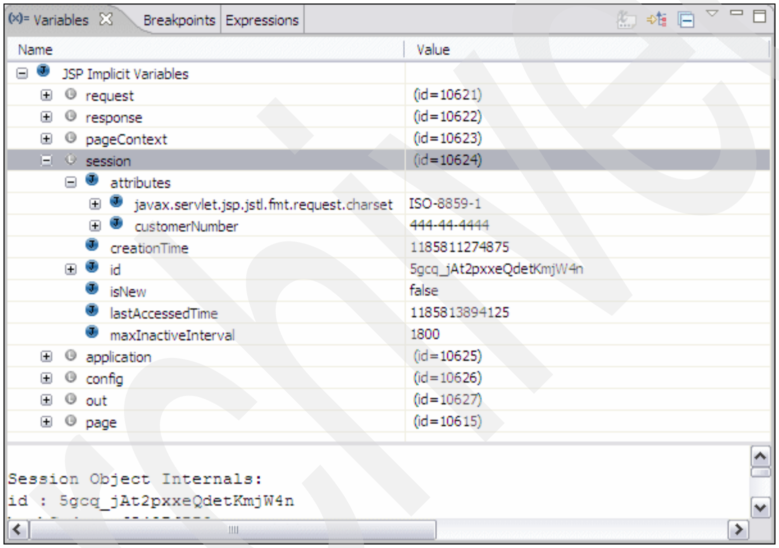
```
:/thead>
:tbody>
   <c:forEach var="varAccounts" items="${requestScope.accounts}">
      <tr>
         <td width="409"><c:url value="AccountDetails" var="url\
            <c:param name="accountId"
               value="${varAccounts.accountNumber}"></c:param
         </c:url> <a href="<c:out value='${urlVariable}'/>">
            <c:out value="${varAccounts.accountNumber}" /> </a
         <td align="right" width="359"><c:url value="AccountDeta
            var="urlVariable">
```

```
WebSphere Application Server v6.1 [WebSphere v6.1 Server] WebSphere Application Server v6.1 (WebSphere v6.1)
[7/30/07 9:27:36:062 PDT] 0000001b ServletWrappe I   SRVE0242I: [RAD7BankBasicEAR] [/RAD7BankBasicWeb] [Pe
```

Figure 22-14 Debugging a JSP

Note: If you have two JSPs with the same name in multiple Web applications, the wrong JSP source might be displayed. Open the correct JSP to see its source code.

- Once a breakpoint is hit, you can analyze variables and step through lines of the JSP code. The same functions available for Java classes are available for JSP debugging. The difference is that the debugger shows the JSP source code and not the generated Java code.
- The JSP variables are shown in the Variables view. Note that the JSP implicit variables are also visible, and it is possible to look at things such as request parameters or session data (see Figure 22-15)



Name	Value
JSP Implicit Variables	
request	(id=10621)
response	(id=10622)
pageContext	(id=10623)
session	(id=10624)
attributes	
javax.servlet.jsp.jstl.fmt.request.charset	ISO-8859-1
customerNumber	444-44-4444
creationTime	1185811274875
id	5gcq_jAt2pxxeQdetKmjW4n
isNew	false
lastAccessedTime	1185813894125
maxInactiveInterval	1800
application	(id=10625)
config	(id=10626)
out	(id=10627)
page	(id=10615)

```
Session Object Internals:
id : 5gcq_jAt2pxxeQdetKmjW4n
```

Figure 22-15 JSP Implicit variables in Variable view

- Step over the lines within the JSP by pressing the F6 key. Note that the debugger will skip any lines with only HTML code.
- Note the change in the variables view. Expand **JSP Implicit Variables** → **pageContext** → **page_attributes** and select **varAccounts**, which is the variable in the `<c:forEach var="varAccounts"...>` loop:

```
Account 004-999000777: --> Current balance: $850.00
   Transactions:
   1. Debit: --> Amount $50.00 on 2007-07-30 09:27:36.078
   2. Debit: --> Amount $50.00 on 2007-07-30 09:44:54.125
```

- The customer and the accounts are visible under **JSP Implicit Variables → pageContext → request_attributes**.
- Click **Resume** (▶) allow the application to continue with the JSP page generation.
- Select the Web Browser page which shows the `ListAccounts`.jsp page. Select an account, then perform a withdraw or deposit and click **Submit**. If the value is over 50, the debugger takes over at the breakpoint in the `Account` class.

You can perform debugging activities on the `Account` class or click the Resume icon to finish the example.

Debugging a Web application on a remote server

It is possible to connect to and debug a Java Web application that has been launched in debug mode on a remote application server. When debugging a remote program the Debug perspective has the same features as when debugging locally—the difference lies in the fact that the application is on a remote JVM and the debugger must attach to the JVM through a configured debug port. The debugging machine must also map the debug information to its locally stored copy of the source code, so it is important that the source code on the debugger machine matches what is deployed.

This following example scenario includes a node where Application Developer V7.0 is installed (Developer node), and a separate node where IBM WebSphere Application Server v6.1 is installed (Application Server node). The developer node attaches to the Application Server Node and controls it through a debugger.

Tip: If you defined a second server profile in (`AppSRV02`) in Chapter 20, “Servers and server configuration” on page 963, then you can use this server for debugging on a remote server.

Exporting the RedBank as an EAR file

This section describes how to export the RedBank to an EAr file so that it can be deployed on a remote WebSphere Application Server.

- In the Project Explorer right-click **RAD7BankBasicEAR** and select **Export → EAR file**.
- In the EAR Export dialog enter the following and then click **Finish**:
  – EAR application: `RAD7BankBasicEAR`
  – Destination: `C:\temp\RAD7BankBasicEAR.ear`

Deploying the RedBank application

The following steps show how to deploy the RedBank application to a remote system where IBM WebSphere Application Server V6.1 has been installed:

- Ensure that the target WebSphere Application Server - server1 application server is started.
- Start the WebSphere Application Server Administrative Console by entering the following in a Web browser and logging on:

```
http://<hostname>:9060/ibm/console   <=== port 9062 in our case
```

> **Note:** For the `<hostname>` it is sufficient to use the IP address of the machine running WebSphere Application Server. Run the command `ipconfig` from the command line to determine this. Note that our remote server runs on a different set of ports.

- From the Administration Console, expand **Applications** and click **Install New Application**.
- Select **Local file system** and click **Browse** and locate the generated EAR file (`C:\temp\RAD7BankBasicEAR.ear`).
- Accept the default options for the other fields an click **Next**.
- In Step 1, select **Precompile JavaServer Pages files**, and click **Next**.
- In Step 2, click **Next**.
- In Step 3 (Summary), accept the defaults and click **Finish**.

  You should see a screen showing the progress of the installation. After a short time, the following message will be displayed if the application is successfully deployed:

```
Application RAD7BankBasicEAR installed successfully.
```

- Click **Save** directly to the master configuration link URL.
- Navigate to the **Applications** → **Enterprise Application**. Select **RAD7BankBasicEAR** and click **Start**.
- Click **Logout**.
- Verify the application is working properly by opening a Browser and navigating to the following URL of the target machine:

```
http://<hostname or IPaddress>:9080/RAD7BankBasicWeb/
```

> **Note:** If the target system is running WebSphere Application Server v5.1 or V5.0 and debugging is required, it is necessary to install the IBM Rational Agent Controller. See the online help for details on how to do this.

Configuring debug on a remote WebSphere Application Server

The following steps explain how to configure WebSphere Application Server V6.1 to start in debug mode:

- If it is not already running, start the application server.

```
<was_home>\bin\startServer.bat server1
```

- Start the WebSphere Administrative Console by entering the following in a Web browser and then logging in:

```
http://<hostname>:9060/ibm/console  <=== port 9062 in our case
```

- In the left-hand frame expand **Servers** → **Application Servers**.
- In the Application Servers page, click **server1**.
- On the Configuration tab, select **Debugging Service** in the Additional Properties section to open the Debugging Service configuration page.
- In the General Properties section of the Configuration tab, select **Enable service at startup**. This enables the debugging service when the server starts.

> **Note:** The value of the JVM debug port is needed when connecting to the application server with the debugger. The default value is **7777**; in our case the server used port **7779**.

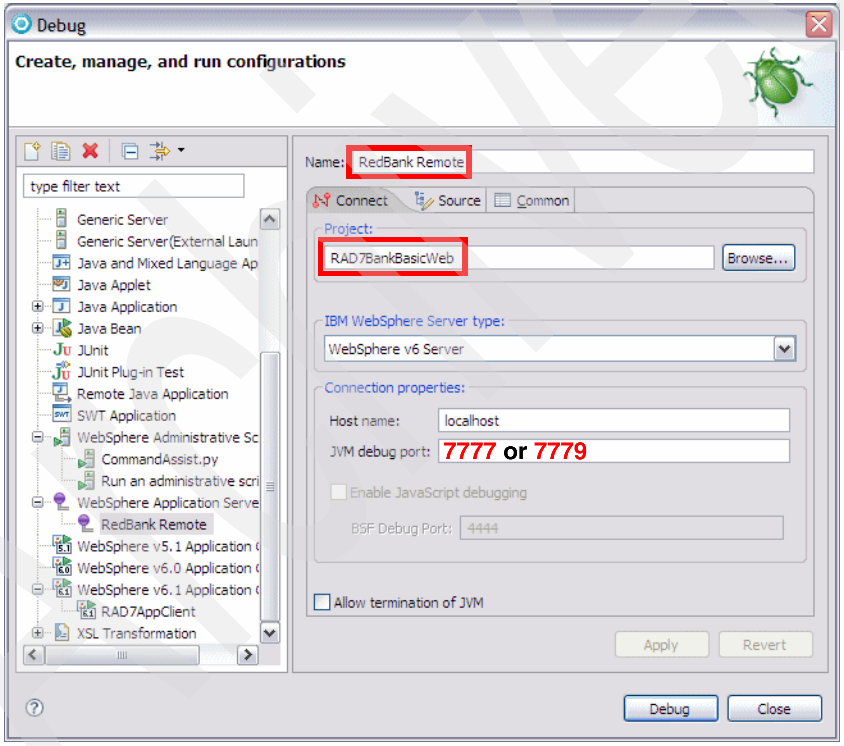
- Click **OK** to make the changes to your local configuration.
- Save the configuration changes.
- Click **Logout**.
- You must restart the application server before the changes that have been made take effect.
- Again, verify the application is working properly by navigating to the following URL:

```
http://<hostname>:9080/RAD7BankBasicWeb/  <=== port 9081 in our case
```

Attaching to the remote server in Application Developer

Assuming that the target server is running in debug mode, complete the following steps to attach to the remote WebSphere Application Server V6.1 from within Application Developer V7.0. Note that the workspace used must contain the `RAD7BankBasicWeb` project.

- In the Project Explorer right-click **RAD7BankBasicWeb**, and select **Debug As** → **Debug**.
- Create a new remote Debug configuration for WebSphere Application Server V6.1 server.
  - On the **Create, Manage, and run configurations** page, right-click **WebSphere Application Server** and select **New**.
  - Enter a meaningful name for this Debug configuration (`RedBank Remote`).
  - In the **Connect** tab, make sure the project is `RAD7BankBasicWeb`, select `WebSphere v6 Server` for the IBM WebSphere Server Type, enter the IP address or target machine name on the **Host Name** field and 7777 as the **port number**.
  - Click **Apply** (Figure 22-16).


Debug
Create, manage, and run configurations
type filter text
Generic Server
Generic Server(External Laun
Java and Mixed Language Ap
Java Applet
Java Application
Java Bean
JUnit
JUnit Plug-in Test
Remote Java Application
SWT Application
WebSphere Administrative Sc
CommandAssist.py
Run an administrative scri
WebSphere Application Serve
RedBank Remote
WebSphere v5.1 Application
WebSphere v6.0 Application
WebSphere v6.1 Application
RAD7AppClient
XSL Transformation
Connect
Source
Common
Browse...
Apply
Revert
Debug
Close

Name: RedBank Remote

Project: RAD7BankBasicWeb

IBM WebSphere Server type: WebSphere v6 Server

Connection properties:

Host name: localhost

JVM debug port: 7777 or 7779

□ Enable JavaScript debugging

BSF Debug Port: 4444

□ Allow termination of JVM

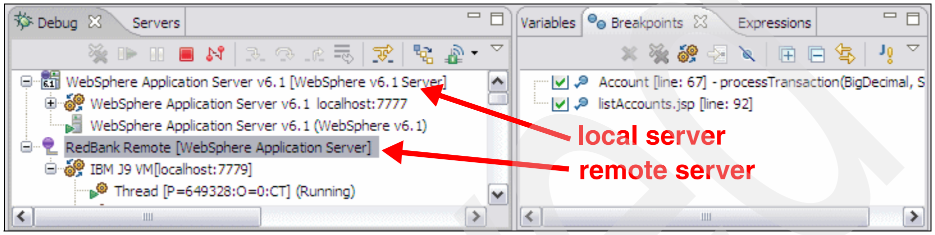
Figure 22-16 Debug config for Remote debugging

  - On the **Source** tab expand the Default folder. Note that about halfway down is the `RAD7BasicBankWeb` project. This lets the debugger know where the source code is.

– On the **Common** tab are standard options for debug configuration, including where to save the configuration and where to output the SystemOut file.

– Click **Debug** to attach the debugger to the remote server.

The debug perspective now shows the Remote debugger running in the Debug view, as shown in Figure 22-17. The debugger is waiting for a breakpoint to be triggered. Note that clicking the Disconnect icon ( ) stops the debugging.



WebSphere Application Server v6.1 [WebSphere v6.1 Server]
WebSphere Application Server v6.1 localhost:7777
WebSphere Application Server v6.1 (WebSphere v6.1)
RedBank Remote [WebSphere Application Server]
IBM J9 VM[localhost:7779]
Thread [P=649328:O=0:CT] (Running)

Account [line: 67] - processTransaction(BigDecimal, S
listAccounts.jsp [line: 92]

Figure 22-17 Debugging perspective while remote debugging

Note: When attaching to a local Application Developer's WebSphere instance, you must start the application server in debug mode and then open the Debug perspective and disconnect the debug instance started by Application Developer automatically when the server was started in debug mode.

After that you can start a remote debug instance using **localhost** as the host name and port **7777** (or the correct port, such as **7779**). This will attach to the test application server and allow the user to perform all the usual debugging facilities.

Debugging a remote application

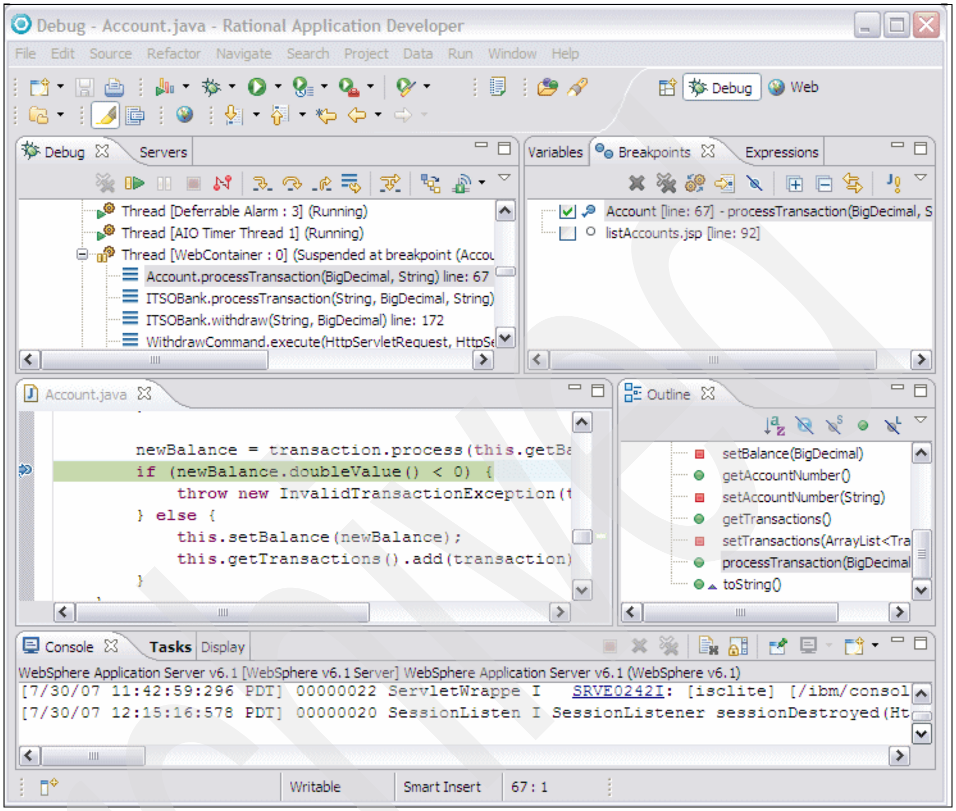
From a Web browser, navigate to the URL where hostname is the IP address or name of the target machine:

```
http://<hostname>:9080/RAD7BankBasicWeb/  <=== port 9081 in our case
```

From the main index page, select the **redbank** tab, enter `444-44-4444` as the customer ID and click **Submit.** In the List Accounts page, select an account number and in the Account Details page select **Withdraw** and enter 100 in the amount field. Click **Submit.**

This causes Application Developer to prompt the user to switch to the Debug perspective (if not already there).

You can now debug the remotely running application in the same way as a locally deployed application (Figure 22-18).


Debug - Account.java - Rational Application Developer
File Edit Source Refactor Navigate Search Project Data Run Window Help
Debug
Web
Debug
Servers
Variables
Breakpoints
Expressions
☑ Account [line: 67] - processTransaction(BigDecimal, S
□ listAccounts.jsp [line: 92]
Account.java
Outline
setBalance(BigDecimal)
getAccountNumber()
setAccountNumber(String)
getTransactions()
setTransactions(ArrayList<Tra
processTransaction(BigDecimal
toString()
Console
Tasks
Display
Writable
Smart Insert
67 : 1

```
Thread [Deferrable Alarm : 3] (Running)
Thread [AIO Timer Thread 1] (Running)
Thread [WebContainer : 0] (Suspended at breakpoint (Accou
    Account.processTransaction(BigDecimal, String) line: 67
    ITSOBank.processTransaction(String, BigDecimal, String)
    ITSOBank.withdraw(String, BigDecimal) line: 172
    WithdrawCommand.execute(HttpServletRequest, HttpSe
```

```
        newBalance = transaction.process(this.getBa
        if (newBalance.doubleValue() < 0) {
            throw new InvalidTransactionException(t
        } else {
            this.setBalance(newBalance);
            this.getTransactions().add(transaction)
        }
```

```
WebSphere Application Server v6.1 [WebSphere v6.1 Server] WebSphere Application Server v6.1 (WebSphere v6.1)
[7/30/07 11:42:59:296 PDT] 00000022 ServletWrappe I   SRVE0242I: [isclite] [/ibm/consol
[7/30/07 12:15:16:578 PDT] 00000020 SessionListen I SessionListener sessionDestroyed(Ht
```

Figure 22-18 Debug perspective after being triggered by a remote application breakpoint

Also note that the browser where the request was performed is frozen, waiting a response from the server.

In the Debug perspective, click **Resume** ( ) and the transaction completes and the Account Details page is displayed in the browser.

Highlight the remote debugging instance and click **Disconnect** to finish the debugging session.

Uninstalling the remote application

You might want to remove the RAD7BankBasicEAR application from the remote server. In the administrative console under **Applications → Enterprise Applications**, select **RAD7BankBasicEAR** and click **Uninstall**. Then save the configuration. Optionally, stop the remote server.

Jython debugger

The Jython debugger enables you to detect and diagnose errors in Jython script that is used for WebSphere Application Server administration. With the debugger, you can control the execution of your code by setting line breakpoints, suspending execution, stepping through your code, and examining the contents of variables. (Note that variable values cannot be changed in Jython.)

The Jython debugger only supports debugging of script that are running on WebSphere Application Server Version 6.1.

You can debug a Jython script that has been developed or imported into a Jython project. When you are debugging a Jython script, you can set line breakpoints.

When the workbench is running the script and encounters a breakpoint, the script temporarily stops running. Execution suspends at the breakpoint before the script is executed, at which point you can check the contents of variables. You can then step over (execute) and see what effect the statement has on the script.

Tip: To debug a Jython script, the server does not have to run in Debug mode.

Debugging a sample Jython script

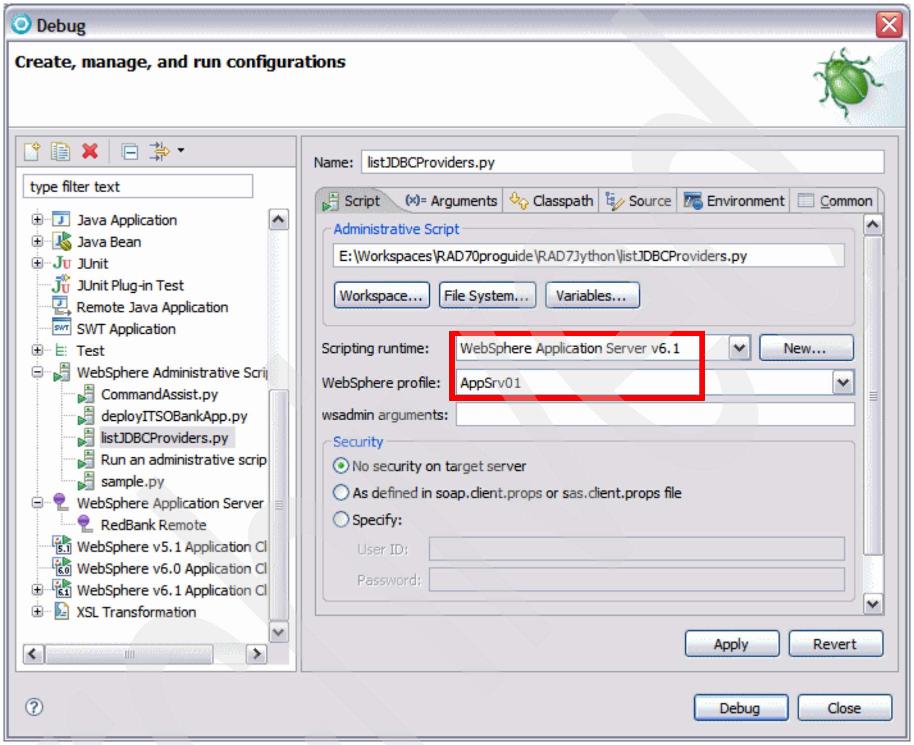
In this section we debug the **listJDBCProviders** script that was described in "Developing automation scripts" on page 993:

- Open the listJDBCProviders.py Jython script file in the RAD7Jython project.
- Set a breakpoint in the showJdbcProviders function at the line:

  ```
  for provider in providerEntryList
  ```

- Right-click **listJDBCProviders.py** and select **Debug As → Administrative Script**.

- In the debug configuration page (Figure 22-19):
  - Select **WebSphere Application Server v6.1** as Scripting runtime and **AppSrv01** as the WebSphere profile.
  - Click **Apply**, then click **Debug**.



Name: listJDBCProviders.py

E:\Workspaces\RAD70proguide\RAD7Jython\listJDBCProviders.py

Scripting runtime: WebSphere Application Server v6.1

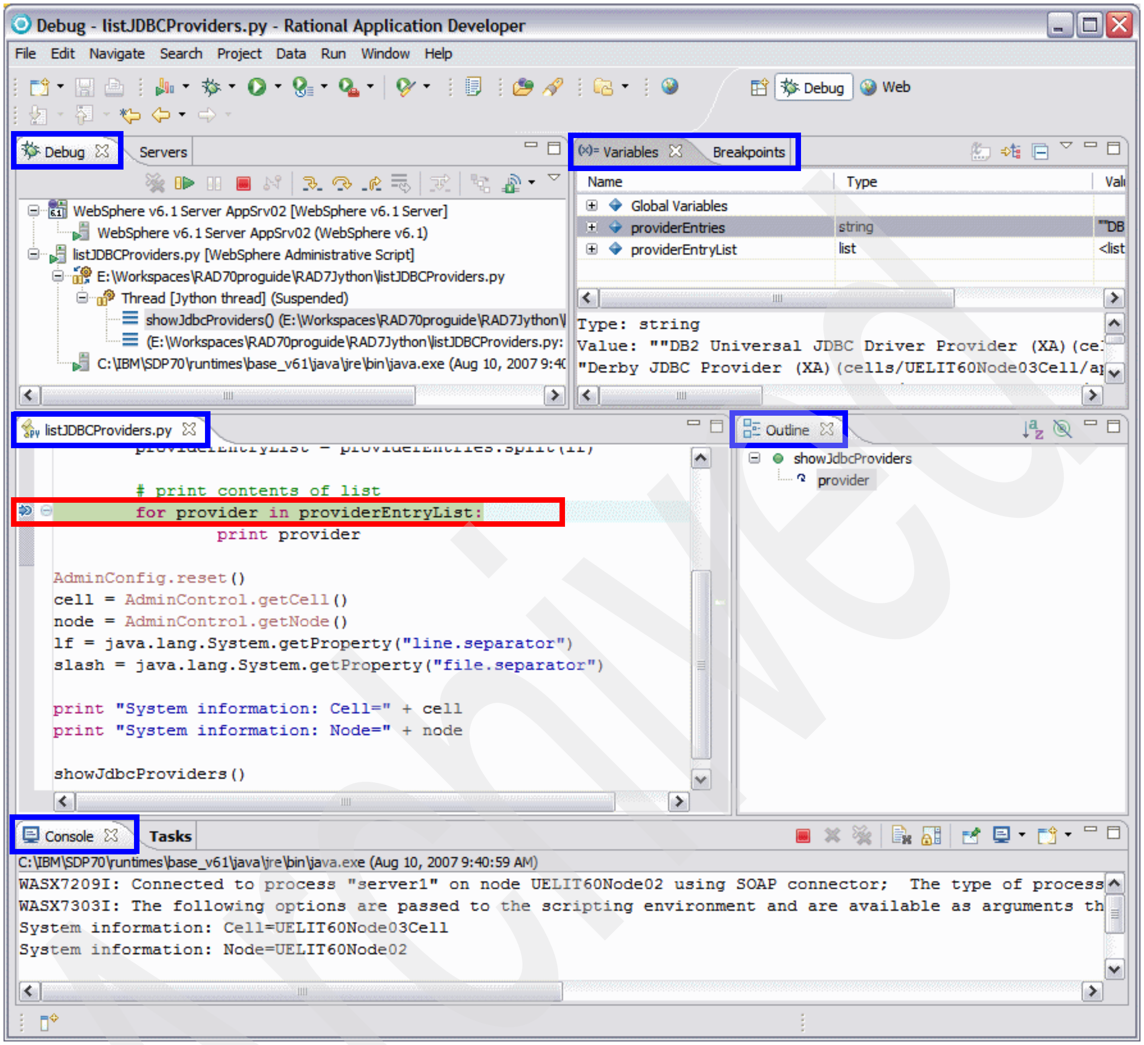
WebSphere profile: AppSrv01

wsadmin arguments:

Security:
- ● No security on target server
- ○ As defined in soap.client.props or sas.client.props file
- ○ Specify:

Figure 22-19 Jython debugging configuration

- Execution of the script starts and when the breakpoint is encountered execution is suspended.
- When prompted, switch to the Debug perspective.
- The Debug perspective opens and displays the familiar views (Figure 22-20):
  - The Debug view shows the thread and is used to step through the code.
  - The editor shows the source code and where we currently are.
  - The Variables view shows the Jython variables, which cannot be changed.
  - The Breakpoints view shows the breakpoints.
  - The Outline view shows the outline of the script.
  - The Console shows the output of the script.



Debug - listJDBCProviders.py - Rational Application Developer
File Edit Navigate Search Project Data Run Window Help
Debug
Web
Debug
Servers
Variables
Breakpoints
listJDBCProviders.py
Outline
Console
Tasks

WebSphere v6.1 Server AppSrv02 [WebSphere v6.1 Server]
WebSphere v6.1 Server AppSrv02 (WebSphere v6.1)
listJDBCProviders.py [WebSphere Administrative Script]
E:\Workspaces\RAD70proguide\RAD7Jython\listJDBCProviders.py
Thread [Jython thread] (Suspended)
showJdbcProviders() (E:\Workspaces\RAD70proguide\RAD7Jython\
(E:\Workspaces\RAD70proguide\RAD7Jython\listJDBCProviders.py:
C:\IBM\SDP70\runtimes\base_v61\java\jre\bin\java.exe (Aug 10, 2007 9:40

Name	Type	Val
Global Variables		
providerEntries	string	""DB
providerEntryList	list	<list

```
Type: string
Value: ""DB2 Universal JDBC Driver Provider (XA)(ce
"Derby JDBC Provider (XA)(cells/UELIT60Node03Cell/ap
```

showJdbcProviders
provider

```
    providerEntryList = providerEntries.split(lf)

    # print contents of list
    for provider in providerEntryList:
        print provider

AdminConfig.reset()
cell = AdminControl.getCell()
node = AdminControl.getNode()
lf = java.lang.System.getProperty("line.separator")
slash = java.lang.System.getProperty("file.separator")

print "System information: Cell=" + cell
print "System information: Node=" + node

showJdbcProviders()
```

C:\IBM\SDP70\runtimes\base_v61\java\jre\bin\java.exe (Aug 10, 2007 9:40:59 AM)

```
WASX7209I: Connected to process "server1" on node UELIT60Node02 using SOAP connector;  The type of process
WASX7303I: The following options are passed to the scripting environment and are available as arguments th
System information: Cell=UELIT60Node03Cell
System information: Node=UELIT60Node02
```

Figure 22-20 Debug perspective when debugging a Jython script

- Step through the Jython code and watch the variables.

The Jython debugger is very useful when you encounter errors in your Jython scripts. Run the script in debug mode, without having to restart the server.

More information

The online help provided with Application Developer has a large section on debugging applications and is a good starting point. It has information on the following topics:

- Java debugging (including local and remote Web Applications and local and remote Java applications).
- SQLJ debugging
- DB2 stored procedure debugging
- Jython script debugging
- XSLT debugging
- JavaScript debugging

For debugging EGL applications, refer to Chapter 15, “Develop applications using EGL” on page 665.

Finally, IBM developerWorks has a number of tutorials and articles to explain debugging for various situations, including the following topics:

- Debugging and Testing Java Applications:

  `http://www.ibm.com/developerworks/edu/i-dw-r-radcert2556.html`

- Getting Started with the New Rational Application Developer XSLT Debugger:

  `http://www-128.ibm.com/developerworks/rational/library/05/614_debug/`

Part 4

Deploy and profile applications

In this part of the book, we describe the tooling and technologies provided by Application Developer for automatic builds, deployment, and profiling.

23

Build applications with Ant

Traditionally, application builds are performed by using UNIX/Linux shell scripts or Windows batch files in combination with tools such as *make*. While these approaches are still valid, new challenges exist when developing Java applications, especially in a heterogeneous environment. Traditional tools are limited in that they are closely coupled to a particular operating system. With Ant, you can overcome these limitations and perform the build process in a standardized fashion regardless of the platform.

This chapter provides an introduction to the concepts and features of Ant within IBM Rational Application Developer V7.0. The focus of the chapter is to demonstrate how to use the Ant tooling included in Rational Application Developer to build projects (applications).

The chapter is organized into the following sections:

- Introduction to Ant
- Ant features in Application Developer
- Building a simple Java application
- Building a J2EE application
- Running Ant outside of Application Developer

Introduction to Ant

Ant is a Java-based, platform-independent, open source build tool. It was formerly a sub-project in the Apache Jakarta project, but in November 2002 it was migrated to an Apache top-level project. Ant's function is similar to the *make* tool. Since it is Java-based and does not make use of any operating system-specific functions, it is platform independent, thus allowing you to build your projects using the same build script on any Java-enabled platform.

The Ant build operations are controlled by the contents of the XML-based script file. This file not only defines what operations to perform, but also defines the order in which they should be performed, and any dependencies between them.

Ant comes with a large number of built-in tasks sufficient to perform many common build operations. However, if the tasks included are not sufficient, you also have the ability to extend Ant's functionality by using Java to develop your own specialized tasks. These tasks can then be plugged into Ant.

Not only can Ant be used to *build* your applications, it can also be used for many other operations such as retrieving source files from a version control system, storing the result back in the version control system, transferring the build output to other machines, deploying the applications, generating Javadoc, and sending messages when a build is finished.

Ant build files

Ant uses XML *build files* to define what operations must be performed to build a project. Here is a list of the main components of a build file:

- **Project**: A build file contains build information for a single project. It can contain one or more *targets*.
- **Target**: A target describes the *tasks* that must be performed to satisfy a goal. For example, compiling source code into class files might be one target, and packaging the class files into a JAR file might be another target.

  Targets can depend upon other targets. For example, the class files must be up-to-date before you can create the JAR file. Ant can resolve these dependencies.
- **Task**: A task is a single step that must be performed to satisfy a target. Tasks are implemented as Java classes that are invoked by Ant, passing parameters defined as attributes in the XML. Ant provides a set of standard tasks (core tasks), a set of optional tasks, and an API, which allows you to write your own tasks.

- **Property**: A property has a name and a value pair. Properties are essentially variables that can be passed to tasks through task attributes. Property values can be set inside a build file, or obtained externally from a properties file or from the command line. A property is referenced by enclosing the property name inside `${}`, for example `${basedir}`.
- **Path**: A path is a set of directories or files. Paths can be defined once and referred to multiple times, easing the development and maintenance of build files. For example, a Java compilation task can use a path reference to determine the classpath to use.

Ant tasks

A comprehensive set of built-in tasks is supplied with the Ant distribution. The tasks that we use in our example are as follows:

- delete: Deletes files and directories
- echo: Outputs messages
- jar: Creates Java archive files
- javac: Compiles Java source
- mkdir: Creates directories
- tstamp: Sets properties containing date and time information

To find out more about Ant, visit the Ant Web site at:

`http://ant.apache.org/`

This chapter provides a basic outline of the features and capabilities of Ant. For complete information, you should consult the Ant documentation included in the Ant distribution or available on the Internet at:

`http://ant.apache.org/manual/index.html`

Note: IBM Rational Application Developer V7.0 includes Ant V1.6.5.

Ant features in Application Developer

Application Developer includes the following features to aid in the development and use of Ant scripts:

- It provides the ability to create and run Ant buildfiles in the Workbench and run the build process in a background task like other tasks within Application Developer.
- The Ant editor also offers content assist (including Ant specific templates) with the ability to insert snippets and syntax highlighting.

- The Ant editor has a format function that will allow you to format your Ant files base on your preferences.
- It provides the ability to add new Ant task and types that will be available for build files.
- A Problems view is available in the Ant editor to highlight syntax errors in your Ant files.

In this section, we highlight the following Ant-related features in Application Developer:

- Content assist
- Code snippets
- Formatting an Ant script
- Defining the format of an Ant script
- Problems view

Preparing for the sample

To demonstrate the basic concepts of Ant, we provide a very simple Java application named `HelloAnt`, which prints a message to the console.

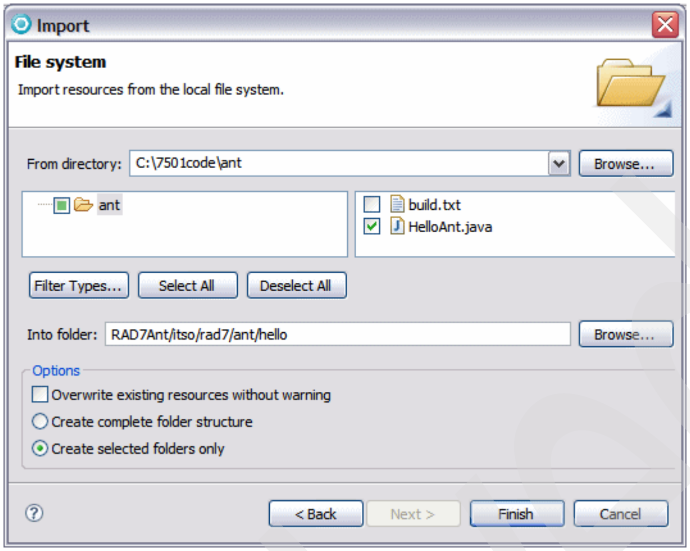
We use a simple Java project (`RAD7Ant`) and class (`HelloAnt`) for this example.

To create a new Java project, do these steps:

- In the Workbench, select **File** → **New** → **Project**.
- In the New Project dialog, select **Java** → **Java Project** and click **Next**.
- When prompted for the project name, enter **`RAD7Ant`** for the project name field, and click **Finish**.

To import the `HelloAnt` class into the `RAD7Ant` Java project, do these steps:

- Right-click on the **RAD7Ant** project and select **New** → **Package**.
- In the New Package dialog, enter `itso.rad7.ant.hello` for the Name field.
- Right-click on the `itso.rad7.ant.hello` package and select **Import**.
- In the Import dialog, select **General** → **File System,** and click **Next**.
- In the File System dialog, select `c:\7501code\ant\` as directory and select `HelloAnt.java` (Figure 23-1).
- Click **Finish**.



Import

File system

Import resources from the local file system.

From directory: C:\7501code\ant

▣ ant

□ build.txt

☑ HelloAnt.java

Into folder: RAD7Ant/itso/rad7/ant/hello

Options

□ Overwrite existing resources without warning

○ Create complete folder structure

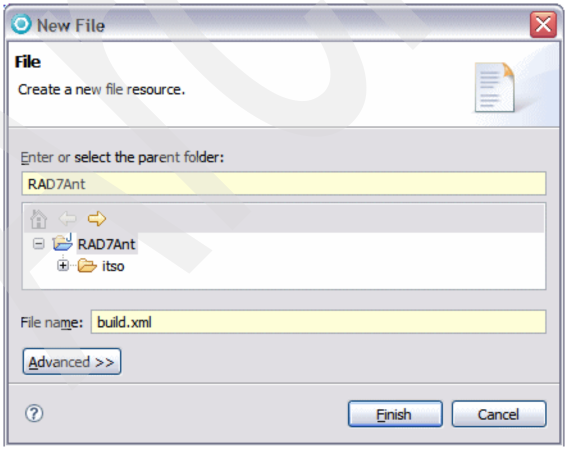
● Create selected folders only

Figure 23-1 Import a Java class

Creating a build file

To create the simple build file, do these steps:

- Right-click the **RAD7Ant** project. and select **New** → **File**.
- In the New File dialog, enter `build.xml` as the file name, and click **Finish** (Figure 23-2).



New File

File

Create a new file resource.

Enter or select the parent folder:

RAD7Ant

⊟ RAD7Ant

⊞ itso

File name: build.xml

Figure 23-2 Create the build.xml file

Note: Application Developer has the ability to link to external files on the file system. The Advance button on the New File dialog allows you to specify the location on the file system that the new file is linked to.

- Double-click the **build.xml** file to open it in the Ant editor. Copy and paste the text in `c:\7501code\ant\build.txt` into the `build.xml` file.

We now walk you through the various sections of this file, and provide an explanation for each of them.

Project definition

The <project> tag in the `build.xml` file defines the project name and the default target. The project name is an arbitrary name; it is not related to any project name in your Application Developer workspace.

The project tag also sets the working directory for the Ant script. All references to directories throughout the script file are based on this directory. A dot (.) means to use the current directory, which, in Application Developer, is the directory where the `build.xml` file resides.

Global properties

Properties that will be referenced throughout the whole script file can be placed at the beginning of the Ant script. Here we define the property `build.compiler` that tells the **`javac`** command what compiler to use. We will tell it to use the Eclipse compiler.

We also define the names for the source directory, the build directory, and the distribute directory. The source directory is where the Java source files reside. The build directory is where the class files end up, and the distribute directory is where the resulting JAR file is placed:

- We define the source property as ".", which means it is the same directory as the base directory specified in the project definition above.
- The build and distribute directories will be created as `c:\temp\build` and `c:\temp\RAD7Ant` directories.

Properties can be set as shown below, but Ant can also read properties from standard Java properties files or use parameters passed as arguments on the command line:

```
<!-- set global properties for this build -->
<property name="build.compiler"
      value="org.eclipse.jdt.core.JDTCompilerAdapter"/>
<property name="source" value="."/>
<property name="build" value="c:\temp\build"/>
<property name="distribute"  value="c:\temp\RAD7Ant"/>
<property name="outFile" value="helloant"/>
```

Build targets

The build file contains four build targets:

- init
- compile
- dist
- clean

Initialization target (init)

The first target we describe is the init target. All other targets (except clean) in the build file depend upon this target. In the init target, we execute the tstamp task to set up properties that include timestamp information. These properties are then available throughout the whole build. We also create a build directory defined by the build property.

```
<target name="init">
   <!-- Create the time stamp -->
   <tstamp/>
   <!-- Create the build directory structure used by compile -->
   <mkdir dir="${build}"/>
</target>
```

Compilation target (compile)

The compile target compiles the Java source files in the source directory and places the resulting class files in the build directory.

```
<target name="compile" depends="init">
   <!-- Compile the java code from ${source} into ${build} -->
   <javac srcdir="${source}" destdir="${build}"/>
</target>
```

With these parameters, if the compiled code in the build directory is up-to-date (each class file has a timestamp later than the corresponding Java file in the source directory), the source will not be recompiled.

Distribution target (dist)

The distribution target creates a JAR file that contains the compiled class files from the build directory and places it in the lib directory under the dist directory. Because the distribution target depends on the compile target, the compile target must have executed successfully before the distribution target is run.

```
<target name="dist" depends="compile">
   <!-- Create the distribution directory -->
   <mkdir dir="${distribute}/lib"/>

   <!-- Put everything in ${build} into the output JAR file -->
   <!-- We add a time stamp to the filename as well -->
   <jar jarfile="${distribute}/lib/${outFile}-${DSTAMP}.jar"
                  basedir="${build}">
      <manifest>
         <attribute name="Main-Class"
                     value="itso.rad7.ant.hello.HelloAnt"/>
      </manifest>
   </jar>
</target>
```

Cleanup target (clean)

The last of our standard targets is the cleanup target. This target removes the build and distribute directories, which means that a full recompile is always performed if this target has been executed.

```
<target name="clean">
   <!-- Delete the ${build} and ${distribute} directory trees -->
   <delete dir="${build}"/>
   <delete dir="${distribute}"/>
</target>
```

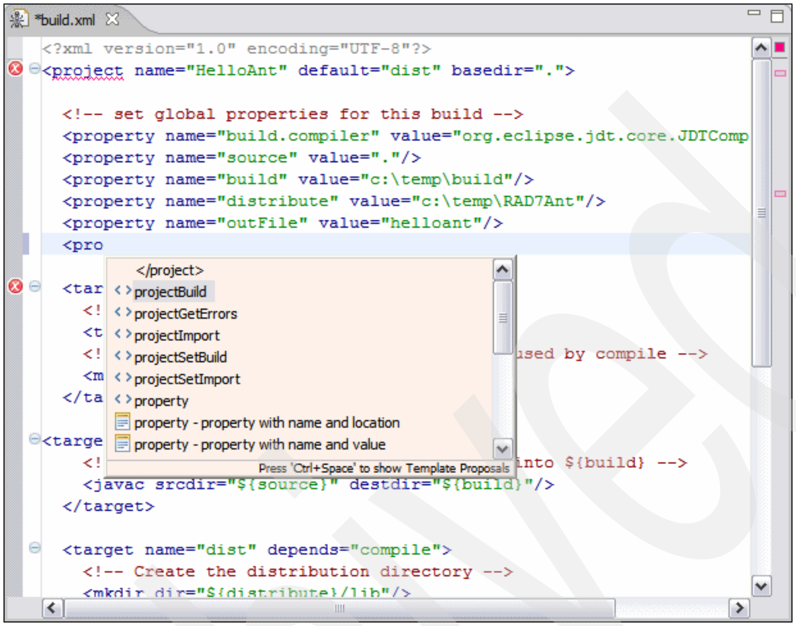
Note that the build.xml file does not call for this target to be executed. It has to be specified when running Ant.

Content assist

To access the content assist feature in the Ant editor, do these steps:

- Open the Java perspective.
- Expand the **RAD7Ant** project.
- Double-click **build.xml** to open the file in an editor.
- Place the cursor in the file and enter <prop, and then press Ctrl+Spacebar.

- The content assist dialog is presented (Figure 23-3). You can then use the up and down arrow keys to select the tag that you want.



```
<?xml version="1.0" encoding="UTF-8"?>
<project name="HelloAnt" default="dist" basedir=".">

  <!-- set global properties for this build -->
  <property name="build.compiler" value="org.eclipse.jdt.core.JDTComp
  <property name="source" value="."/>
  <property name="build" value="c:\temp\build"/>
  <property name="distribute" value="c:\temp\RAD7Ant"/>
  <property name="outFile" value="helloant"/>
  <pro

  <tar
    <!
    <t
    <!                                            used by compile -->
    <m
  </ta

<targe
    <!                                            into ${build} -->
    <javac srcdir="${source}" destdir="${build}"/>
  </target>

  <target name="dist" depends="compile">
    <!-- Create the distribution directory -->
    <mkdir dir="${distribute}/lib"/>
```

```
</project>
projectBuild
projectGetErrors
projectImport
projectSetBuild
projectSetImport
property
property - property with name and location
property - property with name and value
Press 'Ctrl+Space' to show Template Proposals
```

Figure 23-3 Content assist in Ant editor

Code snippets

Application Developer V7 provides the ability to create code snippets that contain commonly used code to be inserted into files rather than typing the code in every time.

To create code snippets, do these steps:

- Open the Snippets view by selecting **Window** → **Show View** → **Other**, and the Show View dialog is displayed.
- Expand the **General** folder, select the **Snippets** view and click **OK** (Figure 23-4).

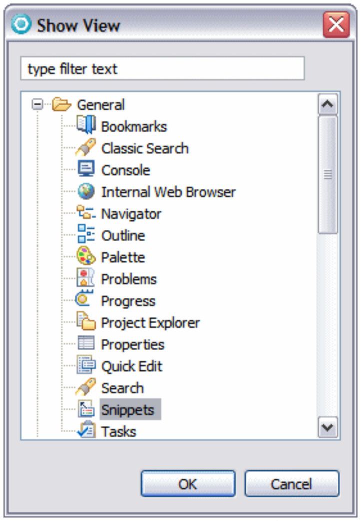


Figure 23-4 Show View dialog

- Right-click the Snippets view and select **Customize** (Figure 23-5).

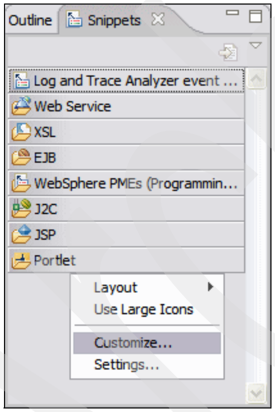
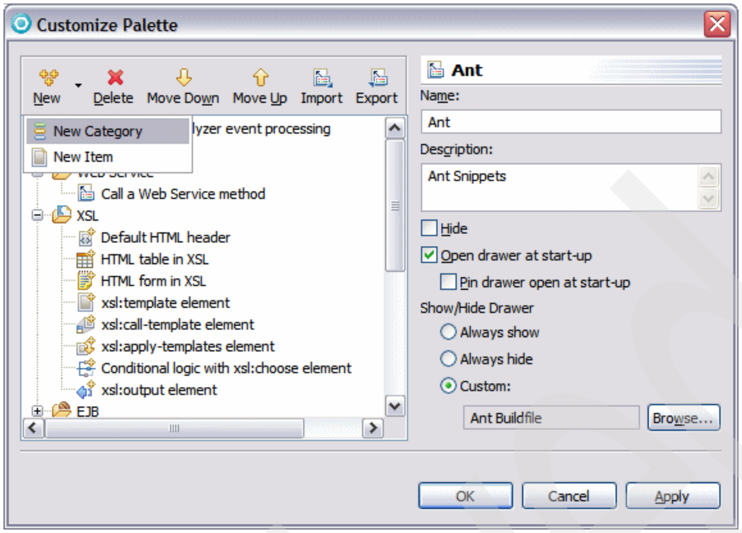


Figure 23-5 Customizing snippets

- In the Customize Palette dialog, select **New** → **New Category**.
- In the New Customize Palette dialog (Figure 23-6), do these steps:
  - Name: `Ant`
  - Description: `Ant Snippets`
  - Select **Custom**.



Ant

Name: Ant

Description: Ant Snippets

☐ Hide

☑ Open drawer at start-up

☐ Pin drawer open at start-up

Show/Hide Drawer

○ Always show

○ Always hide

● Custom: Ant Buildfile

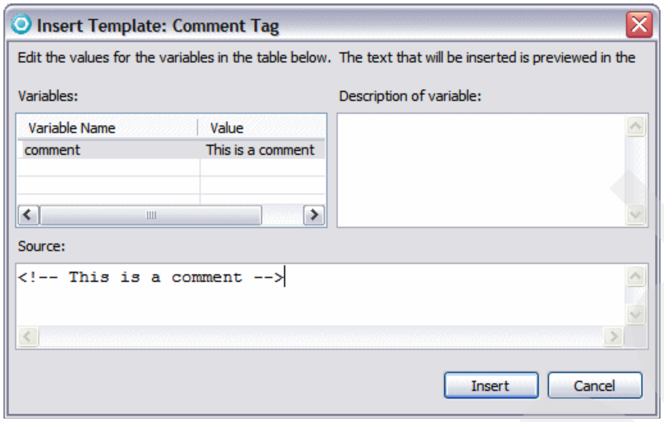
Figure 23-6 New Customize Palette dialog

- Click **Browse** next to Custom, select **Ant Buildfiles** for Content Type Selection, and click **OK** to return to the Customize Palette dialog.
- In the Customize Palette dialog, select **New** → **New Item**.
- In the Unnamed Template dialog, enter the following items:
  - Name: `Comment Tag`
  - Click **New** in the variables section.
  - Variable Name: `comment`
  - Template Pattern: `<!-- ${comment} -->`
- Click **OK** in the Customize Palette dialog.

Use the code snippet

Now that you have created a code snippet, you can use it in any Ant build file. To use a code snippet, do these steps:

- Double-click the **build.xml** file to open it in the editor.
- Place the cursor under the `<project>` tag, double-click the **Comment Tag** in the Snippets view, and the Insert Template dialog is displayed (Figure 23-7).



Edit the values for the variables in the table below. The text that will be inserted is previewed in the

Variable Name	Value
comment	This is a comment

```
<!-- This is a comment -->
```

Figure 23-7 Insert Template dialog

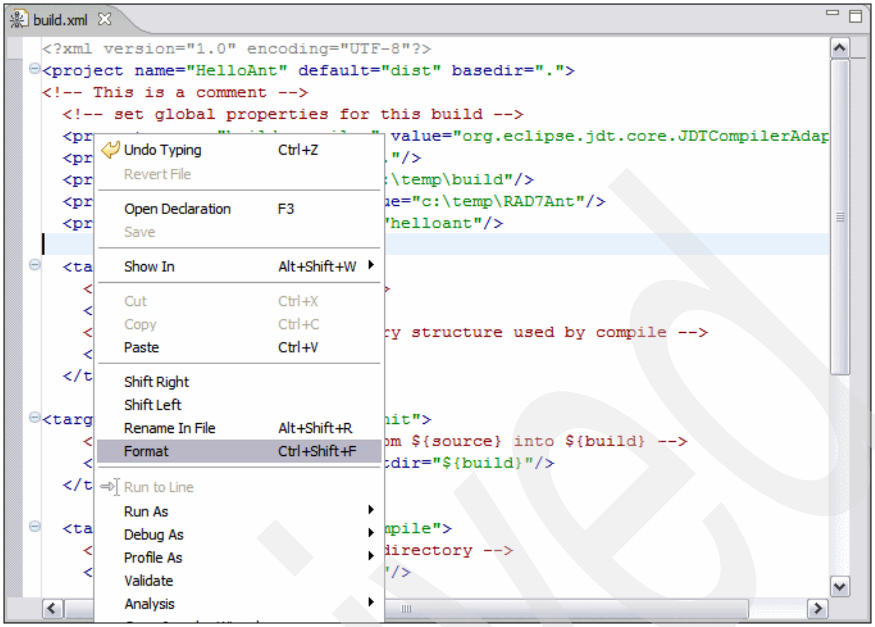
- In the variables table, enter `This is a comment` in the comment variable.
- Click **Insert**.
- The comment line is inserted. Save the file.

Formatting an Ant script

Application Developer offers you the ability to format Ant scripts in the Ant editor. To format the Ant script, do these steps:

- Double-click **build.xml** to open it in the Ant editor.
- Right-click the editor and select **Format** (Figure 23-8).

  Alternatively, you can press `Ctrl+Shift+F`.



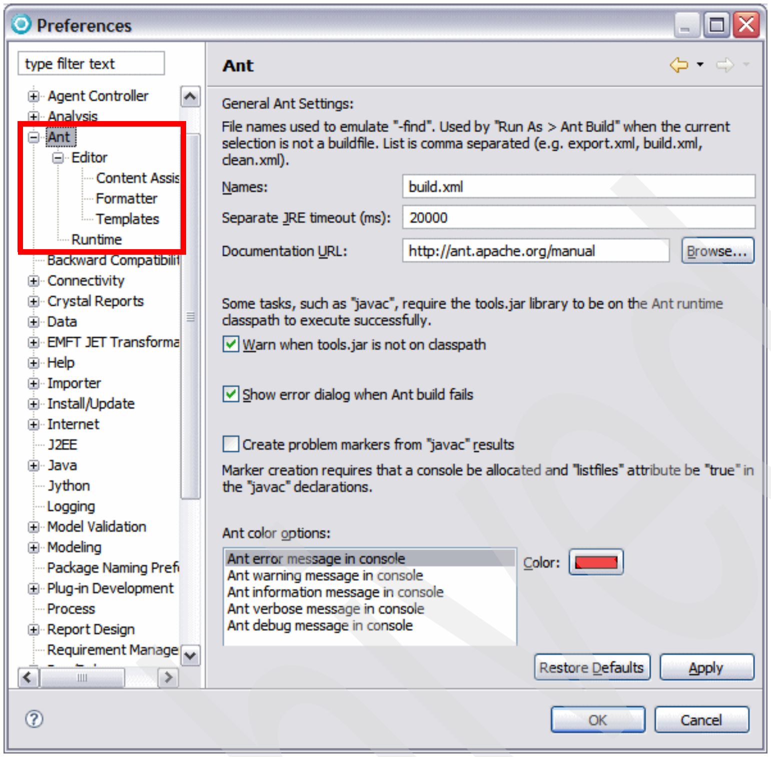
```
<?xml version="1.0" encoding="UTF-8"?>
<project name="HelloAnt" default="dist" basedir=".">
<!-- This is a comment -->
  <!-- set global properties for this build -->
  <pr                                  value="org.eclipse.jdt.core.JDTCompilerAdap
  <pr                                 ."/>
  <pr                                 :\temp\build"/>
  <pr                                 e="c:\temp\RAD7Ant"/>
  <pr                                 'helloant"/>

  <ta
    <                                 >
    <
    <                                 y structure used by compile -->
    <
  </t

<targ                                 it">
    <                                 om ${source} into ${build} -->
    <                                 dir="${build}"/>
  </t

  <ta                                 mpile">
    <                                 directory -->
    <                                 '/>
```

Figure 23-8 Formatting the Ant file

Defining the format of an Ant script

To define the format of an Ant script, do these steps:

- Select **Window** → **Preferences**.
- In the Preferences dialog, select **Ant**.
- In the Ant preferences dialog, you can specify the console colors (Figure 23-9).



General Ant Settings:

File names used to emulate "-find". Used by "Run As > Ant Build" when the current selection is not a buildfile. List is comma separated (e.g. export.xml, build.xml, clean.xml).

Names: build.xml

Separate JRE timeout (ms): 20000

Documentation URL: http://ant.apache.org/manual

Some tasks, such as "javac", require the tools.jar library to be on the Ant runtime classpath to execute successfully.

☑ Warn when tools.jar is not on classpath

☑ Show error dialog when Ant build fails

☐ Create problem markers from "javac" results

Marker creation requires that a console be allocated and "listfiles" attribute be "true" in the "javac" declarations.

Ant color options:

- Ant error message in console
- Ant warning message in console
- Ant information message in console
- Ant verbose message in console
- Ant debug message in console

Figure 23-9 Ant preferences

- Expand the **Ant** folder and select **Editor**.
  - In the Appearance tab, you can change the layout preferences of your Ant file.
  - In the Syntax tab, you can change the syntax highlighting preferences with a preview of the results, and on the Problems tab you can define how certain problems should be handled.
  - In the Problems tab, you can change the severity levels for the buildfile problems.
  - In the Folding tab, you can enable folding when opening a new editor and specify which region types should be folded.
- Expand **Editor**.
  - In the Content Assist window, you can define the content assist preferences.
  - In the Formatter window, you can define the preferences for the formatting tool for the Ant files.

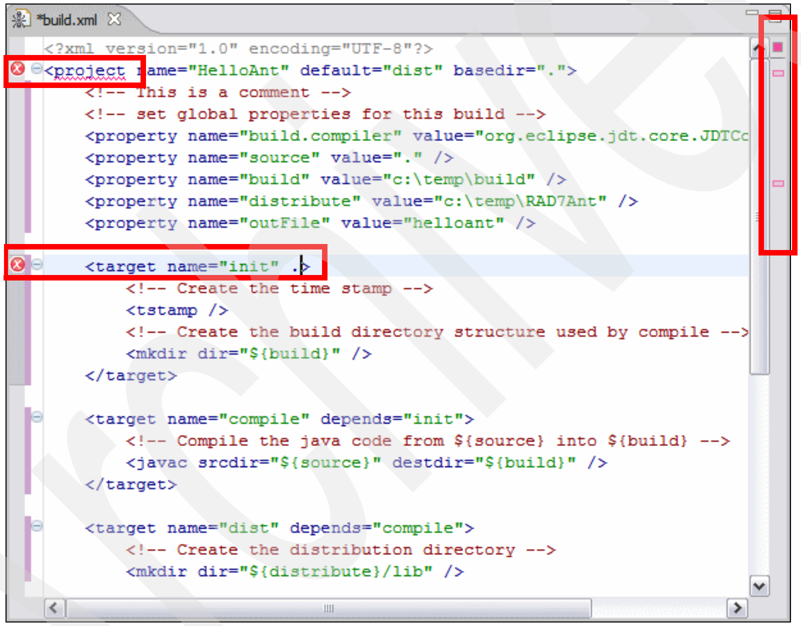
  - In the Templates window, you can create, edit, delete, import and export templates for Ant files.

- Select **Runtime**.

  In this dialog, you can define your preferences such as classpath, tasks, types, and properties.

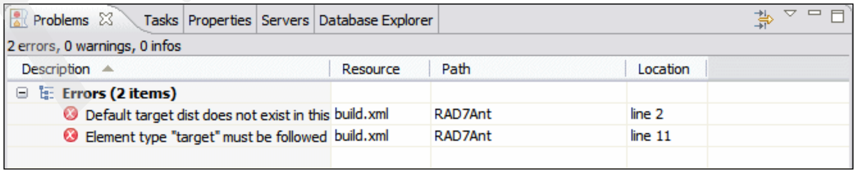
Problems view

Application Developer offers you the Problems view for the Ant file. The editor presents an error in the view by placing a *red X* on the left of the line with the problem as well as a line marker in the file on the right of the window, as shown in Figure 23-10. The Problems view lists the problems as seen in Figure 23-11.



```
<?xml version="1.0" encoding="UTF-8"?>
<project name="HelloAnt" default="dist" basedir=".">
    <!-- This is a comment -->
    <!-- set global properties for this build -->
    <property name="build.compiler" value="org.eclipse.jdt.core.JDTCc
    <property name="source" value="." />
    <property name="build" value="c:\temp\build" />
    <property name="distribute" value="c:\temp\RAD7Ant" />
    <property name="outFile" value="helloant" />

    <target name="init" .>
        <!-- Create the time stamp -->
        <tstamp />
        <!-- Create the build directory structure used by compile -->
        <mkdir dir="${build}" />
    </target>

    <target name="compile" depends="init">
        <!-- Compile the java code from ${source} into ${build} -->
        <javac srcdir="${source}" destdir="${build}" />
    </target>

    <target name="dist" depends="compile">
        <!-- Create the distribution directory -->
        <mkdir dir="${distribute}/lib" />
```

Figure 23-10 Problems in the Ant editor



2 errors, 0 warnings, 0 infos

Description	Resource	Path	Location
⊟ Errors (2 items)			
⊗ Default target dist does not exist in this	build.xml	RAD7Ant	line 2
⊗ Element type "target" must be followed	build.xml	RAD7Ant	line 11

Figure 23-11 Problems view displaying Ant problems

Building a simple Java application

We created a simple build file that compiles the Java source for our `HelloAnt` application and generates a JAR file with the result. The build file is called `build.xml`, which is the default name assumed by Ant if no build file name is supplied.

The example simple build file has the following targets:

- init: Performs build initialization tasks. All other targets depend upon this target.
- compile: Compiles Java source into class files.
- dist: Creates the deliverable JAR for the module, and depends upon the compile target.
- clean: Removes all generated files. Used to force a full build.

Each Ant build file can have a default target. This target is executed if Ant is invoked on a build file and no target is supplied as a parameter. In our example, the default target is dist. The dependencies between the targets are illustrated in Figure 23-12.

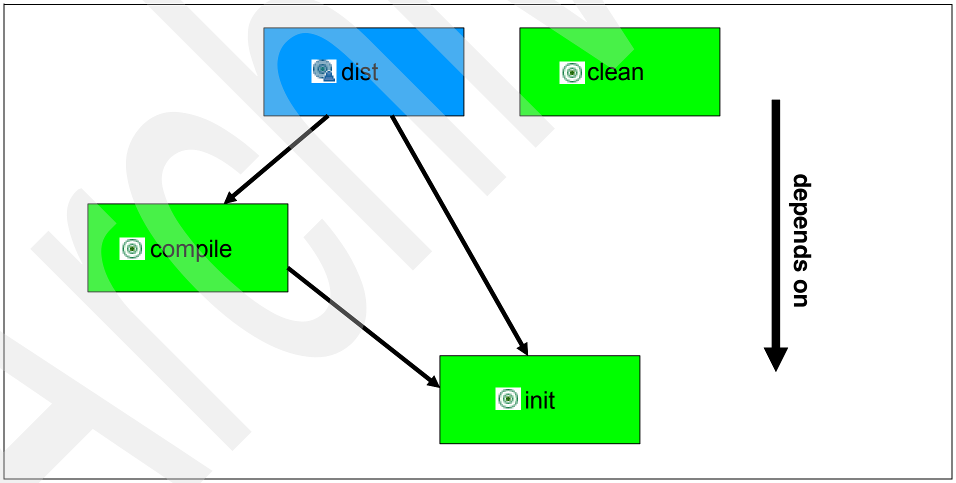


Figure 23-12 Ant example dependencies

Running Ant

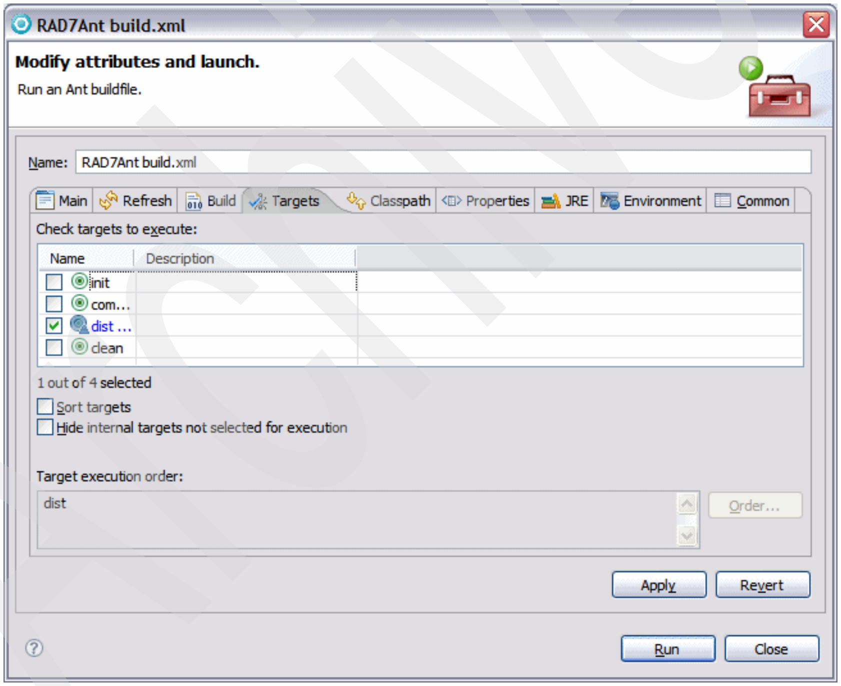
Ant is a built-in function to Application Developer. You can launch it from the context menu of any XML file, although it will run successfully only on valid Ant XML build script files. When launching an Ant script, you are given the option to select which targets to run.

To run our build script:

- Open the Java perspective.
- Expand the **RAD7Ant** project.
- Right-click **build.xml** and select **Run As** → **3 Run Ant** .
- In the Modify Attributes and Launch dialog (Figure 23-13), select the desired attributes. For example, select the **JRE** tab, select **Run in the same JRE as the workspace**, and then click **Run**.

  The default target specified in the build file is already selected as one target to run. You can check, in sequence, which ones are to be executed, and the execution sequence is shown in the Target execution order field.

> **Note:** Because the `dist` target depends on `compile`, even if you only select `dist`, the `compile` target is executed as well.


RAD7Ant build.xml

Modify attributes and launch.

Run an Ant buildfile.

Name: RAD7Ant build.xml

Check targets to execute:

	Name	Description
☐	init	
☐	com...	
☑	dist ...	
☐	clean	

1 out of 4 selected

☐ Sort targets

☐ Hide internal targets not selected for execution

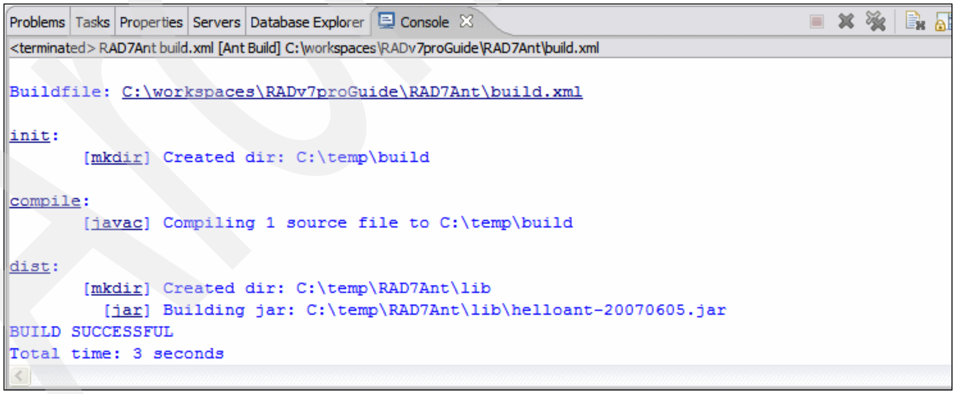
Target execution order:

dist

Figure 23-13 Selecting Ant targets to run

- The Run Ant wizard gives you several tabs to configure or run the Ant process. The tabs allow you to do the following operations:
  - Main: This tab allows you to select the build file, base directory, and arguments to pass to the Ant process.
  - Refresh: This tab allows you to set some refresh options when the Ant process has finished running.
  - Build: This tab allows you to set some build options before the Ant process is run.
  - Targets: This tab allows you to select the targets and the sequences the targets are to run.
  - Classpath: This tab allows you to customize the classpath for the Ant process.
  - Properties: This tab allows you to add, edit, or remove properties to be used by the Ant process.
  - JRE: This tab allows you to select the Java Runtime Environment to use to run the Ant process.
  - Environment: This tab allows you to define environmental variables to be used by the Ant process. This tab is only relevant when running in an external JRE.
  - Common: This tab allows you to define the launch configuration for the Ant process.

  When Ant is running, you can see the output in the Console (Figure 23-14).



<terminated> RAD7Ant build.xml [Ant Build] C:\workspaces\RADv7proGuide\RAD7Ant\build.xml

```
Buildfile: C:\workspaces\RADv7proGuide\RAD7Ant\build.xml

init:
       [mkdir] Created dir: C:\temp\build

compile:
       [javac] Compiling 1 source file to C:\temp\build

dist:
       [mkdir] Created dir: C:\temp\RAD7Ant\lib
         [jar] Building jar: C:\temp\RAD7Ant\lib\helloant-20070605.jar
BUILD SUCCESSFUL
Total time: 3 seconds
```

Figure 23-14   Ant output in the Console

Ant console

The Console view opens automatically when running Ant, but if you want to open it manually, select **Window** → **Show view** → **Console**.

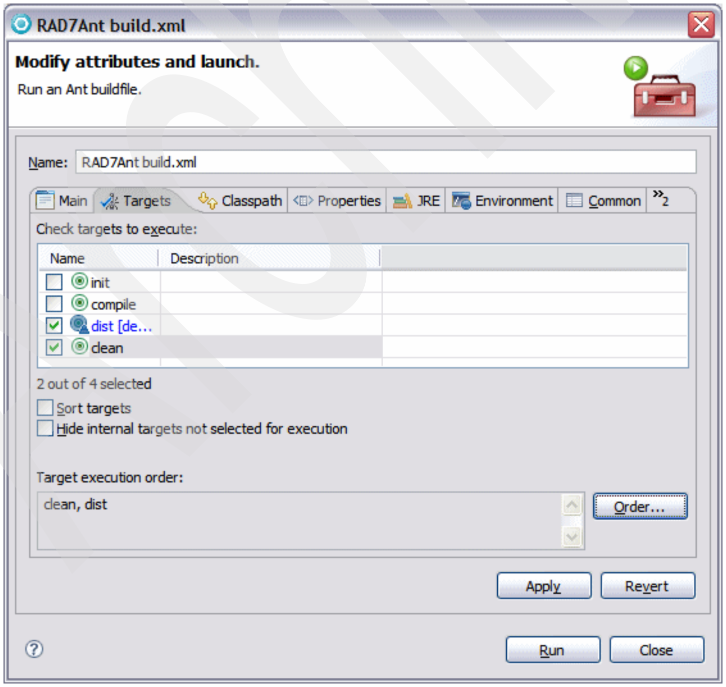
The Console shows that Ant has created the `c:\temp\build` directory, compiled the source files, created the `c:\temp\RAD7Ant\lib` directory, and generated a JAR file (`helloant-20070605.jar`).

Rerun Ant

If you launch Ant again with the same target selected, Ant will not do anything at all, because the `c:\temp\build` and `c:\temp\RAD7Ant\lib` directories were already created, and the class files in the build directory were already up-to-date.

Forced build

To generate a complete build, select the clean target as the first target and the dist target as the second target to run. You have to de-select dist, select clean, and then select dist again to get the execution order right (Figure 23-15). Alternatively you can click **Order** to change the execution order.



Name: RAD7Ant build.xml

Check targets to execute:

Name	Description
☐ init	
☐ compile	
☑ dist [de...	
☑ clean	

2 out of 4 selected

☐ Sort targets

☐ Hide internal targets not selected for execution

Target execution order:

clean, dist

Figure 23-15 Launching Ant to generate complete build

Classpath problem

The classpath specified in the Java build path for the project is not available to the Ant process. If you are building a project that references another project, the classpath for the javac compiler must be set up in the following way:

```
<javac srcdir="${source}" destdir="${build}" includes="**/*.java">
   <classpath>
      <pathelement location="../MyOtherProject"/>
      <pathelement location="../MyThirdProject"/>
   </classpath>
</javac>
```

Running the sample application to verify the Ant build

Now that you have completed the Ant build, we recommend that you verify the build by running the sample application as follows:

- Open a Windows command window.
- Navigate to the output directory of the Ant build (for example, `c:\temp\RAD7Ant\lib`).
- Set the Java path by entering the following command:

  Enter the location of the installation directory (for example, our installation directory is found in `c:\IBM\SDP70`).

  ```
  set PATH=%PATH%;c:\IBM\SDP70\runtimes\base_v61\java\bin
  ```

- Enter the following to run the program:

  ```
  java -jar helloant-20070605.jar
  ```

  Note that the timestamp in the jar filename is dependent on when it is built.

- You should see the following output:

  ```
  Hello from Turkey!
  ```

Building a J2EE application

As we have just demonstrated in the previous section, building a simple Java application using Ant is quite easy. In this section we demonstrate how to build a J2EE application from existing J2EE-related projects.

This section is organized as follows:

- J2EE application deployment packaging
- Preparing for the sample

- Creating the build script
- Running the Ant J2EE application build

J2EE application deployment packaging

EAR, WAR, and EJB JAR files contain a number of deployment descriptors that control how the artifacts of the application are to be deployed onto an application server. These deployment descriptors are mostly XML files and are standardized within the J2EE specification.

While working in Application Developer, some of the information in the deployment descriptor is stored in XML files. The deployment descriptor files also contain information in a format convenient for interactive testing and debugging. This is one of the reasons that it is so quick and easy to test J2EE applications in the integrated WebSphere Application Server V6.1 Test Environment included with Rational Application Developer.

The actual EAR being tested, and its supporting WAR, EJB, and client application JARs, are not actually created as a standalone file. Instead, a special EAR is used that simply points to the build contents of the various J2EE projects. Since these individual projects can be anywhere on the development machine, absolute path references are used.

When an enterprise application project is exported, a true standalone EAR is created, including all the module WARs, EJB JARs, and Java utility JARs it contains. Therefore, during the export operation, all absolute paths are changed into self-contained relative references within that EAR, and the internally optimized deployment descriptor information is merged and changed into a standard format. To create a J2EE-compliant WAR or EAR, we therefore have to use Application Developer's export function.

Preparing for the sample

For the purposes of demonstrating how to build a J2EE application using Ant, we will use the J2EE applications developed in Chapter 16, “Develop Web applications using EJBs” on page 719.

To import the `RAD7EJB.zip` project interchange file containing the sample code into Rational Application Developer, do these steps:

- Open the J2EE perspective Project Explorer view.
- Select **File** → **Import**.
- Expand the **Other** folder and select **Project Interchange** from the list of import sources and then click **Next**.

- In the Import Projects dialog, click **Browse** next to the zip file, navigate to and select the **RAD7EJB.zip** from the `c:\7501code\zInterchangeFiles\ejb` folder, and click **Open**.

> **Note:** For information on downloading the sample code, refer to Appendix B, "Additional material" on page 1307.

- Click **Select All** to select all projects and then click **Finish**.
- Repeat the import for `c:\7501code\zInterchangeFiles\ejb\RAD7EJBWeb.zip`, and select both projects (`RAD7EJBWeb` and `RAD7EJBWebEAR`).

After importing the two project interchange files, you have the following projects:

- `RAD7EJBEAR`
- `RAD7EJB`
- `RAD7EJBClient`
- `RAD7EJBJava`
- `RAD7EJBWeb`
- `RAD7EJBWebEAR`

Creating the build script

To build the `RAD7EJBEAR` enterprise application, we created an Ant build script (`build.xml`) that utilizes the J2EE Ant tasks provided by Application Developer.

To add the Ant build script to the project, do these steps:

- Open the J2EE perspective Project Explorer view.
- Expand **RAD7EJBEAR**, and select **META-INF**.
- Select **File** → **New** → **Other**.
- In the New File dialog, select **General** → **File**, and click **Next**.
- Enter `build.xml` in the File name field, and click **Finish**.
- Double-click the **build.xml** file to open it in the editor. Copy and paste the text in `c:\7501code\ant\j2ee\build.txt` into the `build.xml` file.
- Modify the value for the `work.dir` property to match your desired working directory (for example, `c:/RAD7EAR_workdir`), as highlighted in Example 23-1.

Example 23-1 J2EE Ant build.xml script

```
<?xml version="1.0" encoding="UTF-8"?>
<project name="ITSO RAD Pro Guide Ant" default="Total" basedir=".">

   <!-- Set global properties -->
   <property name="work.dir" value="c:/RAD7EAR_workdir" />
   <property name="dist" value="${work.dir}/dist" />
   <property name="project.ear" value="RAD7EJBEAR" />
   <property name="project.ejb" value="RAD7EJB" />
   <property name="project.war" value="RAD7EJBWeb" />
   <property name="type" value="incremental" />
   <property name="debug" value="true" />
   <property name="source" value="true" />
   <property name="meta" value="false" />
   <property name="noValidate" value="false" />
```

The `build.xml` script includes the following Ant targets, which correspond to common J2EE application build:

- `deployEjb`: This generates the deploy code for all EJBs in the project.
- `buildEjb`: This builds the EJB project (compiles resources within the project).
- `buildWar`: This builds the Web project (compiles resources within the project).
- `buildEar`: This builds the Enterprise Application project (compiles resources within the project).
- `exportEjb`: This exports the EJB project to a jar file.
- `exportWar`: This exports the Web project to a WAR file.
- `exportEar`: This exports the Enterprise Application project to an EAR file.
- `buildAll`: This invokes the `buildEjb`, `buildWar`, and `buildEar` targets.
- `exportAll`: This invokes the `exportEjb`, `exportWar`, and `exportEar` targets to create the `RAD7EJBEAR.ear` used for deployment.

In the global properties for this script, we define a number of useful variables, such as the project names and the target directory. We also define a number of properties that we pass on to the Application Developer Ant tasks. These properties allow us to control whether the build process should perform a full or incremental build, whether debug statements should be included in the generated class files, and whether Application Developer's metadata information should be included when exporting the project.

When launching this Ant script, we can also override these properties by specifying other values in the arguments field, allowing us to perform different kinds of builds with the same script.

Running the Ant J2EE application build

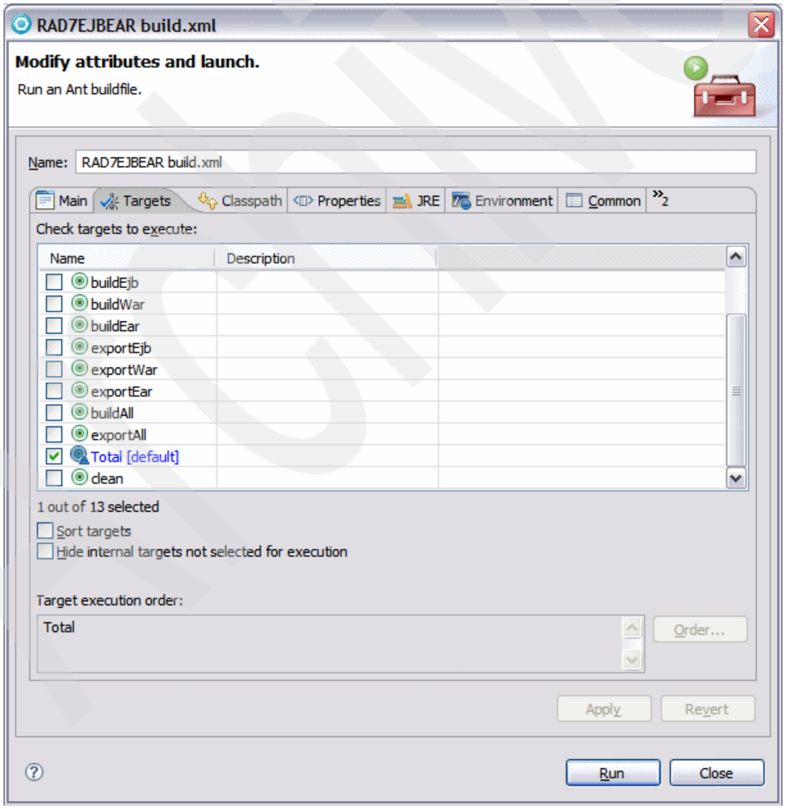
When launching the build.xml script, you can select which targets to run and the execution order.

To run the Ant `build.xml` to build the J2EE application, do these steps:

- Open the J2EE perspective Project Explorer view.
- Expand **RAD7EJBEAR** → **META-INF**.
- Right-click **build.xml**, and select **Run As** → **3 Ant Build...**.

> **Note:** From the content menu for the Ant build script, the following two build options exist:
>
> - 2 Ant Build: This will invoke the default target for the Ant build. In our example, this is the Total target, which in turn invokes buildAll and exportAll targets.
> - 3 Ant Build: This will launch a dialog where you can select the targets and order, and provide parameters, as seen in Figure 23-16.

- The Modify attributes and launch dialog opens (Figure 23-16).



RAD7EJBEAR build.xml

Modify attributes and launch.

Run an Ant buildfile.

Name: RAD7EJBEAR build.xml

Check targets to execute:

Name	Description
□ buildEjb	
□ buildWar	
□ buildEar	
□ exportEjb	
□ exportWar	
□ exportEar	
□ buildAll	
□ exportAll	
☑ Total [default]	
□ clean	

1 out of 13 selected

□ Sort targets

□ Hide internal targets not selected for execution

Target execution order:

Total

Figure 23-16 Launch Ant to build and export a J2EE project (EAR)

- Select the **Main** tab:
  - To build the J2EE EAR file with debug, source files, and meta data, enter the following values in the Arguments text area:

    ```
    -DDebug=true -Dsource=true -Dmeta=true
    ```

  Or:

  - To build the J2EE EAR for production deployment (without debug support, source code, and meta data), enter the following value in the Arguments text area:

    ```
    -Dtype=full
    ```

- Select the **Targets** tab, and select **Sort Targets**. Ensure that **Total** is selected (default).
- Select the **JRE** tab. Select **Run in the same JRE as the workspace**.
- Click **Apply** and then click **Run**.
- Change to the following directory to ensure that the `RAD7EJBEAR.ear` file was created:

  ```
  c:\RAD7EAR_workdir\dist
  ```

The Console view displays the operations performed and their results.

Running Ant outside of Application Developer

To automate the build process even further, you might want to run Ant outside of Application Developer by running Ant in *headless mode*.

Preparing for the headless build

Rational Application Developer includes a `runAnt.bat` file that can be used to invoke Ant in headless mode and passes the parameters that you specify. This will have to be customized for your environment.

The `runAnt.bat` file included with Rational Application Developer is located in the following directory (for example, our `<rad_home>` directory is found in `c:\IBM\SDP70`):

```
<rad_home>\bin
```

To create a headless Ant build script for J2EE project, do these steps:

- Copy the `runAnt.bat` file to a new file called `itsoRunAnt.bat`.

- Modify the following values in the `itsoRunAnt.bat` (Example 23-2):

```
set AST_DIR=C:\IBM\SDP70
set JAVA_DIR=%AST_DIR%\jdk\jre\bin
set WORKSPACE=C:\workspaces\RADv7proGuide
```

Example 23-2 Snippet of the itsoRunAnt.bat (modified runAnt.bat)

```
@echo off
setlocal
set AST_DIR=C:\IBM\SDP70

:java
if not $%JAVA_DIR%$==$$ goto workspace
set JAVA_DIR=%AST_DIR%\jdk\jre\bin

:workspace
if not $%WORKSPACE%$==$$ goto check
REM #######################################################
REM ##### you must edit the "WORKSPACE" setting below #####
REM #######################################################
REM *********** The location of your workspace ************
set WORKSPACE=C:\workspaces\RADv7proGuide
```

Running the headless Ant build script

Attention: Prior to running Ant in headless mode, Rational Application Developer must be closed. If you do not close Rational Application Developer, you will get build errors when attempting to run Ant build in headless mode.

To run the `itsoRunAnt.bat` command file, do these steps:

- Ensure that you have closed Application Developer.
- Open a Windows command prompt.
- Navigate to the location of the `itsoRunAnt.bat` file.
- Run the command file by entering the following command:

```
itsoRunAnt -buildfile
   c:\workspaces\RADv7proGuide\RAD7EJBEAR\META-INF\build.xml clean Total
   -DDebug=true -Dsource=true -Dmeta=true
```

  The `-buildfile` parameter should specify the fully qualified path of the `build.xml` script file. We can pass the targets to run as parameters to `itsoRunAnt` and we can also pass Java environment variables by using the `-D` switch.

In this example, we chose to run the clean and Total targets, and we chose to include the debug, Java source, and metadata files in the resulting EAR file.

Note: We have included files named `itsoRunAnt.xml` and `output.txt`, which contains the build file and output from the headless Ant script for review purposes. The files can be found in the `c:\7501code\ant\j2ee` directory.

- There are several build output files, which can be found in the `c:\RAD7EAR_workdir\dist` directory:
  - `RAD7EJBWeb.war`—`RAD7EJBWeb` project
  - `RAD7EJB.war`—`RAD7EJB` project
  - `RAD7EJBEAR.ear`—`RAD7EJBEAR` project with all the dependent projects included

More information on Ant

For more information on Ant, we recommend the following resources:

- *Automatically generate project builds using Ant*, white paper found at:

  `http://www.ibm.com/developerworks/library/ar-auototask/`

24

Deploy enterprise applications

The term *deployment* can have many different meanings depending on the context. In this chapter we start out by defining the concepts of application deployment. The remainder of the chapter provides a working example for packaging and deploying the ITSO Bank enterprise application to a standalone IBM WebSphere Application Server V6.1.

The application deployment concepts and procedures described in this chapter apply to Application Server V6.1 (Base, Express, and Network Deployment editions). In Application Developer V7, the configuration of the integrated Application Server V6.1 for deployment, testing, and administration is the same for the separately installed Application Server V6.1. In previous versions of WebSphere Studio, the configuration of the test environment v5.x was different than a separately installed Application Server 5.x.

The chapter is organized into the following sections:

- Introduction to application deployment
- Preparing for the deployment of the EJB application
- Packaging the application for deployment
- Manual deployment of enterprise applications
- Automated deployment using Jython based wsadmin scripting

Introduction to application deployment

Deployment is a critical part of the J2EE application development cycle. Having a solid understanding of the deployment components, architecture, and process is essential for the successful deployment of the application.

In this section we review the following concepts of the J2EE and WebSphere deployment architecture:

- Common deployment considerations
- J2EE application components and deployment modules
- Preparing for the deployment of the EJB application
- WebSphere deployment architecture
- Java and WebSphere class loader

> **Note:** Further information on the IBM Application Server deployment can be found in the following sources:
>
> - *WebSphere Application Server V6.1: Planning and Design*, SG24-7305
> - *WebSphere Application Server V6.1: Systems Management and Configuration*, SG24-7304
> - *WebSphere Application Server V6: Scalability and Performance*, SG24-6392
> - IBM WebSphere Application Server V6.1 InfoCenter, found at:
>
>   http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.base.doc/info/welcome_base.html

Common deployment considerations

Some of the most common factors that impact the deployment of a J2EE application are as follows:

- **Deployment architecture**: How can you create, assemble, and deploy an application properly if you do not understand the deployment architecture?
- **Infrastructure**: What are the hardware and software constraints for the application?
- **Security**: What security will be imposed on the application and what is the current security architecture?
- **Application requirements**: Do they imply a distributed architecture?
- **Performance**: How many users are using the system (frequency, duration, and concurrency)?

J2EE application components and deployment modules

Within the J2EE application development life cycle, the application components are created, assembled, and then deployed. In this section, we explore the application component types, deployment modules, and packaging formats to gain a better understanding of what is being packaged (assembled) for deployment.

Application component types

In J2EE V1.4, there are four application component types supported by the runtime environment:

- Application clients: Run in the Application client container.
- Applets: Run in a browser (or a stand-alone applet container).
- Web applications (servlets, JSPs, HTML pages): Run in the Web container.
- EJBs: Run in the EJB container.

Deployment modules

The J2EE deployment components are packaged for deployment as modules:

- Web module
- EJB module
- Resource adapter module
- Application client module

Packaging formats (WAR and EAR)

There are two key packaging formats used to package J2EE modules for deployment, namely Web Application Archive (WAR) and Enterprise Application Archive (EAR). The packaging technology for both is similar to that of a jar file. WAR files can include servlets, JSPs, HTML, and images. Enterprise Application Archive (EAR) can be used to package EJB modules, resource adapter modules, and Web modules.

Deployment descriptors

Information describing a J2EE application and how to deploy it into a J2EE container is stored in XML files called deployment descriptors. An EAR file normally contains multiple deployment descriptors, depending on the modules it contains. Figure 24-1 shows a schematic overview of a J2EE EAR file. In this figure the various deployment descriptors are designated with DD after their name.

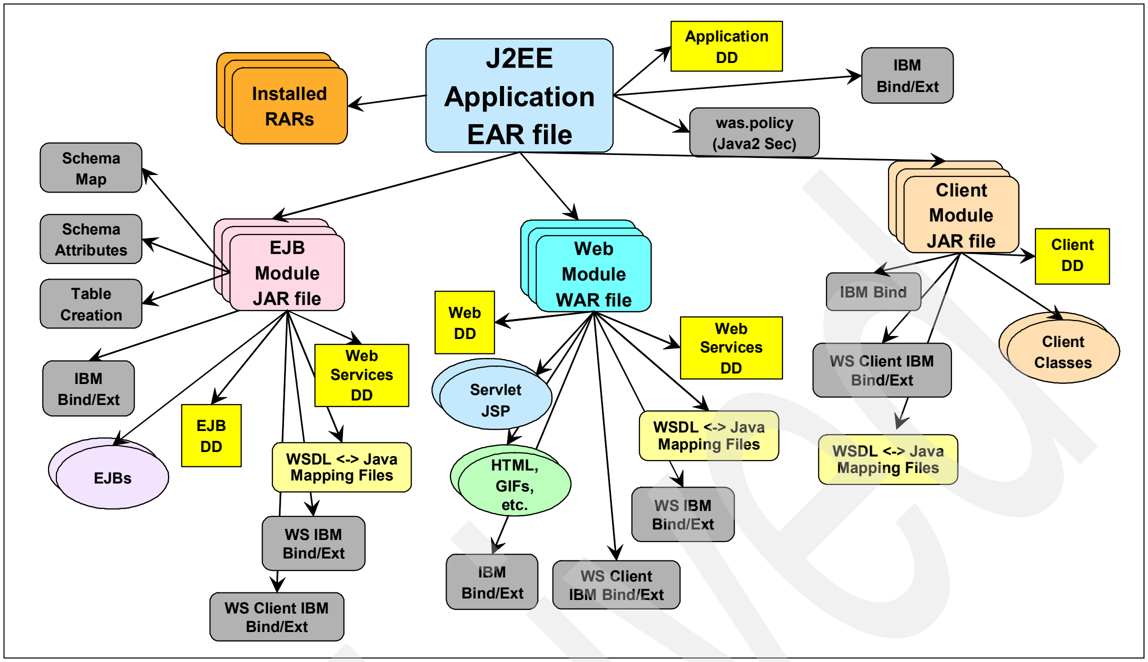


Figure 24-1 J2EE EAR file structure

The deployment descriptor of the EAR file itself is stored in the `META-INF` directory in the root of the EAR and is called application.xml. It contains information about the modules making up the application.

The deployment descriptors for each module are stored in the `META-INF` directory of the module and are called `web.xml` (for Web modules, actually in `WEB-INF`), `ejb-jar.xml` (for EJB modules), `ra.xml` (for resource adapter modules), and `application-client.xml` (for Application client modules). These files describe the contents of a module and allow the J2EE container to configure servlet mappings, JNDI names, and so forth.

Classpath information, specifying which other modules and utility JARs are needed for a particular module to run, is stored in the `manifest.mf` file, also in the `META-INF` directory of the modules.

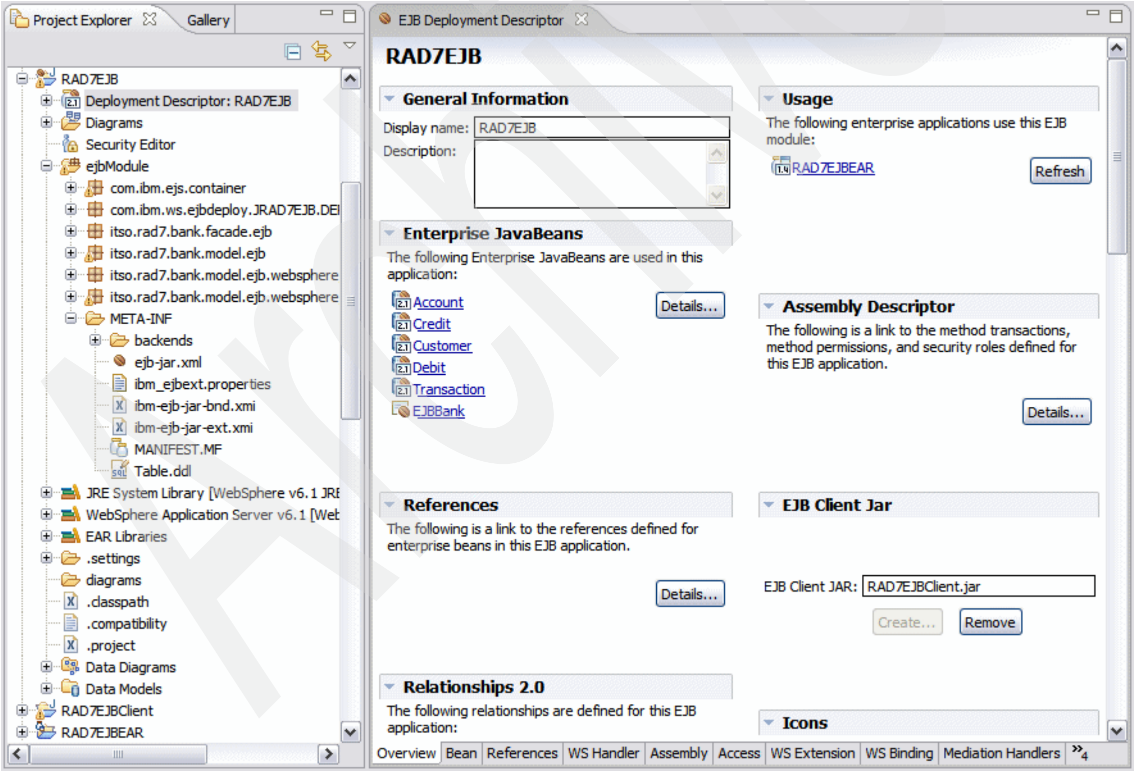
In addition to the standard J2EE deployment descriptors, EAR files produced by Application Developer or the Application Server Toolkit can also include additional WebSphere-specific information used when deploying applications to WebSphere environments. This supplemental information is stored in an XMI file, also in the `META-INF` (or `WEB-INF`) directory of the respective modules. Examples of information in the IBM-specific files are IBM extensions, such as servlet reloading and EJB access intents.

Application Developer and the Application Server Toolkit have easy-to-use editors for working with all deployment descriptors. The information that goes into the different files is shown on one page in the IDE, eliminating the need to be concerned about what information is put into what file. However, if you are interested, you can click the Source tab of the Deployment Descriptor editor to see the text version of what is actually stored in that descriptor.

For example, if you open the EJB deployment descriptor, you will see settings that are stored across multiple deployment descriptors for the EJB module, including:

- EJB deployment descriptor, `ejb-jar.xml`
- Extensions deployment descriptor, `ibm-ejb-jar-ext.xmi`
- Bindings file, `ibm-ejb-jar-bnd.xmi`
- Access intent settings, `ibm-ejb-access-bean.xmi`

The deployment descriptors can be modified in Application Developer by double-clicking the file to open the Deployment Descriptor Editor (Figure 24-2).



Display name: RAD7EJB

Description:

The following enterprise applications use this EJB module:

RAD7EJBEAR

The following Enterprise JavaBeans are used in this application:

The following is a link to the method transactions, method permissions, and security roles defined for this EJB application.

The following is a link to the references defined for enterprise beans in this EJB application.

EJB Client JAR: RAD7EJBClient.jar

The following relationships are defined for this EJB application:

Figure 24-2 Deployment Descriptor editor in Application Developer

While the editor shows you information stored in all the relevant deployment descriptor files on the appropriate tabs, the Source tab only shows the source of the deployment descriptor itself (for example, `ejb-jar.xml` or `web.xml`), and not the IBM extensions and bindings stored in the WebSphere-specific deployment descriptor files. If you want to view the results of updates to those files in the source, you have to open each file individually. By hovering over the EJB Deployment Descriptor Caption tab, you can see the different files that make up the EJB deployment descriptor you are editing. The descriptor files are kept in the `META-INF` directory of the module you are editing.

When you have made changes to a deployment descriptor, save and close it.

> **Note:** “Customizing the deployment descriptors” on page 1125 provides an example of customizing the ITSO Bank EJB deployment descriptors for the desired database server type (Derby or DB2 Universal Database).

WebSphere deployment architecture

This section provides an overview of the IBM WebSphere Application Server V6.1 deployment architecture.

Application Developer V7.0 includes an integrated Application Server V6.1. Administration of the server and applications is performed by using the WebSphere Administrative Console for such configuration tasks as:

- J2C authentication aliases
- Data sources
- Service buses
- JMS queues and connection factories

Due to the loose coupling between Application Developer and Application Server, applications can deploy in the following ways:

- Deploy from an Application Developer project to the integrated Application Server V6.1.
- Deploy from an Application Developer project to a separate Application Server runtime environment.
- Deploy through an EAR to an integrated Application Server V6.1.
- Deploy through an EAR to a separate Application Server runtime environment.

Administration of the application server is performed through the WebSphere Administrative Console that runs in a Web browser.

In addition, the Application Server V6.1 can obtain an EAR from external tools without the deployment code generated and be loaded into the Application Developer or Application Server Toolkit (AST). Application Developer or AST generates the deployment code for the Application Server and a new EAR is saved to deploy to the application server.

A diagram of the deployment architecture and the various mechanisms to deploy out an application are provided in Figure 24-3.

Details on how to configure the servers in Application Developer V7 are documented in Chapter 20, “Servers and server configuration” on page 963.

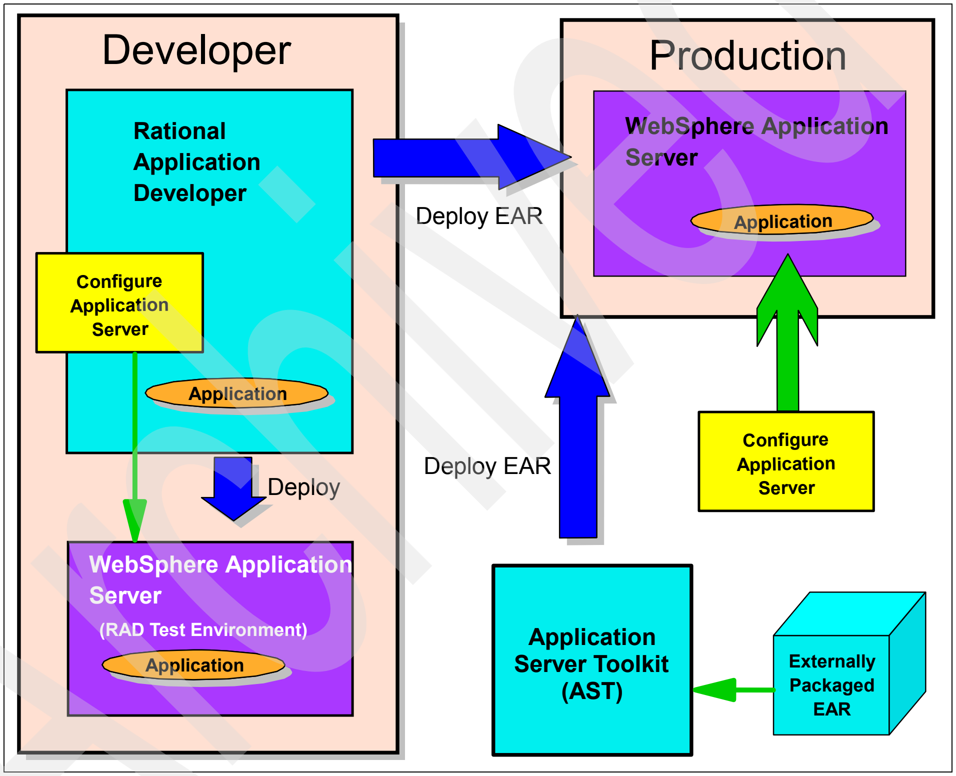

Developer
Rational Application Developer
Configure Application Server
Application
Deploy
WebSphere Application Server
(RAD Test Environment)
Application
Deploy EAR
Production
WebSphere Application Server
Application
Deploy EAR
Configure Application Server
Application Server Toolkit (AST)
Externally Packaged EAR

Figure 24-3 Deployment architecture

WebSphere profiles

Application Server V6.0 introduced the concept of WebSphere profiles. It has been split into two separate components:

- A set of shared read-only product files
- A set of configuration files known as WebSphere profiles

The first component is the runtime part of Application Server, while the second component is a new concept.

A WebSphere profile is the set of configurable files including Application Server configuration, applications, and properties files that constitute a new application server. Having multiple profiles equates to having multiple Application Server instances for use with a number of applications.

> **Note:** This function is similar to the **`wsinstance`** command that was available in Application Server V5.x, creating multiple runtime configurations using the same installation.

In Application Developer V7, this allows a developer to configure multiple application servers for various applications that they might be working with. Separate WebSphere profiles can then be set up as test environments in Rational Web/Application Developer (see Chapter 20, “Servers and server configuration” on page 963).

WebSphere enhanced EAR

WebSphere Application Server V6 introduced the **enhanced EAR** feature. The enhanced EAR information, which includes settings for the resources required by the application, is stored in an **ibmconfig** subdirectory of the enterprise application (EAR file) `META-INF` directory.

In the `ibmconfig` directory are the well-known directories for a WebSphere cell configuration (Figure 24-5 on page 1113). The enhanced EAR feature provides an extension of the J2EE EAR with additional configuration information for resources typically required by J2EE applications. This information is not mandatory to be supplied at packaging time, but it can simplify the deployment of applications to Application Server for selected scenarios.

The Enhanced EAR editor can be used to edit several WebSphere Application Server V6 specific configurations, such as JDBC providers, data sources, class loader policies, substitution variables, shared libraries, virtual hosts, and authentication settings. The configuration settings can be made simply within the editor and published with the EAR at the time of deployment.

The upside of the tool is that it makes the testing process simpler and repeatable, since the configurations can be saved to files and then shared within a team’s repository. Even though it will not let you configure every possible runtime setting, it is a good tool for development purposes because it eases the process of configuring the most common ones.

The downside is that the configurations the tool makes will be attached to the EAR, and will not be visible from the WebSphere Administrative Console. The WebSphere Administrative Console is only able to edit settings that belong to the cluster, node, and server contexts.

When you change a configuration using the Enhanced EAR editor, these changes are made within the application context. The deployer can still make changes to the EAR file using the Application Server Toolkit (AST), but it still requires a separate tool. Furthermore, in most cases these settings are dependent on the node the application server is installed in anyway, so it might not make sense to configure them at the application context for production deployment purposes.

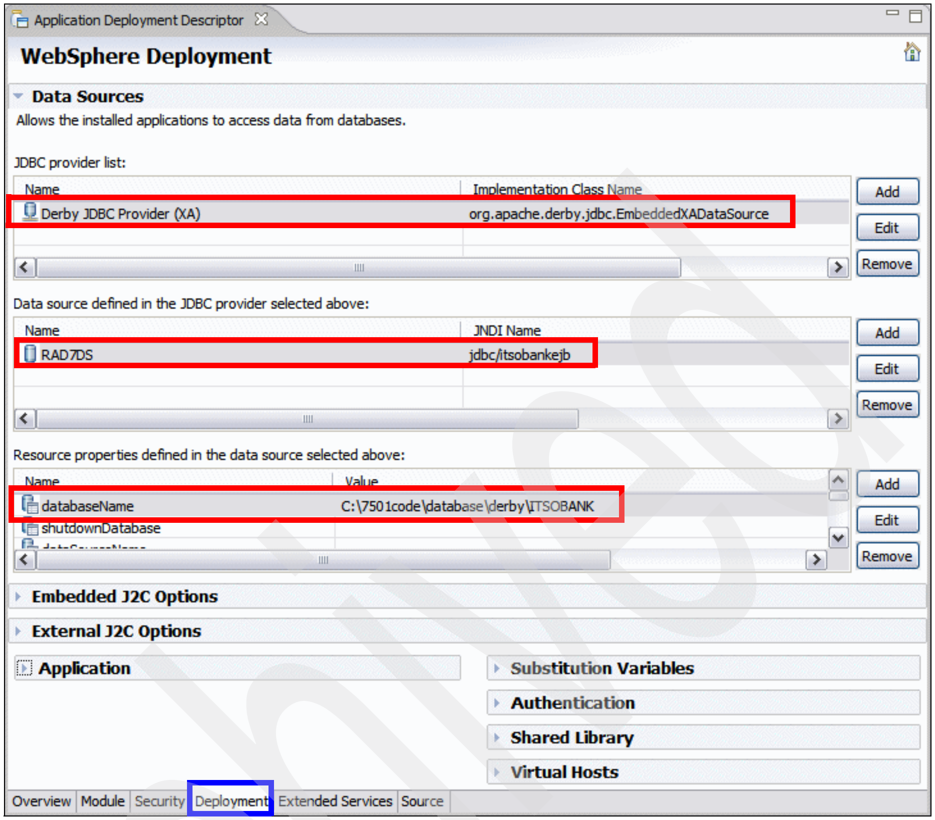
Table 24-1 lists the supported resources that the enhanced EAR provides and the scope in which they are created.

Table 24-1 Enhanced EAR resources supported and their scope

Scope	Resources
Application	JDBC providers. data sources, substitution variables, class loader policies
Server	Shared libraries
Cell	JAAS authentication aliases, virtual hosts

Figure 24-4 shows the view of the Deployment tab of the Application Deployment Descriptor. This is where the enhanced EAR information can be configured in Application Developer. It displays the configured data source (RAD7DS) and its JDBC Provider (Derby JDBC Provider (XA)).

Note that in the first section (Data Sources) you have to select a JDBC provider to see the associated data sources, and after selecting a data source you can see the properties of that data source.



Allows the installed applications to access data from databases.

JDBC provider list:

Name	Implementation Class Name
Derby JDBC Provider (XA)	org.apache.derby.jdbc.EmbeddedXADataSource

Data source defined in the JDBC provider selected above:

Name	JNDI Name
RAD7DS	jdbc/itsobankejb

Resource properties defined in the data source selected above:

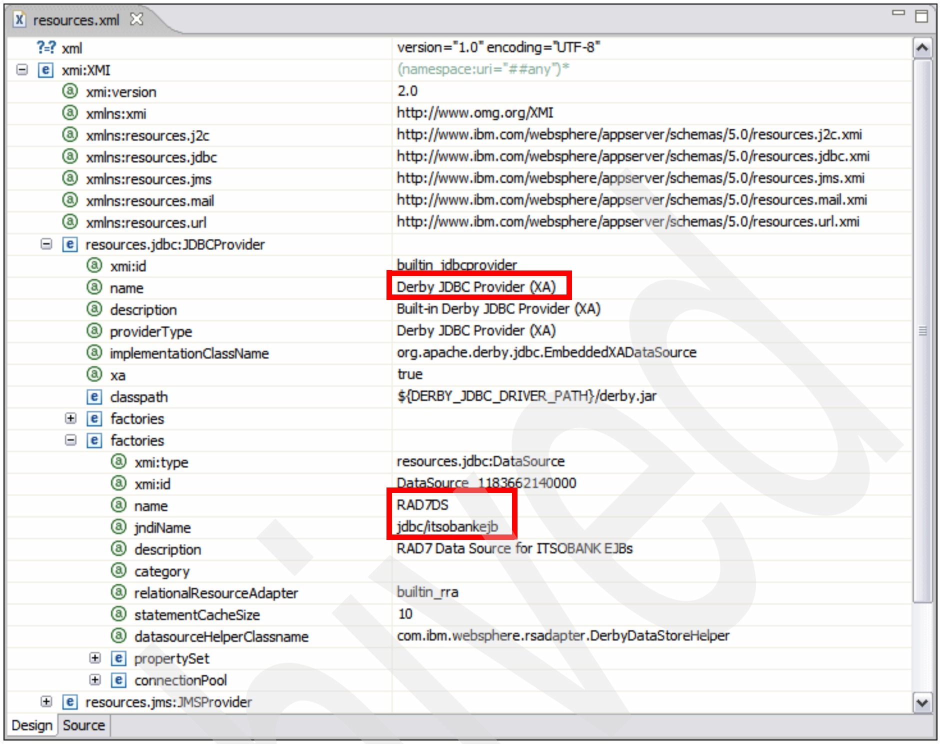
Name	Value
databaseName	C:\7501code\database\derby\ITSOBANK
shutdownDatabase	

Figure 24-4   Enhanced EAR Deployment Descriptor: Deployment tab

Figure 24-5 displays the contents of the **resources.xml** file that is part of the enhanced EAR information stored in:

```
ibmconfig/cells/defaultCell/applications/defaultApp/deployments/defaultApp
```

You can also notice that in the `ibmconfig` directory are the well-known directories for a WebSphere cell configuration. The XML editor of Application Developer displays the configuration contents for the RAD7DS data source and the Derby JDBC Provider (XA).



?-? xml	version="1.0" encoding="UTF-8"
xmi:XMI	(namespace:uri="##any")*
xmi:version	2.0
xmlns:xmi	http://www.omg.org/XMI
xmlns:resources.j2c	http://www.ibm.com/websphere/appserver/schemas/5.0/resources.j2c.xmi
xmlns:resources.jdbc	http://www.ibm.com/websphere/appserver/schemas/5.0/resources.jdbc.xmi
xmlns:resources.jms	http://www.ibm.com/websphere/appserver/schemas/5.0/resources.jms.xmi
xmlns:resources.mail	http://www.ibm.com/websphere/appserver/schemas/5.0/resources.mail.xmi
xmlns:resources.url	http://www.ibm.com/websphere/appserver/schemas/5.0/resources.url.xmi
resources.jdbc:JDBCProvider	
xmi:id	builtin_jdbcprovider
name	Derby JDBC Provider (XA)
description	Built-in Derby JDBC Provider (XA)
providerType	Derby JDBC Provider (XA)
implementationClassName	org.apache.derby.jdbc.EmbeddedXADataSource
xa	true
classpath	${DERBY_JDBC_DRIVER_PATH}/derby.jar
factories	
factories	
xmi:type	resources.jdbc:DataSource
xmi:id	DataSource_1183662140000
name	RAD7DS
jndiName	jdbc/itsobankejb
description	RAD7 Data Source for ITSOBANK EJBs
category	
relationalResourceAdapter	builtin_rra
statementCacheSize	10
datasourceHelperClassname	com.ibm.websphere.rsadapter.DerbyDataStoreHelper
propertySet	
connectionPool	
resources.jms:JMSProvider	

Figure 24-5 Enhanced EAR: resources.xml

Note: For more detailed information and an example of using the WebSphere enhanced EAR, refer to these sources:

- *Packaging applications* chapter in the *WebSphere Application Server V6.1: Systems Management and Configuration*, SG24-7304
- IBM WebSphere Application Server V6.1 Info Center, found at:

  `http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.base.doc/info/welcome_base.html`

An example of defining a data source using the Enhanced EAR editor can be found in "Creating a data source in the enhanced EAR" on page 985.

WebSphere Rapid Deployment

WebSphere Rapid Deployment is a collection of tools and technologies introduced in IBM WebSphere Application Server V6.1 to make application development and deployment easier than ever before.

WebSphere Rapid Deployment consists of the following elements:

- Annotation-based programming (Xdoclet)
- Rapid deployment tools
- Fine-grained application updates

Annotation-based programming (Xdoclet)

Annotation-based programming speeds up application development by reducing the number of artifacts that you have to develop and manage on your own. By adding metadata tags to the Java code, the WebSphere Rapid Deployment tools can automatically create and manage the artifacts to build a J2EE-compliant module and application.

Rapid deployment tools

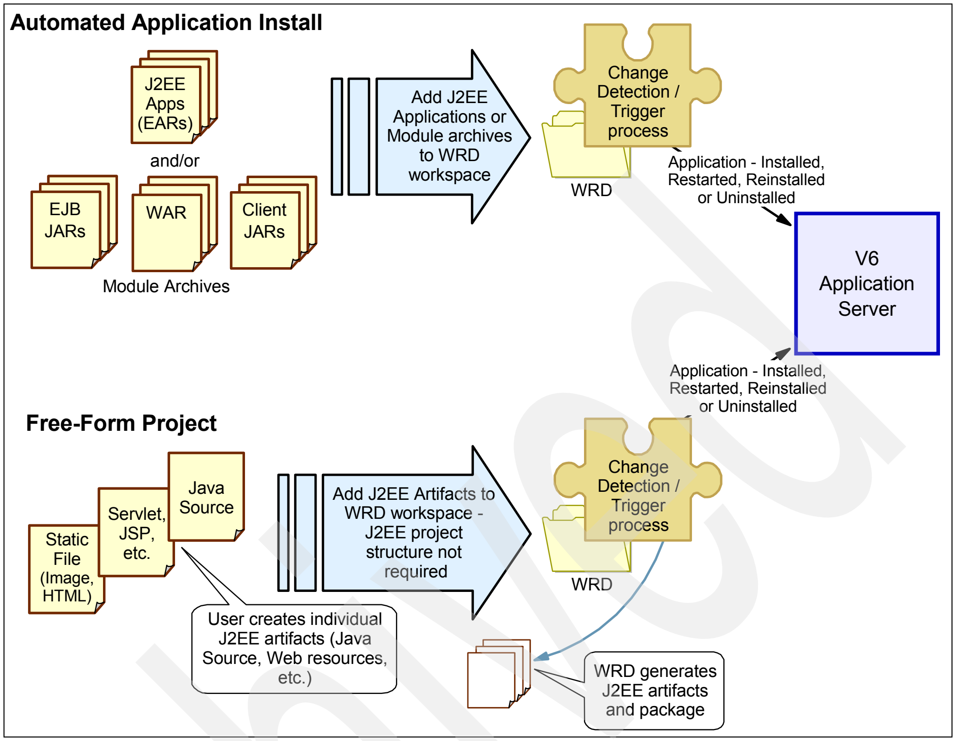
Using the rapid deployment tools part of WebSphere Rapid Deployment, you can accomplish the following tasks:

- Create a new J2EE application quickly without the overhead of using an integrated development environment (IDE).
- Package J2EE artifacts quickly into an EAR file.
- Deploy and test J2EE modules and full applications quickly on a server.

For example, you can place full J2EE applications (EAR files), application modules (WAR files, EJB JAR files), or application artifacts (Java source files, Java class files, images, and JSPs) into a configurable location on your file system, referred to as the *monitored,* or *project,* directory. The rapid deployment tools then automatically detect added or changed parts of these J2EE artifacts and perform the steps necessary to produce a running application on an application server.

There are two ways to configure the monitored directory, each performing separate and distinct tasks (as shown in Figure 24-6):

- Free-form project
- Automatic application installation project



Automated Application Install
J2EE Apps (EARs)
and/or
EJB JARs
WAR
Client JARs
Module Archives
Add J2EE Applications or Module archives to WRD workspace
Change Detection / Trigger process
WRD
Application - Installed, Restarted, Reinstalled or Uninstalled
V6 Application Server
Application - Installed, Restarted, Reinstalled or Uninstalled
Free-Form Project
Java Source
Servlet, JSP, etc.
Static File (Image, HTML)
Add J2EE Artifacts to WRD workspace - J2EE project structure not required
Change Detection / Trigger process
WRD
User creates individual J2EE artifacts (Java Source, Web resources, etc.)
WRD generates J2EE artifacts and package

Figure 24-6 WebSphere Rapid Deployment modes

With the **free-form project** approach, you can place in a single project directory the individual parts of your application, such as Java source files that represent servlets or enterprise beans, static resources, XML files, and other supported application artifacts. The rapid deployment tools then use your artifacts to automatically place them in the appropriate J2EE project structure, generate any additional required artifacts to construct a J2EE-compliant application, and deploy that application on a target server.

The advantage of using a free-form project is that you do not have to know how to package your application artifacts into a J2EE application. The free-form project takes care of the packaging part for you. The free-form project is suitable when you just want to test something quickly, perhaps write a servlet that performs a task.

The **automatic application installation project** allows you to quickly and easily install, update, and uninstall J2EE applications on a server. If you place EAR files in the project directory, they are automatically deployed to the server. If you delete EAR files from the project directory, the application is uninstalled from the server. If you place a new copy of the same EAR file in the project directory, the application is reinstalled. If you place WAR or EJB JAR files in the automatic application installation project, the rapid deployment tool generates the necessary EAR wrapper and then publishes that EAR file on the server. For RAR files, a wrapper is not created. The standalone RAR files are published to the server.

An automatic application installation project simplifies management of applications and relieves you of the burden of going through the installation panels in the WebSphere Administrative Console or developing wsadmin scripts to automate your application deployment.

The rapid deployment tools can be configured to deploy applications either onto a local or remote WebSphere Application Server.

Note: For more detailed information on WebSphere Rapid Deployment, refer to these sources:

- *WebSphere Application Server V6.1: Planning and Design*, SG24-7305
- *WebSphere Application Server V6.1: Systems Management and Configuration*, SG24-7304
- IBM WebSphere Application Server V6.1 InfoCenter, found at:

  http://publib.boulder.ibm.com/infocenter/ws60help/index.jsp?topic=/com.ibm.websphere.base.doc/info/welcome_base.html

Java and WebSphere class loader

Class loaders are responsible for loading classes, which can be used by an application. Understanding how Java and WebSphere class loaders work is an important element of Application Server configuration needed for the application to work properly after deployment. Failure to set up the class loaders properly will often result in class loading exceptions such as ClassNotFoundException when trying to start the application.

Java class loader

Java class loaders enable the Java virtual machine (JVM) to load classes. Given the name of a class, the class loader should locate the definition of this class. Each Java class must be loaded by a class loader.

When the JVM is started, three class loaders are used:

- Bootstrap class loader: The bootstrap class loader is responsible for loading the core Java libraries (that is, core.jar, server.jar) in the <JAVA_HOME>/lib directory. This class loader, which is part of the core JVM, is written in native code.

> **Note:** Beginning with JDK 1.4, the core Java libraries in the IBM JDK are no longer packaged in rt.jar as was previously the case (and is the case for the Sun JDKs), but instead split into multiple JAR files.

- Extensions class loader: The extensions class loader is responsible for loading the code in the extensions directories (`<JAVA_HOME>/lib/ext` or any other directory specified by the `java.ext.dirs` system property). This class loader is implemented by the `sun.misc.Launcher$ExtClassLoader` class.
- System class loader: The system class loader is responsible for loading the code that is found on `java.class.path`, which ultimately maps to the system CLASSPATH variable. This class loader is implemented by the `sun.misc.Launcher$AppClassLoader` class.

Delegation is a key concept to understand when dealing with class loaders. It states that a custom class loader (a class loader other than the bootstrap, extension, or system class loaders) delegates class loading to its parent before trying to load the class itself. The parent class loader can either be another custom class loader or the bootstrap class loader. Another way to look at this is that a class loaded by a specific class loader can only reference classes that this class loader or its parents can load, but not its children.

The Extensions class loader is the parent for the System class loader. The Bootstrap class loader is the parent for the Extensions class loader. The class loaders hierarchy is shown in Figure 24-7.

If the System class loader has to load a class, it first delegates to the Extensions class loader, which in turn delegates to the Bootstrap class loader. If the parent class loader cannot load the class, the child class loader tries to find the class in its own repository. In this manner, a class loader is only responsible for loading classes that its ancestors cannot load.

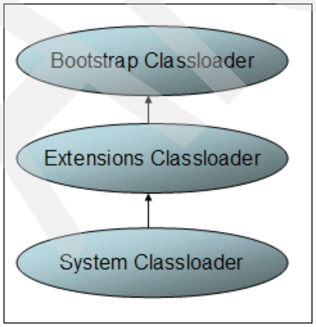


Figure 24-7 Java class loaders hierarchy

WebSphere class loader

It is important to keep in mind when reading the following material on WebSphere class loaders, that each Java Virtual Machine (JVM) has its own setup of class loaders. This means that in a WebSphere environment hosting multiple application servers (JVMs), such as a Network Deployment configuration, the class loaders for the JVMs are completely separated even if they are running on the same physical machine.

WebSphere provides several custom delegated class loaders, as shown in Figure 24-8.

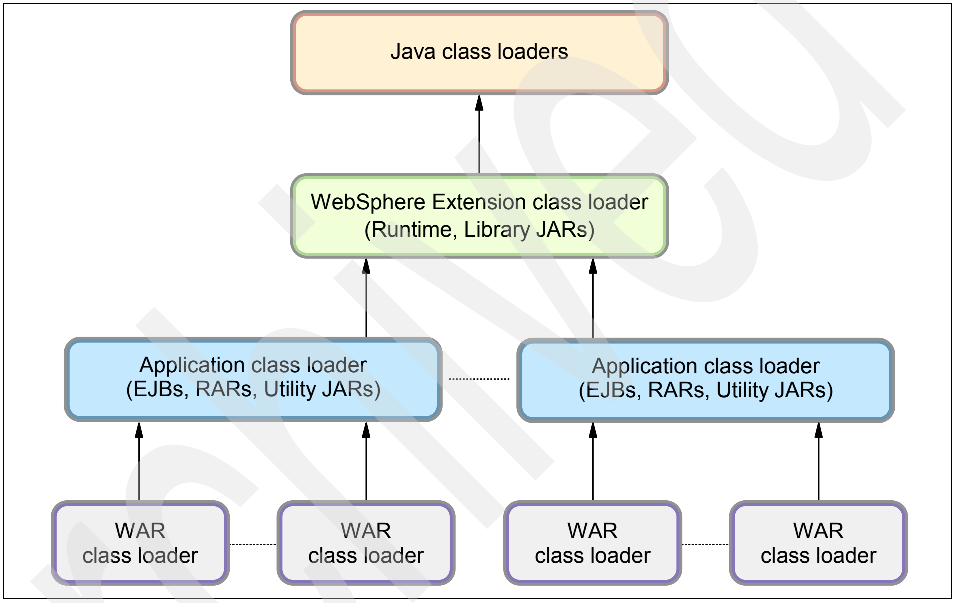


Figure 24-8 WebSphere class loaders hierarchy

In Figure 24-8, the top box represents the Java (Bootstrap, Extension, and System) class loaders. WebSphere does not load much here, just enough to get itself bootstrapped and initialize the WebSphere extension class loader.

WebSphere extensions class loader

The WebSphere extensions class loader is where WebSphere itself is loaded. It uses the following directories to load the required WebSphere classes:

```
<JAVA_HOME>\lib
<WAS_HOME>\classes           (Runtime Class Patches directory, or RCP)
<WAS_HOME>\lib               (Runtime class path directory, or RP)
<WAS_HOME>\lib\ext           (Runtime Extensions directory, or RE)
<WAS_HOME>\installedChannels
```

The WebSphere runtime is loaded by the WebSphere extensions class loader based on the `ws.ext.dirs system` property, which is initially derived from the `WS_EXT_DIRS` environment variable set in the `setupCmdLine.bat` file. The default value of `ws.ext.dirs` is as follows:

```
SET WAS_EXT_DIRS=%JAVA_HOME%\lib;%WAS_HOME%\classes;%WAS_HOME%\lib;
   %WAS_HOME%\installedChannels;%WAS_HOME%\lib\ext;%WAS_HOME%\web\help;
   %ITP_LOC%\plugins\com.ibm.etools.ejbdeploy\runtime
```

The `RCP` directory is intended to be used for fixes and other APARs that are applied to the application server runtime. These patches override any copies of the same files lower in the `RP` and `RE` directories. The `RP` directory contains the core application server runtime files. The bootstrap class loader first finds classes in the `RCP` directory, then in the `RP` directory. The `RE` directory is used for extensions to the core application server runtime.

Each directory listed in the `ws.ext.dirs` environment variable is added to the WebSphere extensions class loaders class path. In addition, every JAR file and/or ZIP file in the directory is added to the class path.

You can extend the list of directories and files loaded by the WebSphere extensions class loaders by setting a `ws.ext.dirs` custom property to the Java virtual machine settings of an application server.

Application and Web module class loaders

J2EE applications consist of five primary elements: Web modules, EJB modules, application client modules, resource adapters (RAR files), and utility JARs. Utility JARs contain code used by both EJBs and/or servlets. Utility frameworks (such as log4j) are a good example of a utility JAR.

EJB modules, utility JARs, resource adapters files, and shared libraries associated with an application are always grouped together into the same class loader. This class loader is called the application class loader. Depending on the application class loader policy, this application class loader can be shared by multiple applications (EAR) or be unique for each application (the default).

By default, Web modules receive their own class loader (a WAR class loader) to load the contents of the `WEB-INF/classes` and `WEB-INF/lib` directories. The default behavior can be modified by changing the application's WAR class loader policy (the default being *Module*). If the WAR class loader policy is set to *Application*, the Web module contents are loaded by the *application class loader* (in addition to the EJBs, RARs, utility JARs, and shared libraries). The application class loader is the parent of the WAR class loader.

The application and the Web module class loaders are reloadable class loaders. They monitor changes in the application code to automatically reload modified classes. This behavior can be altered at deployment time.

Handling JNI code

Due to a JVM limitation, code that has to access native code through a Java Native Interface (JNI) must not be placed on a reloadable class path, but on a static class path. This includes shared libraries for which you can define a native class path, or the application server class path. So if you have a class loading native code through JNI, this class must not be placed on the WAR or application class loaders, but rather on the WebSphere extensions class loader.

It might make sense to break out just the lines of code that actually load the native library into a class of its own and place this class on a static class loader. This way you can have all the other code on a reloadable class loader.

Preparing for the deployment of the EJB application

This section describes the steps required to prepare the environment for the deployment sample. We will use the ITSO Bank enterprise application developed in Chapter 16, “Develop Web applications using EJBs” on page 719, to demonstrate the deployment process.

This section includes the following tasks:

- Reviewing the deployment scenarios
- Installing the prerequisite software
- Importing the sample application project interchange files
- Sample database

Reviewing the deployment scenarios

Now that the Application Developer integration with Application Server is managed the same as a standalone Application Server, the procedure to deploy the ITSO Bank sample application is nearly identical to that of a standalone Application Server.

There are several possible configurations in which our sample can be installed, but we will use the following deployment process in this chapter:

- Deploy ITSO Bank to a separate production IBM Application Server V6.0. This scenario uses two nodes (developer node, application server node).

Installing the prerequisite software

The application deployment sample requires that you have the software mentioned in this section installed. Within the example, you can choose between DB2 Universal Database or Derby as your database server. We used Derby for this chapter and our deployment exercises.

The sample for the working example environment consists of two nodes (see Table 24-2 for product mapping):

- Developer node—This node will be used by the developer to import the sample code and package the application in preparation for deployment.
- Application server node—This node will be used as the target server where the enterprise application will be deployed.

Table 24-2 Product mapping for deployment

Software	Version
Developer node	
Microsoft Windows	XP + Service Pack 2 + Critical fixes and security patches.
IBM Application Developer * Integrated IBM Application Server V6.1	v7.0.0.3
Derby (installed by default)	v10.1
Application server node	
Microsoft Windows	XP + Service Pack 2 + Critical fixes and security patches.
IBM Application Server (Base stand-alone)	v6.1
Derby (installed by default)	v10.1

Note: For detailed information on installing the required software for the sample, refer to Appendix A, “Product installation” on page 1281.

Tip: DB2 Universal Database or Derby

Because Derby is installed by default as part of the Application Server (integrated and stand-alone), we used Derby as the database for this chapter and the deployment exercises. But you can decide to use DB2 Universal Database instead, as we also provide information on using DB2.

Importing the sample application project interchange files

This section describes how to import the project interchange files into Application Developer. The RAD7EJB.zip contains the following projects for the ITSO Bank enterprise application developed in Chapter 16, “Develop Web applications using EJBs” on page 719:

- **RAD7EJBJava**—This is the Java project that contains the model code.
- **RAD7EJB**—This is the EJB project with entity and session beans.
- **RAD7EJBClient**—This is the EJB client project that has the EJB code used by a client.
- **RAD7EJBEAR**—This is the EAR project for the enterprise application; it includes the above modules.

A second project interchange file, RADEJBWeb.zip, contains the Web application that uses the EJBs:

- **RAD7EJBWeb**—This is a Web project (JSP and servlets) for the front-end application.
- **RAD7EJBWebEAR**—This is the EAR project for the Web application. Note that this EAR also references the RAD7EJBClient and RAD7Java projects.

To import the RAD7EJB.zip project interchange file, do these steps:

- Open the J2EE (or Web) perspective Project Explorer view.
- Select **File** → **Import**.
- Select **Other** → **Project Interchange** from the list of import sources and then click **Next**.
- In the Import Projects dialog, click **Browse** for the zip file, navigate to and select the **RAD7EJB.zip** from the C:\7501code\zInterchangeFiles\ejb folder, and click **Open**.
- Click **Select All** to select all projects and then click **Finish**.
- Repeat this sequence for the RAD7EJBWeb.zip file.

Sample database

The ITSO Bank application are based on the ITSOBANK database. Refer to “Setting up the ITSOBANK database” on page 1312 for instructions on how to create the ITSOBANK database. You can either use the DB2 or Derby database. For simplicity we chose to use the built-in Derby database in this chapter. But we have also provided instructions for how to set up the ITSOBANK database using DB2 UDB.

To make it even simpler, we have placed the Derby database as part of the file you have downloaded for this chapter. The `ITSOBANK` folder under the `C:\7501code\Database\Derby` folder constitutes the Derby database. Therefore, the complete path or location of the database is:

```
C:\7501code\Database\Derby\ITSOBANK
```

This value has to be configured for the `databaseName` resource property for the `RAD7DS` data source we will define in the server (see “Configuring the data source in the application server” on page 1128). Therefore, if you configure this location in the data source, you will not have to set up a sample Derby database, as it has already been done for you. Make sure that you test the data source connection.

Packaging the application for deployment

This section describes the steps in preparation for packaging, as well as how to export the enterprise application from Application Developer to an EAR file, which is used to deploy on the application server.

This section includes the following procedures:

- Generating the EJB to RDB mapping (*only if using DB2*)
- Customizing the deployment descriptors
- Removing the enhanced EAR data source
- Generating the deploy code
- Exporting the EAR files

Generating the EJB to RDB mapping

> **Note: This section is only needed when using DB2.** If you are using Derby as your database server, this section is not required. This section describes how to generate the EJB to RDB mapping for the DB2 Universal Database.

The following procedure describes how to generate the EJB to RDB mapping for the enterprise beans when using DB2 for the `ITSOBANK` database.

Create a connection to the ITSOBANK database in DB2

Open the Data perspective. For Derby we already have the `ITSOBANK` connection, created in “Creating a database connection” on page 358. To use DB2 we have to create a connection to the DB2 database:

- Right-click **Connections** and select **New Connection**.

- In the New Connection dialog:
  - Clear **Use default naming convention** and type **ITSOBANKDB2** as the name.
  - Expand **DB2 UDB** → **V8.2** (or the Version that you have installed).
  - Type **ITSOBANK** for the database name.
  - Enter s user ID and password that can access DB2 (for example the user ID used to install DB2).
  - Click **Test Connection** and you receive a successful message.
  - Click **Finish**.

Import the physical data model into the EJB project

The mapping is defined between the entity beans and the physical tables of the database. We have to import the tables into the project do perform the mapping:

- In the J2EE perspective Project Explorer expand **RAD7EJB** → **ejbModule** → **META-INF**.
- Select **File** → **New** → **Other** → **Data** → **Physical Data Model** and click **Next**.
- In the Model File dialog, do these steps and click **Next**:
  - File name: **ITSOBANK DB2 Model**
  - Database: **DB2 UDB**
  - Version: **8.2**
  - Select **Create from reverse engineering**
- Select **Database** for the type of source, and click **Next**.
- Select **Use an existing connection**, **ITSOBANKDB2**, and click **Next**.
- Select the **ITSO** schema, and click **Next**.
- Select **Tables**, clear all other objects, and click **Next**.
- Clear all options (like the Overview diagram), and click **Finish**.
- A folder `META-INF/backends/DB2UDBNT_V82_1` is created.

Create the EJB to RDB mapping

Now we can create the mapping:

- Right-click the **RAD7EJB** project and select **EJB to RDB Mapping** → **Generate Map**.
- Select **Use an existing backend folder**, select the **DB2UDBNT_V82_1** backend folder, and click **Next**.
- Select **Meet-In-The-Middle** and click **Next**.
- Select **Match by Name** and click **Finish**.

- The Mapping editor opens. The mapping is the same for Derby and for Db2. Follow the instructions in “Completing the EJB-to-RDB mapping” on page 768 to complete the mapping.

Customizing the deployment descriptors

Depending on your development process, you can customize your deployment descriptors prior to exporting the enterprise application to an EAR file. For example, you might want to modify the context paths for the target Application Server to which the EAR will be deployed, or change the target database from Derby to DB2 Universal Database.

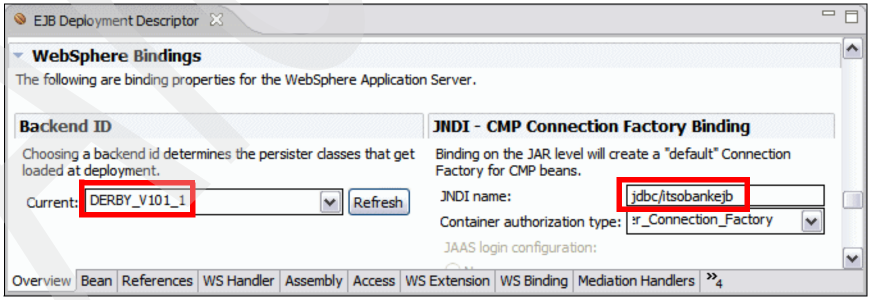
Alternatively, the deployment descriptors of the exported EAR could be modified later using the Application Server Toolkit provided with the Application Server.

> **Note:** For more information, refer to “Preparing for the deployment of the EJB application” on page 1120.

Modify the EJB deployment descriptor for Derby

Do these steps:

- In the Project Explorer view, expand the **RAD7EJB** project.
- Open the **Deployment Descriptor: RAD7EJB** in the Deployment Descriptor editor.
- Scroll down near the bottom to the section named **WebSphere Bindings**.
- Select **DERBY_V101_1** for the Backend ID. Save the EJB Deployment Descriptor (Figure 24-9).



WebSphere Bindings

The following are binding properties for the WebSphere Application Server.

Backend ID

Choosing a backend id determines the persister classes that get loaded at deployment.

Current: DERBY_V101_1

JNDI - CMP Connection Factory Binding

Binding on the JAR level will create a "default" Connection Factory for CMP beans.

JNDI name: jdbc/itsobankejb

Container authorization type: :r_Connection_Factory

JAAS login configuration:

Figure 24-9 Verify the backend ID for the Derby database

Modify the EJB deployment descriptor for DB2 UDB

Do these steps:

- In the Project Explorer view, expand the **RAD7EJB** project.
- Open the **Deployment Descriptor: RAD7EJB** in the Deployment Descriptor editor.
- Scroll down near the bottom to the section named **WebSphere Bindings**.
- Select **DB2UDBNT_V82_1** for the Backend ID. If you have both Derby and DB2 installed, you might want to change the JNDI name, for example, **jdbc/itsobankdb2**. Save the EJB Deployment Descriptor.

Removing the enhanced EAR data source

As explained earlier, the application deployment descriptor contains enhanced EAR information for the JDBC Provider and data source configuration. These settings are useful when running the application within Application Developer.

As we are deploying the application to a remote application server system and because the enhanced EAR data source configuration overrides the administrative console configuration, the enhanced EAR data source settings must be removed.

To remove the enhanced EAR data source settings, do these steps:

- In the Project Explorer view, expand the **RAD7EJBEAR** project and open the **Deployment Descriptor: RAD7EJBEAR** (Figure 24-2 on page 1107).
- Select the **Deployment** tab.
- Select **Derby JDBC Provider (XA)** and click **Remove**.
- Save the deployment descriptor.

Generating the deploy code

Prior to exporting the EAR, we recommend that you generate the deploy code as follows:

- Make sure that you have selected the correct backend folder in the deployment descriptor, either Derby or DB2. Deployment code is generated for the selected mapping (backend folder).
- Deployment is required for EJBs. This can be performed in the IDE now, or when installing the application on the server.
- In the Project Explorer view, right-click the **RAD7EJBEAR** project and select **Prepare for Deployment**.

You can also right-click the **RAD7EJB** project and select **Prepare for Deployment** (the Web modules do not require deployment).

- A number of helper packages and classes are generated in the RAD7EJB project based on the EJB-to-RDB mapping:
  - `com.ibm.ws.ejbdeploy.JRAD7EJB.DB2UBDNT_V82_1` (or Derby)
  - `itso.rad7.bank.model.ejb.websphere_deploy` (shared code)
  - `itso.rad7.bank.model.ejb.websphere_deploy.DB2UDBNT_V82_1` (or Derby)

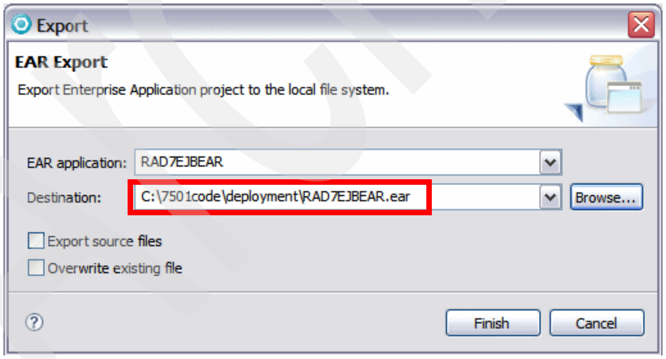
> **Note:** There are many warnings for the `RAD7EJB` project in the Problems view. These can be safely ignored, or you can filter the view and remove warnings from the list.

Exporting the EAR files

We have two enterprise applications for the EJB sample in the workspace: `RAD7EJBEAR` and `RAD7EJBWebEAR`. We have to export both of these enterprise projects as EAR files.

To export the enterprise applications from Application Developer to EAR files, do these steps:

- In the Project Explorer view, right-click **RAD7EJBEAR** and select **Export** → **EAR file**.
- In the EAR Export dialog, enter the destination path (for example, `C:\7501code\deployment\RAD7EJB.ear`) and click **Finish** (Figure 24-10).



Export Enterprise Application project to the local file system.

EAR application: RAD7EJBEAR

Destination: C:\7501code\deployment\RAD7EJBEAR.ear

☐ Export source files

☐ Overwrite existing file

Figure 24-10 Choices for exporting an EAR file

- Repeat the export for the project `RAD7EJBWebEAR`.

Filtering the content of an EAR

It is possible to exclude certain files from the exported EAR, for example, diagrams and Javadoc.

To filter (exclude) files from an EAR, do these steps:

- Right-click a Java, Web, or EJB project and select **Properties**.
- On the **Source** tab, expand the source folder (typically **src**), select **Excluded** and click **Edit**.
- In the Inclusion and Exclusion Patterns dialog, click **Add** to exclude files of a given extension (for example, **/*.dnx to exclude diagrams).
- Click **Finish** and then **OK**.

Manual deployment of enterprise applications

Both enterprise applications have been packaged as an EAR file and can be deployed to the application server. This section describes the steps required to configure the target Application Server and install the two enterprise applications.

> **Note:** You can either use the `EAR` files exported in the previous section for deployment to the target Application server, or alternatively use the solution `EAR` files from the `C:\7501code\jython` directory of the sample code.

Configuring the data source in the application server

The data source for the application server can be created in several ways, including these:

- Enhanced EAR—This is the ideal option for deploying the application to the Integrated Application Server in Application Developer, but because we are deploying to the stand-alone server, we are not using the enhanced EAR functionality.
- Scripting using wsadmin command line interface—For details, refer to the Application Server V6.1 InfoCenter. This is also covered in the second part of this chapter, which is about the automated deployment using Jython-based wsadmin scripting.
- WebSphere Administrative Console—For our example, we create and configure the data source for the ITSO Bank application using the administrative console of the target stand-alone Application Server V6.1.

The high-level configuration steps to configure the data source within Application Server for the ITSO Bank application sample are as follows:

- Start the application server.
- If you configured the stand-alone application server as a Windows service, you can start the server by starting its service.
- Configure J2C authentication data.
- Configure the JDBC provider.
- Create the data source.

Start the application server

Ensure that the application server where you want to deploy the applications is started. You can use any WebSphere Application Server, stand-alone or configured as a profile in Application Developer.

If the server is not running, start the server in any of these ways:

- Use the Windows start menu. For example, select **Start** → **Programs** → **IBM WebSphere** → **Application Server V6** → **Profiles** → **default** → **Start server**.
- If you followed the instructions in Chapter 20, “Servers and server configuration” on page 963 and installed a second WebSphere profile, you can start that server from Application Developer.
- An application server can also be started using the **`startServer server1`** command in the `bin` directory where the server profile is installed, for example:

  ```
  C:\Program Files\IBM\WebSphere\AppServer\profiles\default\bin
  ```

  ```
  <RAD_HOME>\runtimes\base_v61\profiles\AppSrv02\bin
  ```

- If you configured the stand-alone application server as a Windows service, you can start the server by starting its service.

> **Tip:** You can also use the WebSphere Application Server v6.1 test environment to go through the enterprise application installation dialogs.
>
> However, make sure that the `RAD7EJBEAR` and `RAD7EJBWebEAR` applications are removed from the server (right-click the server and select **Add and Remove Projects**).

> **Note:** To verify that the server has started properly, you can look for the message **`Server server1 open for e-business`** in the `SystemOut.log` found in the `<was_profile_root>\logs\server1` directory.

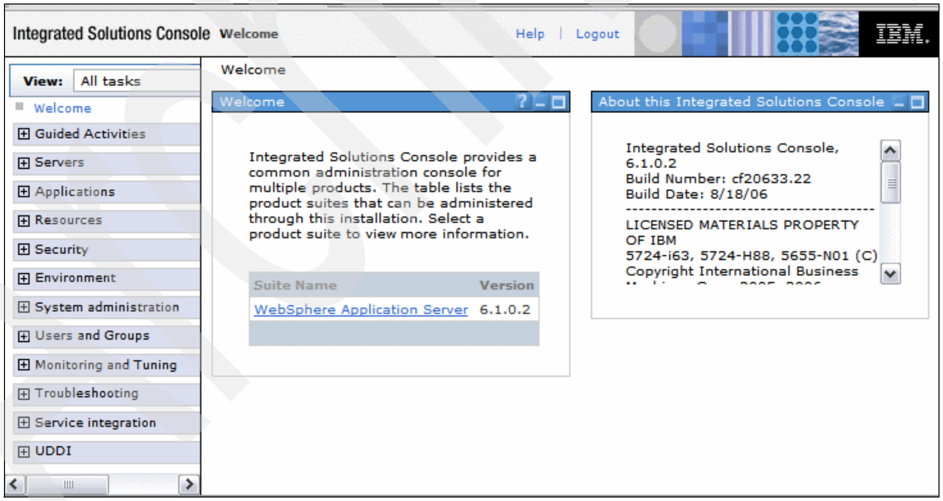
Start the administrative console

We use the WebSphere Administrative Console to define the data source and install the applications. When using a server from Application Developer, you can start the administrative console by right-clicking the server and selecting **Run administrative console**. However, we recommend to perform the configuration in an external browser to simulate a real deployment environment.

- To start the administrative console, open a Web browser (Internet Explorer, Firefox) and use this URL:

```
http://<hostname>:<port>/ibm/console
http://localhost>:9060/ibm/console          test environment
http://localhost>:9062>/ibm/console         alternate profile
```

- For a stand-alone server, you can also select **Start** → **Programs** → **IBM WebSphere** → **Application Server V6** → **Profiles** → **default** → **Start administrative console**.
- Click **Login** (without security, no user ID is required). See Figure 24-11.



Integrated Solutions Console provides a common administration console for multiple products. The table lists the product suites that can be administered through this installation. Select a product suite to view more information.

Suite Name	Version
WebSphere Application Server	6.1.0.2

Integrated Solutions Console, 6.1.0.2
Build Number: cf20633.22
Build Date: 8/18/06

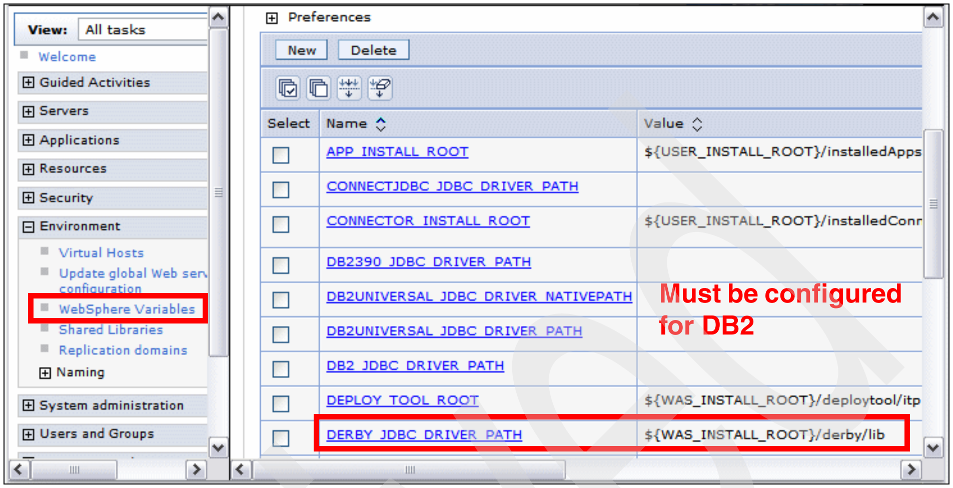
LICENSED MATERIALS PROPERTY OF IBM
5724-i63, 5724-H88, 5655-N01 (C) Copyright International Business

Figure 24-11 Welcome page of the administrative console

Create the JDBC driver variable

If you are using Derby, verify that the WebSphere variable for the Derby JDBC driver (`DERBY_JDBC_DRIVER_PATH`) is defined.

- Select **Environment** → **WebSphere Variables**. Verify that `DERBY_JDBC_DRIVER_PATH` is defined (Figure 24-12).



Select	Name	Value
□	APP_INSTALL_ROOT	${USER_INSTALL_ROOT}/installedApps
□	CONNECTJDBC_JDBC_DRIVER_PATH	
□	CONNECTOR_INSTALL_ROOT	${USER_INSTALL_ROOT}/installedConr
□	DB2390_JDBC_DRIVER_PATH	
□	DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH	
□	DB2UNIVERSAL_JDBC_DRIVER_PATH	
□	DB2_JDBC_DRIVER_PATH	
□	DEPLOY_TOOL_ROOT	${WAS_INSTALL_ROOT}/deploytool/itp
□	DERBY_JDBC_DRIVER_PATH	${WAS_INSTALL_ROOT}/derby/lib

Must be configured for DB2

*Figure 24-12 WebSphere Variable: DERBY_JDBC_DRIVER _PATH *

> **Note for DB2:** The corresponding variable (`DB2UNIVERSAL_JDBC_DRIVER_PATH`) for DB2 is already defined, but not configured. You have to set the value to the DB2 folder, for example `C:\Program Files\SQLLIB\java`.

Configure J2C authentication data

> **Note**: This step is required for DB2 UDB and optional for Derby.

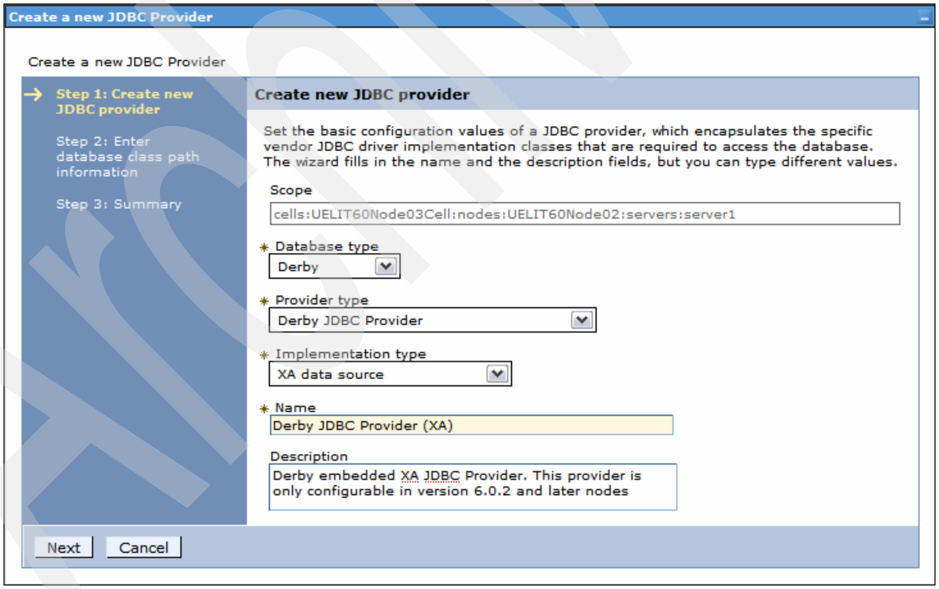
This section describes how to configure the J2C authentication data, that is, the database login and password to connect to the database system.

- Select **Security** → **Secure administration, applications, and infrastructure**.
- Under the Authentication properties, select **Java Authentication and Authorization Service** → **J2C authentication data**.
- Click **New**, then specify these values (for DB2):
  - Alias: `dbuser`
  - User ID: `db2admin` (or the administrative password used to install)
  - Password: `<password>`
- Click **Save** to the master configuration when prompted.

Configure the JDBC provider

This section describes how to configure the JDBC provider for the selected database type. The following procedure demonstrates how to configure the JDBC Provider for Derby, with notes on how to do the equivalent for DB2 Universal Database:

- Select **Resources** → **JDBC** → **JDBC Providers**.
- Select the server scope: **Node=<userid>Node<xx>, Server=server1**
- Click **New**.
- In the Create new JDBC Providers page (Figure 24-13), do these steps:
  - For Database type, select **Derby** (or **DB2**).
  - For JDBC Provider, select **Derby JDBC Provider** (or **DB2 Universal JDBC Driver Provider**).
  - For Implementation type, select **XA data source** (same for DB2).
  - Accept the default name of Derby JDBC Provider (XA) (or DB2 Universal JDBC Driver Provider (XA))
  - Click **Next**.



Create a new JDBC Provider

Create a new JDBC Provider

→ Step 1: Create new JDBC provider

Step 2: Enter database class path information

Step 3: Summary

Create new JDBC provider

Set the basic configuration values of a JDBC provider, which encapsulates the specific vendor JDBC driver implementation classes that are required to access the database. The wizard fills in the name and the description fields, but you can type different values.

Scope: cells:UELIT60Node03Cell:nodes:UELIT60Node02:servers:server1

* Database type: Derby

* Provider type: Derby JDBC Provider

* Implementation type: XA data source

* Name: Derby JDBC Provider (XA)

Description: Derby embedded XA JDBC Provider. This provider is only configurable in version 6.0.2 and later nodes

Figure 24-13 Create a JDBC provider

- For Derby, Step 2 is skipped (the class path is set correctly), and in Step 3: Summary page, click **Finish**.

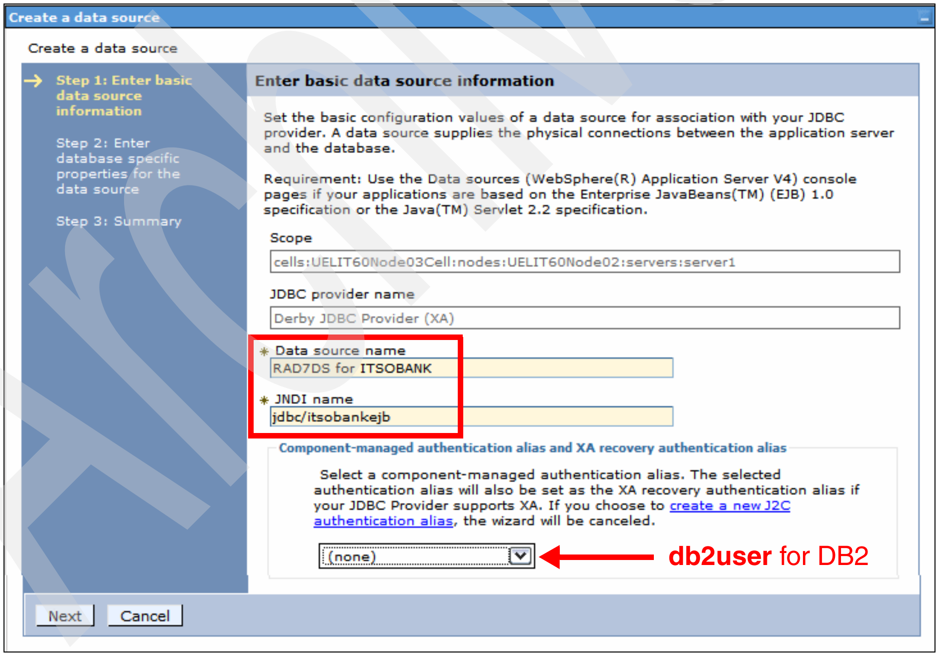
- For DB2, Step 2 is displayed, and you can enter the class path for the JDBC drivers:
  - `${DB2UNIVERSAL_JDBC_DRIVER_PATH}`
  - `${DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH}`

  These values will be saved in the corresponding WebSphere variables. Then click **Next** and click **Finish** in the Step 3: Summary page.
- Click **Save** to the master configuration.

Create the data source

We create the data source for the `ITSOBANK` database for the selected JDBC provider:

- Select **Derby JDBC Provider (XA)** for Derby or **DB2 Universal JDBC Driver Provider (XA)** for DB2.
- Under Additional Properties (right-hand side of page), click **Data sources**.
- In the Data source page, click **New**.
- Enter the basic configuration for the new Data source (Figure 24-14).



Create a data source

Create a data source

→ Step 1: Enter basic data source information
Step 2: Enter database specific properties for the data source
Step 3: Summary

Enter basic data source information

Set the basic configuration values of a data source for association with your JDBC provider. A data source supplies the physical connections between the application server and the database.

Requirement: Use the Data sources (WebSphere(R) Application Server V4) console pages if your applications are based on the Enterprise JavaBeans(TM) (EJB) 1.0 specification or the Java(TM) Servlet 2.2 specification.

Scope: cells:UELIT60Node03Cell:nodes:UELIT60Node02:servers:server1

JDBC provider name: Derby JDBC Provider (XA)

* Data source name: RAD7DS for ITSOBANK

* JNDI name: jdbc/itsobankejb

Component-managed authentication alias and XA recovery authentication alias

Select a component-managed authentication alias. The selected authentication alias will also be set as the XA recovery authentication alias if your JDBC Provider supports XA. If you choose to create a new J2C authentication alias, the wizard will be canceled.

(none)

db2user for DB2

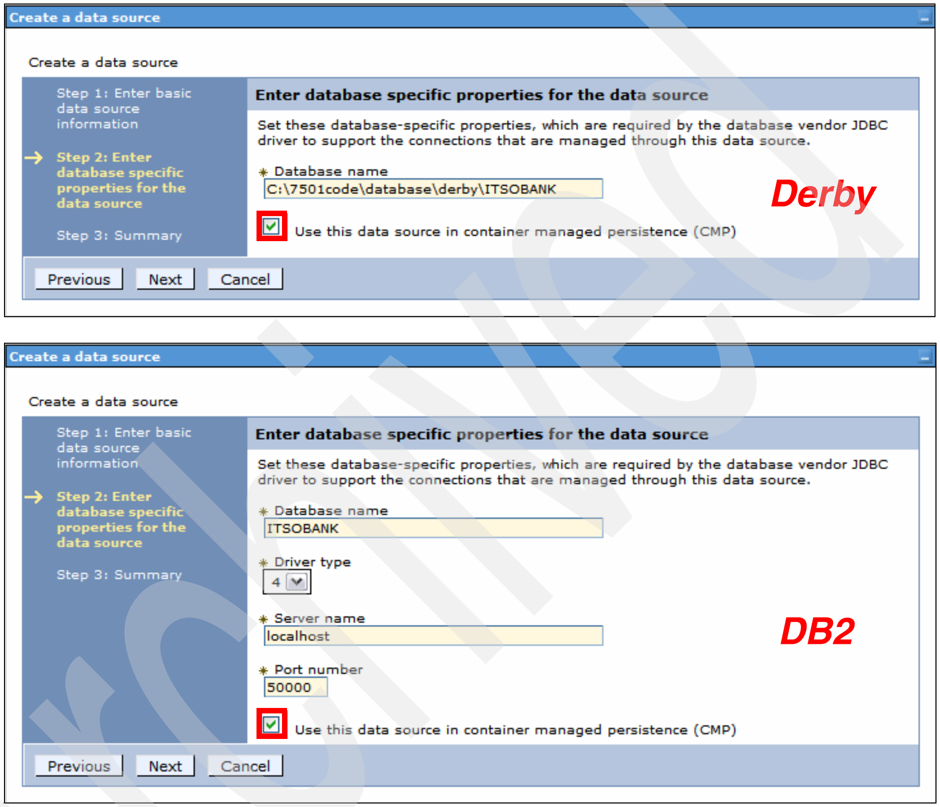
Figure 24-14 Basic configuration for the data source

  - Name: **RAD7DS for ITSOBANK** (this can be anything).

- JNDI name: **jdbc/itsobankejb** (this must match the JNDI name given in the EJB deployment descriptor (refer to Figure 24-9 on page 1125).

  For DB2, enter **jdbc/itsobankdb2** as JNDI name and select **db2user** for the authentication alias.

► In the next page (Figure 24-15), enter the database name.



Create a data source

Create a data source

Step 1: Enter basic data source information

→ Step 2: Enter database specific properties for the data source

Step 3: Summary

Enter database specific properties for the data source

Set these database-specific properties, which are required by the database vendor JDBC driver to support the connections that are managed through this data source.

* Database name
C:\7501code\database\derby\ITSOBANK

☑ Use this data source in container managed persistence (CMP)

Create a data source

Create a data source

Step 1: Enter basic data source information

→ Step 2: Enter database specific properties for the data source

Step 3: Summary

Enter database specific properties for the data source

Set these database-specific properties, which are required by the database vendor JDBC driver to support the connections that are managed through this data source.

* Database name
ITSOBANK

* Driver type
4

* Server name
localhost

* Port number
50000

☑ Use this data source in container managed persistence (CMP)

Figure 24-15 Enter the database path or name

- **Derby**: Make sure that you enter the complete path of the database file. For example, in our case this value is:

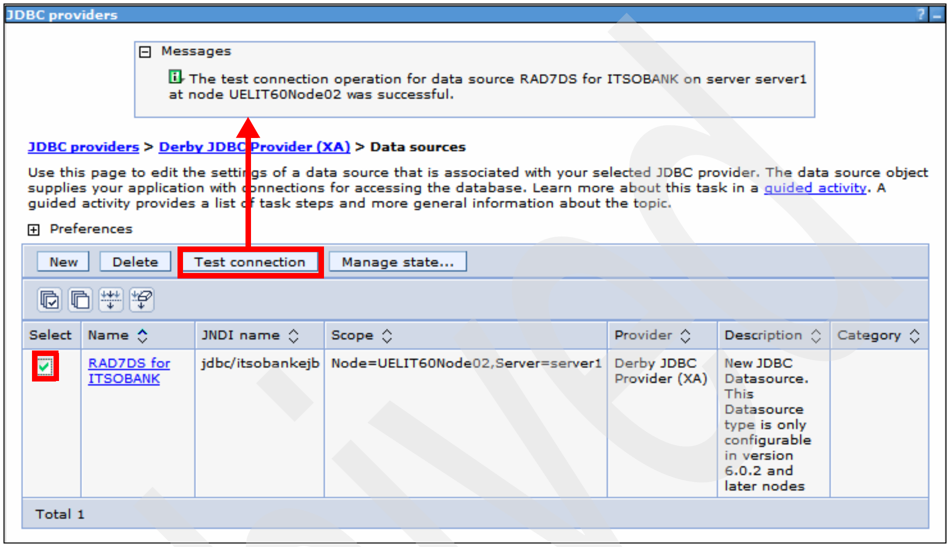
  ```
  C:\7501code\database\derby\ITSOBANK
  ```

- **DB2**: Enter **ITSOBANK** as the database name, the driver type (**4**), the host name of the server, and the connection port (default is 5000).

- Select **Use this data source in container managed persistence (CMP)**. This is required when using entity EJBs. Click **Next**.

► In the summary page, verify the configuration information you entered for the data source and click **Finish**.

- Click **Save** to the master configuration.
- Verify the database connection for the new data source (Figure 24-16):
  - Select the data source (check box)
  - Click **Test connection** and a message indicates success or failure.



JDBC providers

⊟ Messages

The test connection operation for data source RAD7DS for ITSOBANK on server server1 at node UELIT60Node02 was successful.

JDBC providers > Derby JDBC Provider (XA) > Data sources

Use this page to edit the settings of a data source that is associated with your selected JDBC provider. The data source object supplies your application with connections for accessing the database. Learn more about this task in a guided activity. A guided activity provides a list of task steps and more general information about the topic.

⊞ Preferences

Select	Name	JNDI name	Scope	Provider	Description	Category
☑	RAD7DS for ITSOBANK	jdbc/itsobankejb	Node=UELIT60Node02,Server=server1	Derby JDBC Provider (XA)	New JDBC Datasource. This Datasource type is only configurable in version 6.0.2 and later nodes	

Total 1

Figure 24-16 Test connection of the data source

> **Tip:** If the connection fails, make sure that no connection is active from Application Developer (in the Data perspective disconnect from `ITSOBANK`).

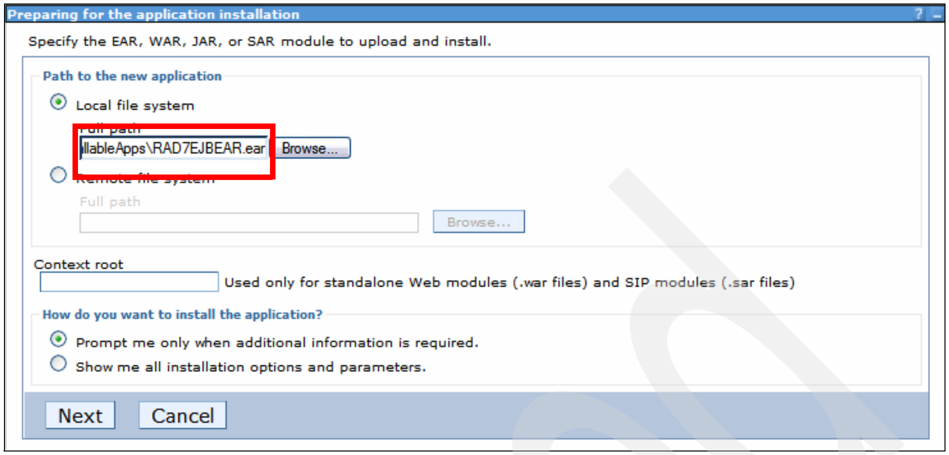
Installing the enterprise applications

To install the two enterprise applications to the target application server, do these steps:

- Copy the `RAD7EJBEAR.ear` and `RAD7EJBWebEAR.ear` files from the developer node (where you exported the files from Application Developer) to the application server node, typically into the `installableApps` directory:

  ```
  <AppServer_HOME>/installableApps
  ```

- In the WebSphere Administrative Console select **Applications** → **Install New Application**.
- Enter the following items (Figure 24-17):
  - Select **Local file system**.
  - Specify path: `C:\.......\installableApps\RAD7EJBEAR.ear`



Preparing for the application installation

Specify the EAR, WAR, JAR, or SAR module to upload and install.

Path to the new application

● Local file system

Full path

llableApps\RAD7EJBEAR.ear Browse...

○ Remote file system

Full path

Browse...

Context root

Used only for standalone Web modules (.war files) and SIP modules (.sar files)

How do you want to install the application?

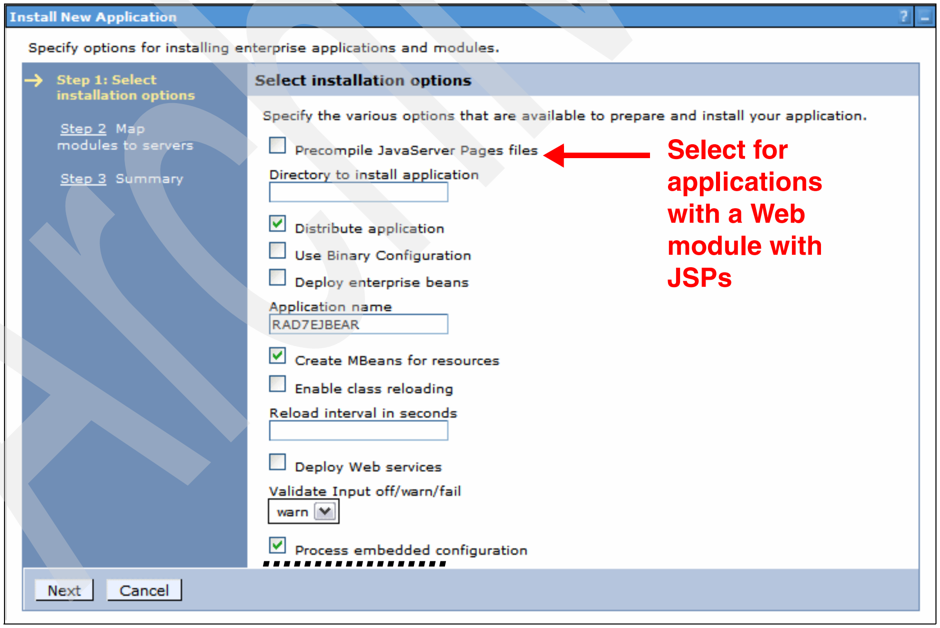
● Prompt me only when additional information is required.

○ Show me all installation options and parameters.

Next Cancel

Figure 24-17 Enterprise application installation: Specify the path to the EAR file

- ► Click **Next** and the installation wizard starts.
- ► In the Select installation options page, accept the default values and click **Next** (Figure 24-18).



Install New Application

Specify options for installing enterprise applications and modules.

→ Step 1: Select installation options

Step 2 Map modules to servers

Step 3 Summary

Select installation options

Specify the various options that are available to prepare and install your application.

□ Precompile JavaServer Pages files

Select for applications with a Web module with JSPs

Directory to install application

☑ Distribute application

□ Use Binary Configuration

□ Deploy enterprise beans

Application name

RAD7EJBEAR

☑ Create MBeans for resources

□ Enable class reloading

Reload interval in seconds

□ Deploy Web services

Validate Input off/warn/fail

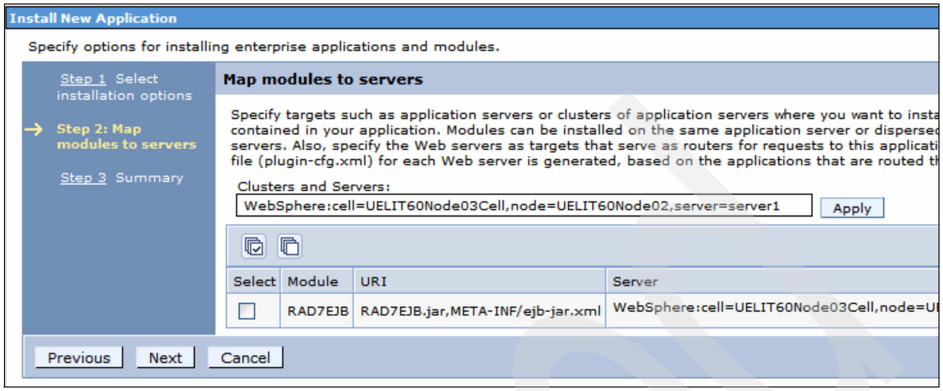
warn

☑ Process embedded configuration

Next Cancel

Figure 24-18 Enterprise application installation: Installation options

- In the Map modules to servers page, accept the default values and click **Next** (Figure 24-19).



Install New Application

Specify options for installing enterprise applications and modules.

Step 1 Select installation options

→ Step 2: Map modules to servers

Step 3 Summary

Map modules to servers

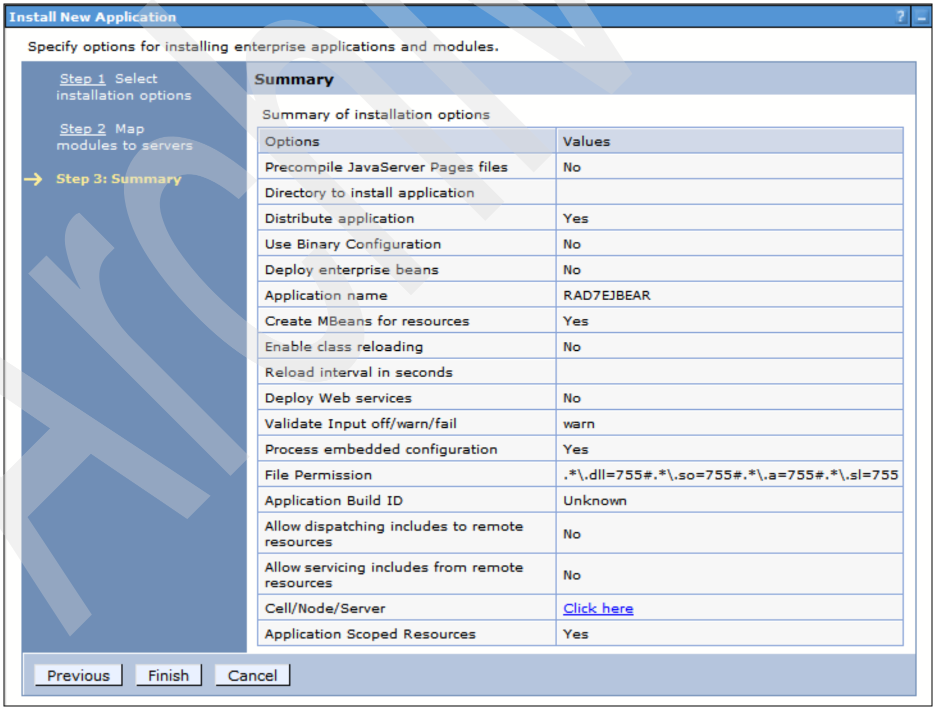
Specify targets such as application servers or clusters of application servers where you want to insta
contained in your application. Modules can be installed on the same application server or disperse
servers. Also, specify the Web servers as targets that serve as routers for requests to this applicati
file (plugin-cfg.xml) for each Web server is generated, based on the applications that are routed th

Clusters and Servers:
WebSphere:cell=UELIT60Node03Cell,node=UELIT60Node02,server=server1 Apply

Select	Module	URI	Server
□	RAD7EJB	RAD7EJB.jar,META-INF/ejb-jar.xml	WebSphere:cell=UELIT60Node03Cell,node=U

Figure 24-19 Enterprise application installation: Map modules to the application servers

- In the summary page, verify the configuration information for the new enterprise application and click **Finish** to confirm (Figure 24-20).



Install New Application

Specify options for installing enterprise applications and modules.

Step 1 Select installation options

Step 2 Map modules to servers

→ Step 3: Summary

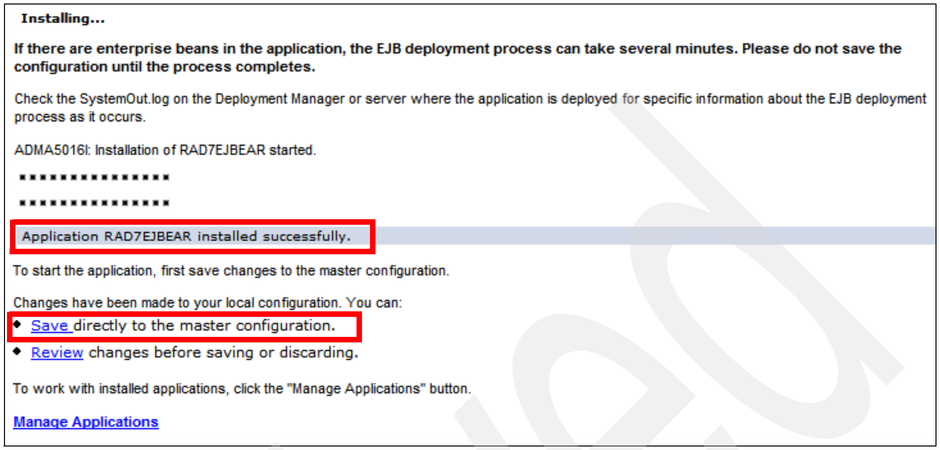
Summary

Summary of installation options

Options	Values
Precompile JavaServer Pages files	No
Directory to install application	
Distribute application	Yes
Use Binary Configuration	No
Deploy enterprise beans	No
Application name	RAD7EJBEAR
Create MBeans for resources	Yes
Enable class reloading	No
Reload interval in seconds	
Deploy Web services	No
Validate Input off/warn/fail	warn
Process embedded configuration	Yes
File Permission	.*\.dll=755#.*\.so=755#.*\.a=755#.*\.sl=755
Application Build ID	Unknown
Allow dispatching includes to remote resources	No
Allow servicing includes from remote resources	No
Cell/Node/Server	Click here
Application Scoped Resources	Yes

Figure 24-20 Summary page for the Install Application wizard

- You will see a number of messages, concluded by successful installation (Figure 24-21).



Installing...

If there are enterprise beans in the application, the EJB deployment process can take several minutes. Please do not save the configuration until the process completes.

Check the SystemOut.log on the Deployment Manager or server where the application is deployed for specific information about the EJB deployment process as it occurs.

ADMA5016I: Installation of RAD7EJBEAR started.

```
***************
***************
```

Application RAD7EJBEAR installed successfully.

To start the application, first save changes to the master configuration.

Changes have been made to your local configuration. You can:

- Save directly to the master configuration.
- Review changes before saving or discarding.

To work with installed applications, click the "Manage Applications" button.

Manage Applications

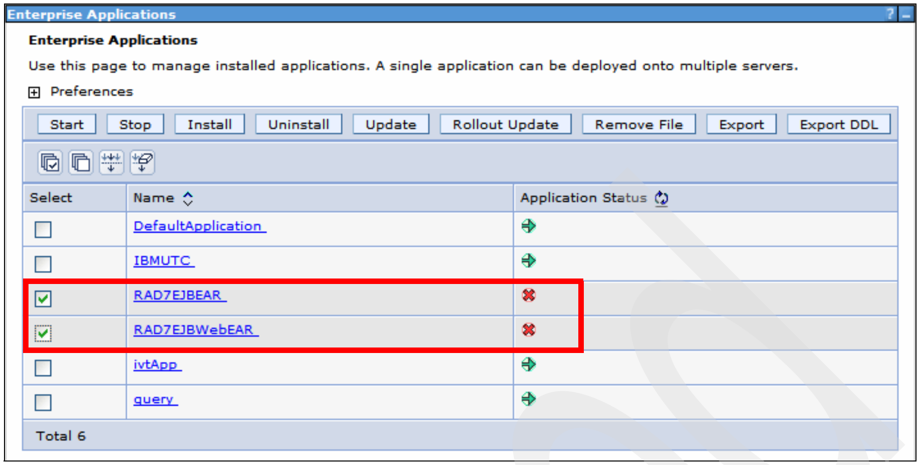
Figure 24-21 Application RAD7EJBEAR installed successfully

- Click **Save** to the master configuration.
- Repeat the installation steps to install the `RAD7EJBWebEAR.ear` enterprise application:
  - In the installation options page, select **Precompile JavaServer Pages files**.
  - The rest of the steps are the same.

Starting the enterprise applications

Do these steps:

- In the administrative console, select **Applications** → **Enterprise Applications**.
- Select the **RAD7EJBEAR** and **RAD7EJBWebEAR** applications and click **Start** (Figure 24-22).
- The status for the applications changes to a green arrow, and two messages about the successful start are displayed at the top.



Enterprise Applications

Enterprise Applications

Use this page to manage installed applications. A single application can be deployed onto multiple servers.

⊞ Preferences

Select	Name	Application Status
□	DefaultApplication	➜
□	IBMUTC	➜
☑	RAD7EJBEAR	✖
☑	RAD7EJBWebEAR	✖
□	ivtApp	➜
□	query	➜
Total 6		

Figure 24-22 Start the newly deployed Enterprise application

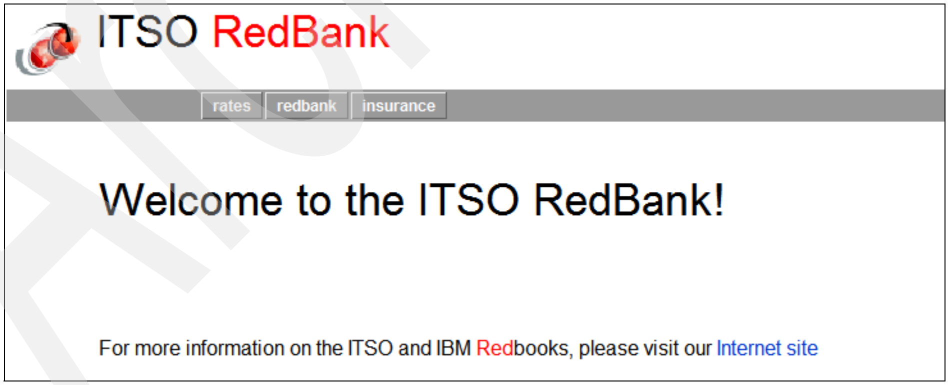
Verifying the application after manual installation

To verify that the ITSO Bank sample is deployed and working properly, do these steps:

- Enter the following URL in a browser to access the ITSO Bank application:

```
http://<hostname>:9080/RAD7EJBWeb/               stand-alone server
http://localhost:9081/RAD7EJBWeb/                WebSphere profile
```

- The ITSO RedBank home page is displayed (Figure 24-23).

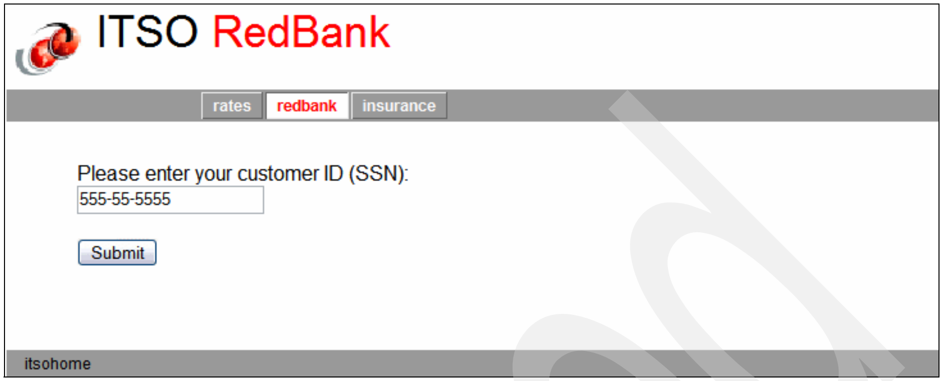


Welcome to the ITSO RedBank!

For more information on the ITSO and IBM Redbooks, please visit our Internet site

Figure 24-23 ITSO RedBank: Home page

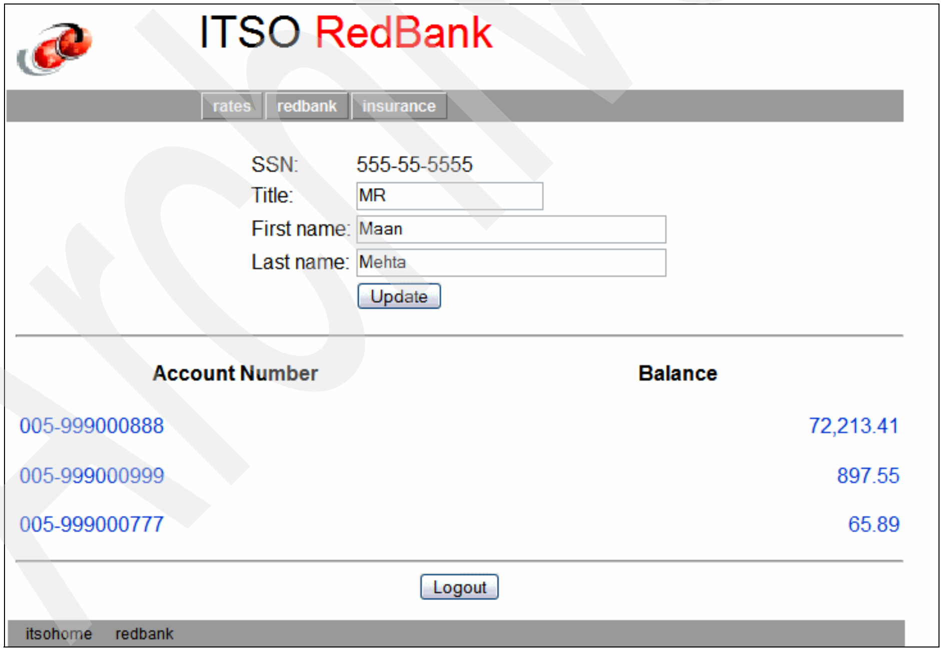
- Click **redbank** and the login page is displayed. Enter a customer ID, for example, `555-55-5555`, and click **Submit** (Figure 24-24).



Please enter your customer ID (SSN):
555-55-5555

Figure 24-24 ITSO RedBank: Login page

- The accounts page lists the accounts of the customer with their balance. (Figure 24-25).



SSN:	555-55-5555
Title:	MR
First name:	Maan
Last name:	Mehta

Account Number	Balance
005-999000888	72,213.41
005-999000999	897.55
005-999000777	65.89

Figure 24-25 ITSO RedBank: Accounts page

- This concludes our testing, though you can further experiment with the application:
  - Update the customer name or title.
  - Click on one of the account numbers to get the account maintenance page.
  - Submit banking transactions, such as deposit, withdraw, and transfer.
  - List the transactions.
  - Logout.

Uninstalling the application

We also want to show deployment using automation scripts, so after testing we uninstall the applications. This is necessary if you want to use the same server for the automation scripts.

- In the WebSphere Administrative Console, select **Applications** → **Enterprise Applications**.
- Select the two **RAD7EJBxxx** applications and click **Stop**.
- Select the two **RAD7EJBxxx** applications and click **Uninstall**.
- Wait for the uninstall successful messages, then click **Save**.

Automated deployment using Jython based wsadmin scripting

In this section we will introduce you to the WebSphere scripting client called *wsadmin*, and the new scripting language used in the client, called *Jython*.

WebSphere Application Server's administration model is based on the Java Management Extensions (JMX) framework. JMX enables you to wrap hardware and software resources in Java and expose them in a distributed environment. WebSphere's administrative services provides functions that use the JMX interfaces to manipulate the application server configuration, which is stored in a XML based repository in the server's file system. Application Server provides the following tools that aid in administration of its configuration:

- WebSphere Administrative Console—A Web application.
- Command-line commands—These executable commands can be found in the `<was_install_root>/bin` folder and also in the `<was_profile_root>/bin` folder.

- wsadmin scripting client—A command-line interface. Scripts can be used to automate the administration of multiple application servers and nodes.
- Thin client—A lightweight runtime package that enables you to run the wsadmin tool or a standalone administrative Java program remotely.

Overview of wsadmin

The wsadmin tool is a scripting client that has a command-line interface. It is targeted towards advanced administrators. It provides extra flexibility that is available through the Web based administrative console and helps make the administration much quicker. It is primarily used to automate administrative activities that can consist of several administrative commands and need to be executed repetitively.

The wsadmin client uses the Bean Scripting Framework (BSF). The BSF allows using a variety of scripting languages. Prior to WebSphere Application Server V6, only one scripting language was supported, Java Command Language (Jacl). Application Server now supports two languages: Jacl and Jython (or jpython). Jython is the strategic direction. New tools in WebSphere Application Server V6.1, Application Server Toolkit V6.1, and Rational Application Developer V7 are available to help create scripts using Jython. A Jacl-to-Jython migration tool is included with the Application Server.

There are five wsadmin objects that are available to use in the scripts:

- **AdminControl**— This object is used to run operational commands.
- **AdminConfig**—This object is used to create or modify Application Server configurational elements.
- **AdminApp**—This object is used to administer applications.
- **AdminTask**—This object is used to run administrative commands.
- **Help**—This object is used to obtain general help.

Overview of Jython

Jython is an implementation of the Python language. The wsadmin tool uses Jython V2.1. The J in Jython represents its Java-like syntax. But Jython is also quite different than Java or C++ syntax. Though like Java, Jython is a typed, case-sensitive, and object-oriented language, but unlike Java it is an indentation based language, which means that it does not have any mandatory statement termination characters (like a semi-colon in Java), and code blocks are specified by indentation. To begin a code block, you have to indent in, and to end a block, indent out. Statements that expect an indentation level end in a colon (`:`).

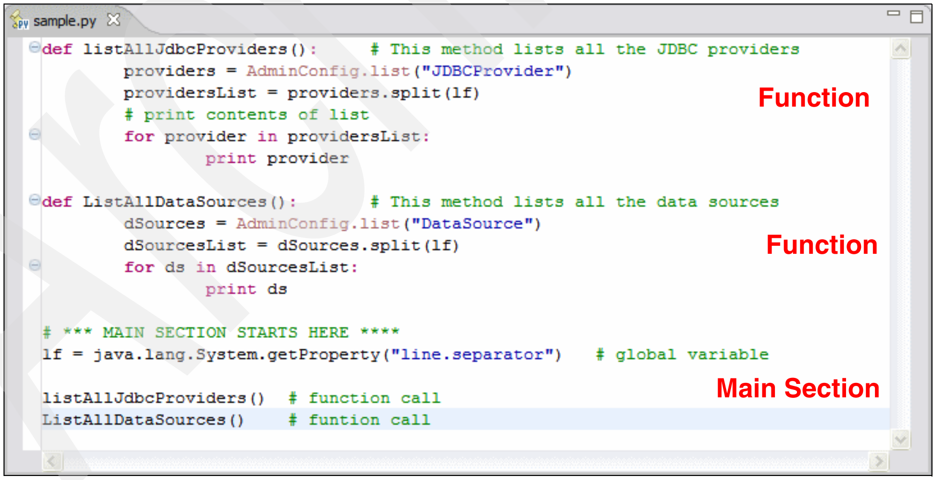
Functions are declared with the def keyword, and comments start with a pound sign (#). Example 24-1 shows a Jython code snippet.

Example 24-1 Jython snippet

```
def listAllApps # This function lists all application
   apps = AdminApp.list()
   if len(apps) == 0:
      print "Inside if block - No enterprise applications"
   else:
      print "Inside the else block"
      appsList = apps.split(lSep)
      for app in appsList:
         print app

# This is a comment - Main section starts here
lSep = java.lang.System.getProperty('line.separator')
listAllApps  # call the listAllApps function defined above
```

Structure of a Jython script

A Jython script usually consists of method definitions and a main section. The outline view of the Application Developer lists the methods of the script file for easy code navigation. The main section is at the end of the Jython script and consists of declaration of global variables, which can be used across all the methods. The main section then calls functions (Figure 24-26).



```
def listAllJdbcProviders():      # This method lists all the JDBC providers
        providers = AdminConfig.list("JDBCProvider")
        providersList = providers.split(lf)
        # print contents of list
        for provider in providersList:
                print provider

def ListAllDataSources():        # This method lists all the data sources
        dSources = AdminConfig.list("DataSource")
        dSourcesList = dSources.split(lf)
        for ds in dSourcesList:
                print ds

# *** MAIN SECTION STARTS HERE ****
lf = java.lang.System.getProperty("line.separator")    # global variable

listAllJdbcProviders()   # function call
ListAllDataSources()     # funtion call
```

Figure 24-26 Structure of a Jython script

Developing a Jython script to deploy the ITSO Bank

In this section we follow a step by step process and complete a Jython script that deploys the RAD7EJBWebAEAR and RAD7EJBEAR files as enterprise applications to a target V6.1 application server. Here are the steps that have to be executed:

- Create a JDBC provider.
- Create and configure a data source.
- Install the ITSO Bank Web application.
- Install the ITSO Bank EJB application.
- Start both applications.

> **Note:** We also have a section on Jython in Chapter 20, “Servers and server configuration” in “Developing automation scripts” on page 993.

Preparation

In this section we create a Jython project and a Jython script in Application Developer:

- The RAD7EJBEAR.ear and RAD7EJBWebEAR.ear files are available in the C:\7501code\deployment folder (where you exported the files), or you can use the solution files from C:\7501code\jython.
- Create a Jython project in Application Developer. If you followed the instructions in Chapter 20, “Servers and server configuration” on page 963, you already have this project, otherwise you can create it as follows:
  - Click **File** → **New** → **Project** and the New Project wizard opens. Select **Jython** → **Jython Project** and click **Next**.
  - For the Project name, type **RAD7Jython** and click **Finish**.
- Create a Jython script to deploy the ITSO Bank application:
  - Select **File** → **New** → **Other** → **Jython** → **Jython Script File** and click **Next**.
  - In the Parent folder field, specify /RAD7Python, and for the File name, type **deployITSOBankApp.py**.
  - Click **Finish** to create the Jython script file.
- In the Servers view, decide which server to use for testing. You can use the test environment or the server profile you defined in Chapter 20, “Servers and server configuration” on page 963.

Main section of the script

Let us define our global variables first.

- Open the Jython script (if you closed it).
- Add the following four global variables:

```
global AdminConfig
global AdminControl
global AdminApp
global AdminTask
```

- Define a global variable for the deployment server node name and the deployment server:

```
nodeName = AdminControl.getNode()
srvrinfo=AdminConfig.list('Server')
srvr=AdminConfig.showAttribute(srvrinfo, 'name')
```

- Define a global variable for the name of the new JDBC provider, the data source, and the database:

```
jdbcProv = "ITSO Derby JDBC Provider (XA)"
dataSourceName = "RAD7JythonDS"
jndiDS = "jdbc/itsobankejb"
dbasename="C:/7501code/database/derby/ITSOBANK"
```

- Define a global variable to store the name of the new connection factory (used for EJB container-managed persistence):

```
cfname = "RAD7JythonDS_CF"
```

- Define global variables to store the path of the EAR files and the enterprise application names:

```
webappEAR = "C:/7501code/jython/RAD7EJBWebEAR.ear"
ejbappEAR = "C:/7501code/jython/RAD7EJBEAR.ear"
webappEARName = "RAD7EJBWebEAR         # display name of the enterprise App
ejbappEARName = "RAD7EJBEAR            # display name of the enterprise App
```

- Create function calls for the six deployment steps:

```
createProvider()                     # Step 1 - Create the JDBC provider
createDS()                           # Step 2 - Create the data source
installApp(webappEAR, webappEARName) # Step 3 - Install ITSO Bank Web App
installApp(ejbappEAR,ejbappEARName)  # Step 4 - Install ITSO Bank EJB App
startApp(ejbappEARName)              # Step 5 - Start ITSO Bank EJB
startApp(webappEARName)              # Step 6 - Start ITSO Bank Web App
```

Next we define the six functions. Note that the function code goes before the main section.

Creating a JDBC provider

We create a function named **createProvider** for this purpose:

- This statement defines the function `createProvider`, which will encapsulate the code for creating a new JDBC provider:

```
def createProvider():
```

- This code snippet checks if the JDBC provider we want to create already exists, and if it does, then we simply return. Make sure that rest of the code for this method is indented:

```
prov = AdminControl.completeObjectName("name=" + jdbcProv +
          ",type=JDBCProvider,Server="+srvr + ",node="+nodeName + ",*")
if len(prov) > 0:
      return
#endif
```

- Define local variables for attributes required to create a new JDBC provider:

```
provName=['name',jdbcProv]
impClass=['implementationClassName',
                'org.apache.derby.jdbc.EmbeddedXADataSource']
jdbcAttrs=[]
jdbcAttrs.append(provName)
jdbcAttrs.append(impClass)
```

- We are using a template to create the JDBC provider:

```
tmplName = 'Derby JDBC Provider (XA)'
templates =
   AdminConfig.listTemplates("JDBCProvider",tmplName).split(lineSeparator)
tmpl = templates[0]
serverId = AdminConfig.getid("/Node:" + nodeName + "/Server:" + srvr + "/")
```

- Create the JDBC provider using the template, save the configuration, and end:

```
AdminConfig.createUsingTemplate("JDBCProvider",serverId,jdbcAttrs,tmpl)
AdminConfig.save()
#enddef
```

The complete code for this function is shown in Example 24-2.

Example 24-2 Create a JDBC provider using the template for Derby JDBC Provider (XA)

```
def createProvider():
   prov = AdminControl.completeObjectName("name="+jdbcProv + ",
          type=JDBCProvider,Server="+srvr + ",node="+nodeName + ",*")
   if len(prov) > 0:
      return
   #endif

   provName=['name',jdbcProv]
   impClass=['implementationClassName',
                   'org.apache.derby.jdbc.EmbeddedXADataSource']
   jdbcAttrs=[]
   jdbcAttrs.append(provName)
   jdbcAttrs.append(impClass)
   tmplName = 'Derby JDBC Provider (XA)'
   templates = AdminConfig.listTemplates("JDBCProvider", tmplName)
                                        .split(lineSeparator)
   tmpl = templates[0]
   serverId = AdminConfig.getid("/Node:" + nodeName + "/Server:" + srvr + "/")
   AdminConfig.createUsingTemplate("JDBCProvider",serverId,jdbcAttrs,tmpl)
   AdminConfig.save()
#enddef
```

Creating a data source

We create a function named **createDS** for this purpose:

```
def createDS():
```

- The following code snippet checks if the data source we want to create already exists, and if it does, then we simply return. Once again, make sure that rest of the code for this method is indented:

```
dsId = AdminConfig.getid("/JDBCProvider:"+jdbcProv + "/DataSource:"+
            dataSourceName + "/")
if len(dsId) > 0:
   return
#endif
```

- Define local variables for attributes required to create a data source:

```
dsname=['name',dataSourceName]
jndiName=['jndiName',jndiDS]
description=['description','ITSOBank Data Source']
dsHelperClassname=['datasourceHelperClassname',
                   'com.ibm.websphere.rsadapter.DerbyDataStoreHelper']
dsAttrs=[]
dsAttrs.append(dsname)
dsAttrs.append(jndiName)
dsAttrs.append(description)
dsAttrs.append(dsHelperClassname)
rovId = AdminConfig.getid("/Node:"+nodeName + "/Server:"+srvr +
                   "/JDBCProvider:"+jdbcProv + "/")
```

- Create the data source and save the configuration:

```
AdminConfig.create('DataSource',provId,dsAttrs)
AdminConfig.save()
```

- We have to configure the data source and add resource property set attributes, such as database name, password, description, and login timeout.

Because this piece of code is more than a few lines, we put the code into a separate function. Let us define a function call:

```
modifyDS()
```

- The last step is to enable the use of this data source in container-managed persistence. We use a separate method as well:

```
useDSinCMP()
```

- The code for this function is shown in Example 24-3.

Example 24-3 Create a data source

```
def createDS():
   dsId = AdminConfig.getid("/JDBCProvider:"+jdbcProv + "/DataSource:"+
                dataSourceName + "/")
   if len(dsId) > 0:
      return
   #endif
   dsname=['name',dataSourceName]
   jndiName=['jndiName',jndiDS]
   description=['description','ITSOBank Data Source']
   dsHelperClassname=['datasourceHelperClassname',
                   'com.ibm.websphere.rsadapter.DerbyDataStoreHelper']
   dsAttrs=[]
   dsAttrs.append(dsname)
   dsAttrs.append(jndiName)
   dsAttrs.append(description)
   dsAttrs.append(dsHelperClassname)
   provId = AdminConfig.getid("/Node:"+nodeName + "/Server:"+srvr +
                   "/JDBCProvider:"+jdbcProv + "/")
      AdminConfig.create('DataSource',provId,dsAttrs)
   AdminConfig.save()
   modifyDS()   # modify DS to add the poperties (databaseName)
   useDSinCMP() # enable this DS in Container Managed Persistence (CMP)
#enddef
```

Modify the data source with properties

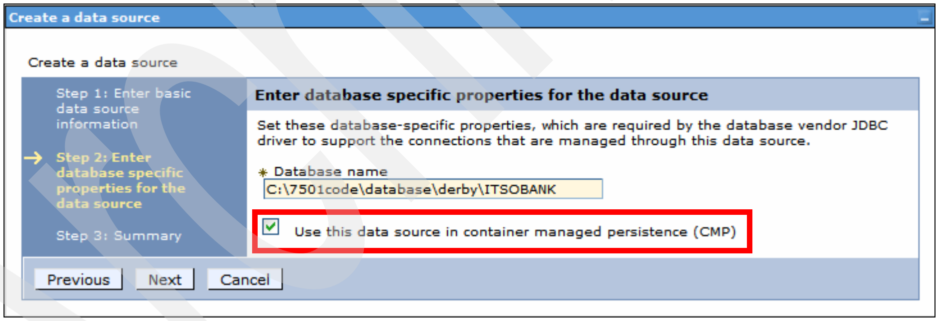
Even though we have already written the code to create the data source, we still have to add a resource property set with attributes such as the database name, password, description, and login time out. The code in Example 24-4 is the complete code for the **modifyDS** function.

Example 24-4 Add resource property set to the data source

```
def modifyDS():
   dsId = AdminConfig.getid("/JDBCProvider:"+jdbcProv + "/DataSource:"+
          dataSourceName + "/")
   dbnameAttrs = [["name", "databaseName"], ["value", dbasename], ["type",
         "java.lang.String"], ["description", "This is a required property"]]
   descrAttrs = [["name", "description"], ["value", ""], ["type",
                                                  "java.lang.String"]]
   passwordAttrs = [["name", "password"], ["value", ""], ["type",
                                                  "java.lang.String"]]
   loginTimeOutAttrs = [["name", "loginTimeout"], ["value", 0], ["type",
                                                  "java.lang.Integer"]]
   propset = []
   propset.append(dbnameAttrs)
   propset.append(descrAttrs)
   propset.append(passwordAttrs)
   propset.append(loginTimeOutAttrs)
   pSet = ["propertySet", [["resourceProperties", propset]]]
   attrs = [pSet]
   AdminConfig.modify(dsId, attrs)
   AdminConfig.save()
#enddef
```

Use the data source in container-managed persistence (CMP)

We have to configure this data source for use in container-managed persistence of entity EJBs. This is quite a simple and easy thing to do when using the administration console (Figure 24-27), but when using a script, this is a multiple step process.


Create a data source
Create a data source
Step 1: Enter basic data source information
Step 2: Enter database specific properties for the data source
Step 3: Summary
Previous
Next
Cancel

Enter database specific properties for the data source

Set these database-specific properties, which are required by the database vendor JDBC driver to support the connections that are managed through this data source.

* Database name
C:\7501code\database\derby\ITSOBANK

☑ Use this data source in container managed persistence (CMP)

Figure 24-27 Enabling a data source to be used in CMP

The code in Example 24-5 is the complete code for the **useDSinCMP** function.

Example 24-5 Enable the data source for container-managed persistence

```
def useDSinCMP():
   dsId = AdminConfig.getid("/JDBCProvider:"+jdbcProv +
         "/DataSource:"+dataSourceName + "/")
   rra = AdminConfig.getid("/Node:"+nodeName+"/Server:"+srvr
            +"/J2CResourceAdapter:WebSphere Relational Resource Adapter/")
   nameAttr = ["name", cfname]
   authmechAttr = ["authMechanismPreference", "BASIC_PASSWORD"]
   cmpdsAttr = ["cmpDatasource", dsId]
   attrs = []
   attrs.append(nameAttr)
   attrs.append(authmechAttr)
   attrs.append(cmpdsAttr)
   newcf = AdminConfig.create("CMPConnectorFactory", rra, attrs)
   AdminConfig.save()
   # Modify the CMPConnectionFactory to add the Mapping attributes
   mapAuthAttr = ["authDataAlias", ""]
   mapConfigaliasAttr = ["mappingConfigAlias", ""]
   mapAttrs = []
   mapAttrs.append(mapAuthAttr)
   mapAttrs.append(mapConfigaliasAttr)
   mappingAttr = ["mapping", mapAttrs]
   attrs2 = []
   attrs2.append(mappingAttr)
   cfId = AdminConfig.getid("/CMPConnectorFactory:"+cfname+"/")
   AdminConfig.modify(cfId, attrs2)
   AdminConfig.save()
   AdminConfig.modify(dsId, attrs2)
   AdminConfig.save()
#enddef
```

Install the enterprise applications

We are at the point where we have written the code required to create a JDBC provider and a data source for the ITSO Bank applications. Therefore, we are ready to start installing the EAR files. The best practice is to create a generic method that encapsulates the code to install any EAR file when provided with the location of the EAR file and the display name of the application.

The code in Example 24-6 is the complete code for the **installApp** function.

Example 24-6 Install or update an enterprise application

```
def installApp(appEAR, appName):
   try:
      app = AdminApp.view(appName)
   except:
      AdminApp.install(appEAR, options)
      AdminConfig.save()
   else:
      if app > 1:
         options = "-operation update -contents " + appEAR
         contentType = 'app'
         AdminApp.update(appName, contentType, options)
         AdminConfig.save()
```

```
#enddef
```

In the main section we had already defined variables that contain the complete path of the EAR files and also their display names. Therefore, we can use these variables as parameters and use the `installApp` function to install the `RAD7EJBEAR` and `RAD7EJBWebEAR` applications:

```
   installApp(webappEAR, webappEARName)
   installApp(ejbappEAR, ejbappEARName)
```

Start the enterprise applications

After installing the applications, we have to start both enterprise applications. Again, it makes sense to create a generic function that can start any enterprise application when provided with the display name of that application.

The code in Example 24-7 is the complete code for the **startApp** function.

Example 24-7 Start an enterprise application

```
def startApp(appName):
   app = AdminControl.completeObjectName
                                        ("type=Application,name="+appName+",*")
   if len(app) > 1:
      return

   appMgr = AdminControl.queryNames
             ("node="+nodeName+",type=ApplicationManager,process="+srvr+",*")

   AdminControl.invoke(appMgr,'startApplication',appName)
#enddef
```

Once again, we can use the global variables that contain the display name of both applications and pass them as parameters to the `startApp` function:

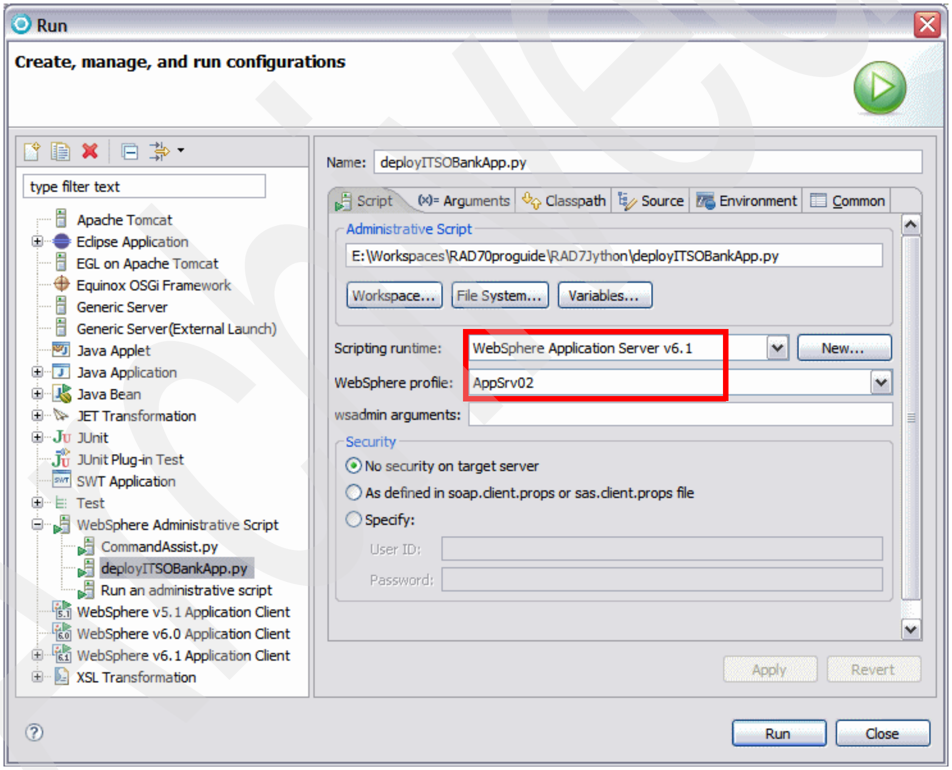
```
   startApp(ejbappEARName)
   startApp(webappEARName)
```

> **Tip:** The complete code of the **deployITSOBankApp.py** Jython script is available in the sample code:
>
> ```
> C:\7501code\jython\deployITSOBankApp.py
> ```
>
> Import this file into the `RAD7Jython` project (or copy/paste from Windows Explorer into Application Developer. In addition to the logic discussed above, the script has many comments and produces test output to the console (print statements) so that the execution can be followed.

Executing the Jython script

We execute the Jython script against the application server that is already configured in Application Developer. Do these steps:

- Make sure that the target server is started.
- Right-click **deployITSOBankApp.py** and select **Run As** → **Administrative Script**.
- In the Modify attributes and launch dialog:
  - Select **WebSphere Application Server V6.1 for** the Scripting runtime.
  - Select the WebSphere profile (**AppSrv01** or **AppSrv02**).
  - Click **Run** to execute the Jython script (Figure 24-28).



Name: deployITSOBankApp.py

Administrative Script: E:\Workspaces\RAD70proguide\RAD7Jython\deployITSOBankApp.py

Scripting runtime: WebSphere Application Server v6.1

WebSphere profile: AppSrv02

wsadmin arguments:

Security

● No security on target server

○ As defined in soap.client.props or sas.client.props file

○ Specify:

User ID:

Password:

Figure 24-28 Select the runtime and the WebSphere profile

The console view shows the result of the execution (Example 24-8).

Example 24-8 Output of the Jython script

```
WASX7209I: Connected to process "server1" on node UELIT60Node02 using SOAP
connector;  The type of process is: UnManagedProcess

***** STEP 1 completed - The ITSO Derby JDBC Provider (XA) has been created
using the template. Config saved ..
Done creating the DS. Config saved ..
Done modifying the DS. Config Saved ..
The DS has been modified to use it in CMP. Config saved ..

***** STEP 2 completed - Done creating the DS. Config saved ..
***** App not found: RAD7EJBWebEAR. Installing it now

ADMA5016I: Installation of RAD7EJBWebEAR started.
ADMA5058I: Application and module versions are validated with versions of
             deployment targets.
ADMA5005I: The application RAD7EJBWebEAR is configured in the WebSphere
             Application Server repository.
ADMA5053I: The library references for the installed optional package are
             created.
ADMA5005I: The application RAD7EJBWebEAR is configured in the WebSphere....
ADMA5001I: The application binaries are saved in C:\<WAS_HOME>\profiles
             \AppSrv02\wstemp\Script1144d386b1d\workspace\cells\<cell>
             \applications\RAD7EJBWebEAR.ear\RAD7EJBWebEAR.ear
ADMA5005I: The application RAD7EJBWebEAR is configured in the WebSphere ...
SECJ0400I: Successfuly updated the application RAD7EJBWebEAR with the
             appContextIDForSecurity information.
ADMA5011I: The cleanup of the temp directory for application RAD7EJBWebEAR is
complete.
ADMA5013I: Application RAD7EJBWebEAR installed successfully.

***** Done installing App: RAD7EJBWebEAR. Config saved ..
***** App not found: RAD7EJBEAR. Installing it now

ADMA5016I: Installation of RAD7EJBEAR started.
......
ADMA5013I: Application RAD7EJBEAR installed successfully.

***** Done installing App: RAD7EJBEAR. Config saved ..

***** The startApplication operation for RAD7EJBEAR completed.

***** The startApplication operation for RAD7EJBWebEAR completed.
```

Verifying the application after automatic installation

To verify that the ITSO Bank sample is deployed and working properly, follow the instructions in “Verifying the application after manual installation” on page 1139.

The verification concludes this chapter, as we have successfully deployed our enterprise applications, both manually by using the WebSphere Administrative Console, and also by using the Jython scripting with the Jython tooling provided by Application Developer V7.

Generation Jython source code for wsadmin commands

WebSphere Application Server provides a WebSphere Administration Command assist tool that can be used to generate the Jython source code for administrative activities. For further information and an example of using this tool, refer to “Generating WebSphere admin commands for Jython scripts” on page 996.

More information

For more information on application deployment, refer to these resources:

- WebSphere Application Server V6.1 Information Center:

  `http://publib.boulder.ibm.com/infocenter/wasinfo/v6r1/index.jsp?topic=/com.ibm.websphere.base.doc/info/welcome_base.html`

- Understand WebSphere Extended Deployment:

  `http://www.ibm.com/developerworks/autonomic/library/ac-webxd/`

- Learn how to publish an enterprise application with WebSphere Application Server and Application Server Toolkit, V6.1:

  `http://www.ibm.com/developerworks/edu/wes-dw-wes-hellowas.html`

25

Profile applications

Profiling is a technique used by developers to collect runtime data and detect application problems such as memory leaks, performance bottlenecks, excessive object creation, and exceeding system resource limits during the development phase.

This chapter introduces the features, architecture, and process for profiling applications using the profiling features of IBM Rational Application Developer V7.0. We have included an example for basic memory analysis, execution time analysis, and method code coverage analysis.

The chapter is organized into the following sections:

- Introduction to profiling
- Preparing for the profiling sample
- Profiling the sample application

Introduction to profiling

Traditionally, performance analysis is performed once an application is getting close to deployment or after it has already been deployed. The profiling tools included with Application Developer allow the developer to move the performance analysis to a much earlier phase in the development cycle, thus providing more time for changes to the application that might effect the architecture of the application before they become critical production environment issues.

The types of problems that Application Developer profiling tooling can assist in detecting include:

- Memory usage problems
- Performance bottlenecks
- Excessive object creation
- System resource limits

The profiling tools can be used to gather data on applications that are running:

- Inside an application server, such as WebSphere Application Server
- As a standalone Java application
- On the same system as Application Developer
- On a remote WebSphere Application Server with the IBM Rational Agent Controller installed
- In multiple JVMs

Profiling features

Within Application Developer, there are several analysis types. Each analysis type includes associated views that provides the user with the ability to concentrate on particular types of analysis such as memory leaks, performance bottlenecks, and excessive object creation while profiling an application.

The following analysis types and their associated views are described in this section:

- Basic memory analysis
- Execution time analysis
- Method code coverage analysis
- Probekit analysis

Basic memory analysis

Basic memory analysis displays statistics about the application heap. It is used to detect memory management problems. Memory analysis can help developers identify memory leaks as well as excessive object allocation that might cause performance problems. Basic memory analysis has been enhanced with the views described in Table 25-1.

Table 25-1 Basic memory analysis views

View name	Description
Memory statistics	Displays statistics about the application heap. It provides detailed information such as the number of classes loaded, the number of instances that are alive, and the memory size allocated by every class.
Object references	Displays references by a set of objects. This is useful to study data structures, to find memory leaks, and to find unexpected references.

Execution time analysis

Execution time analysis is used to detect performance problems by highlighting the most time intensive areas in the code. This type of analysis helps developers identify and remove unused or inefficient coding algorithms. Execution time analysis has been enhanced with the views described in Table 25-2.

Table 25-2 Execution time analysis views

View name	Description
Execution statistics	Displays statistics about the application execution time.
Method invocation	Displays a graphical representation of the entire course of a program's execution and also provides the ability to navigate through the methods that invoked the selected method.
Method invocation details	Displays statistical data on a selected method.
Object references	Displays references by a set of objects. This is useful to study data structures, to find memory leaks, and to find unexpected references.
UML2 trace interactions	Displays execution flow of an application according to the notation defined by UML.
Execution flow	Displays a representation of the entire program execution.

Method code coverage analysis

Method code coverage analysis is used to detect areas of code that have not been executed in a particular scenario that is tested. This capability is a useful analysis tool to integrate with component test scenarios and can be used to assist in identifying test cases that might be missing from a particular test suite or code that is redundant. Method code coverage analysis has been enhanced with the view described in Table 25-3.

Table 25-3 Method code coverage view

View name	Description
Coverage statistics	Displays usage statistics for a selected type of object.

Probekit analysis

Probes are reusable Java code fragments that you write to collect detailed runtime data about a program's objects, instance variables, arguments, and exceptions. *Probekit* provides a framework on the Eclipse platform to help you create and use probes. One common use of Probekit is to create lightweight profilers that collect only the data developers are interested in.

A probekit file can contain one or more probes, with each containing one or more probe fragments. These probes can be specified when to be executed or on which program they will be used. The probe fragments are a set of Java methods that are merged with standard boilerplate code with a new Java class generated and compiled. The functions generated from the probe fragments appear as static methods of the generated probe class.

The probekit engine—also called the byte-code instrumentation (BCI) engine—is used to apply probe fragments by inserting the calls into the target programs. The insertion process of the call statements into the target methods is referred to as *instrumentation*. The data items requested by a probe fragment are passed as arguments (for example, method name and arguments). The benefit of this approach is that the probe can be inserted into a large number of methods with small overhead.

Probe fragments can be executed at the following points:

- At method entry or exit time
- At exception handler time
- Before the original code in the class static initializer
- Before every executable code when source code is available
- When specific methods are called, not inside the called method

Each of the probe fragments can access the following data:

- Package, class, and method name
- Method signature
- this object
- Arguments
- Return value

There are two major types of probes available to the user (Table 25-4).

Table 25-4 Types of probes available with Probekit

Type of probe	Description
Method probe	Probe can be inserted anywhere within the body of a method with the class or jar files containing the target methods instrumented by the BCI engine.
Callsite probe	Probe is inserted into the body of the method that calls the target method. The class or jar files that call the target instrumented by the BCI engine.

Profiling architecture

The profiling architecture that exists in Application Developer is based on the Eclipse Test & Performance Tools Platform (TPTP) project. More detailed information on the Eclipse TPTP project can be found at:

```
http://www.eclipse.org/tptp/
```

In the past TPTP workbench users required the services of the standalone Agent Controller before they could use the function in the Profiling and Logging perspective and in the Test perspective. Even when the user tried to profile a Java application locally or to run TPTP tests locally, the Agent Controller would have to be installed on the local machine.

The *Integrated Agent Controller* is a new feature in the TPTP workbench, which allows users to profile a Java application locally and to run a TPTP test locally without requiring the standalone Agent Controller on the local machine. Profiling on a remote machine or running a TPTP test on a remote machine still requires the Agent Controller on that remote machine.

This feature is packaged in the TPTP runtime install image and therefore no separate install step is required. The Integrated Agent Controller does not require any configuration at all. Unlike the Agent Controller, which requires the user to enter information, such as the path for the Java executable, the Integrated Agent Controller determines the required information from the Eclipse workbench during startup.

TPTP provides the Agent Controller daemon with a process for enabling client applications to launch host processes and interact with agents that exist within host processes. Figure 25-1 shows the profiling architecture.

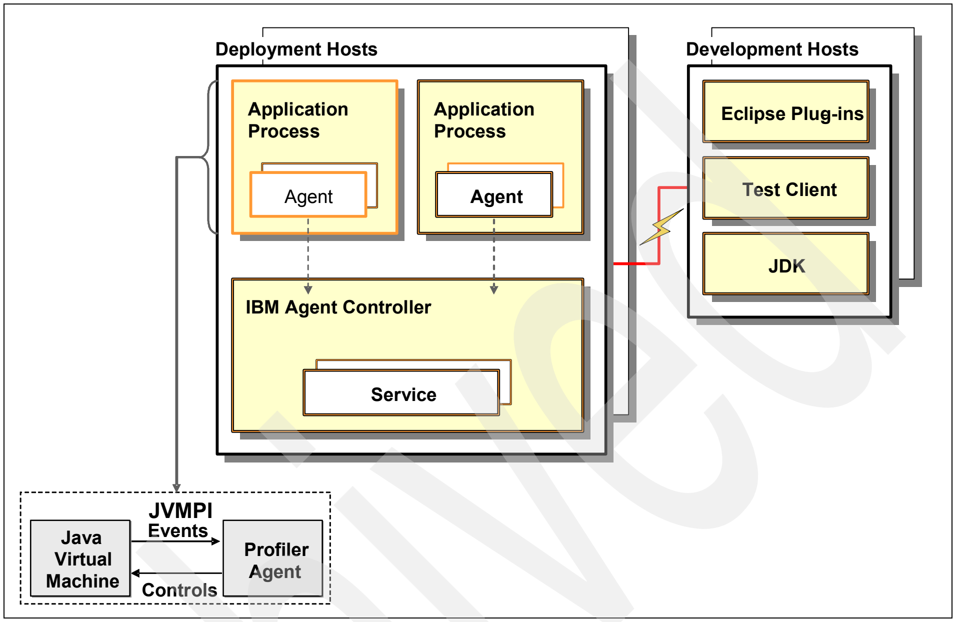


Figure 25-1 Profiling architecture of IBM Rational Application Developer V7.0

The definitions for the profiling architecture are as follows:

- **Application process**: The process that is executing the application consisting of the Java Virtual Machine (JVM) and the profiling agent.
- **Agent**: The profiling component installed with the application that provides services to the host process, and more importantly, provides a portal by which application data can be forwarded to attached clients.
- **Test Client**: A local or remote application that is the destination of host process data that is externalized by an agent. A single client can be attached to many agents at once, but does not always have to be attached to an agent.
- **Agent Controller**: A daemon process that resides on each deployment host providing the mechanism by which client applications can either launch new host processes, or attach to agents coexisting within existing host processes. The Agent Controller can only interact with host processes on the same node.

  Application Developer comes with an *integrated agent controller*, therefore, a separate install of the agent controller is not required. See "If the server fails to start" on page 1165 for a consequence of using the integrated agent controller.

- **Deployment hosts**: The host that an application has been deployed to and is being monitored for the capture of profiling agent.
- **Development hosts**: The host that runs an Eclipse-compatible architecture such as Application Developer to receive profiling information and data for analysis.

Each application process shown in Figure 25-1 represents a JVM that is executing a Java application that is being profiled. A profile agent will be attached to each application to collect the appropriate runtime data for a particular type of profiling analysis. This profiling agent is based on the Java Virtual Machine Profiler Interface (JVMPI) architecture. More details on the JVMPI specification can be found at:

```
http://java.sun.com/j2se/1.4.2/docs/guide/jvmpi
```

The data collected by the agent is then sent to the Agent Controller, which then forwards this information to Application Developer for analysis and visualization.

There are two types of profiling agents available in Application Developer:

- **Java Profiling Agent**: This agent is based on the JVMPI architecture and is shown in Figure 25-1. This agent is used for the collection of both standalone Java applications as well as applications running on an application server.
- **J2EE Request Profiling Agent**: This agent resides in an application server process and collects runtime data for J2EE applications by intercepting requests to the EJB or Web containers.

> **Note:** There is only one instance of the J2EE Request Profiling agent that is active in a process that hosts WebSphere Application Server.

Profiling and Logging perspective

After installing Application Developer, you have to enable the Profiling and Logging capability (see “Enabling the Profiling and Logging capability” on page 1163).

The Profiling and Logging perspective can be accessed by selecting **Window** → **Open Perspective** → **Other** → **Profiling and Logging** and then clicking **OK**. If it is not listed, click **Show all**.

There are many supporting views for the Profiling and Logging perspective. To see the supporting views select **Window** → **Show View** → **Other** under Profiling and Logging (Figure 25-2):

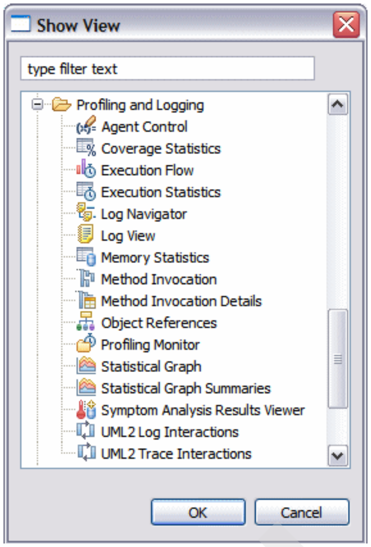


Figure 25-2 Profiling and Logging views

Preparing for the profiling sample

This section describes the tasks that have to be completed prior to profiling the sample Web application. We use the ITSO RedBank sample Web application developed in Chapter 12, “Develop Web applications using JSPs and servlets” on page 465, as our sample application for profiling.

Complete the following tasks in preparation for the profiling sample:

- Prerequisite software installation
- Enabling the Profiling and Logging capability
- Importing the sample project interchange file
- Publishing and running sample application

Prerequisite software installation

The working example requires the following software be installed:

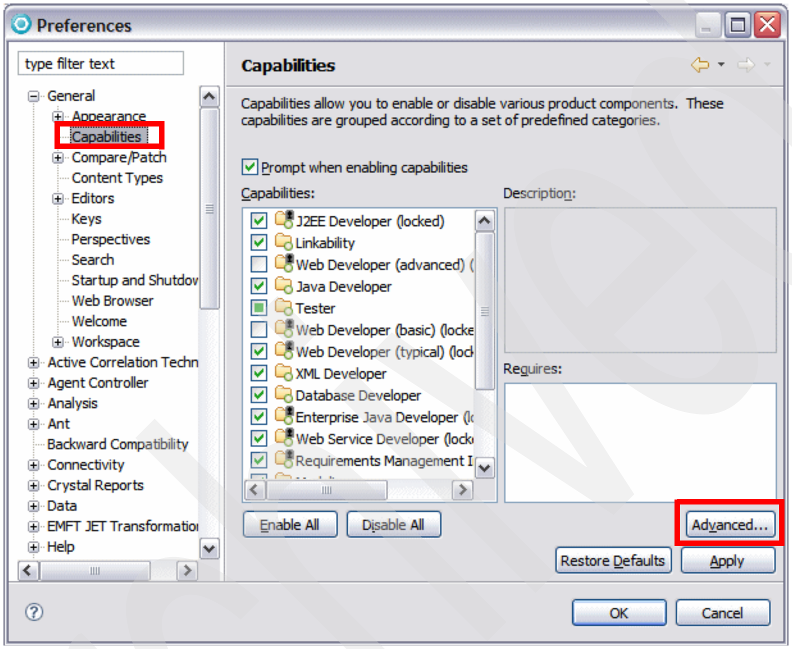
- IBM Rational Application Developer V7.0
- Integrated Agent Controller

  This feature is packaged in the Application Developer install image and therefore no separate install step is required.

Enabling the Profiling and Logging capability

To enable the Profiling and Logging capability in the preferences, do these steps:

- Select **Window** → **Preferences**.
- In the Preferences dialog expand **General** → **Capabilities** and click **Advanced** (Figure 25-3).



Capabilities allow you to enable or disable various product components. These capabilities are grouped according to a set of predefined categories.

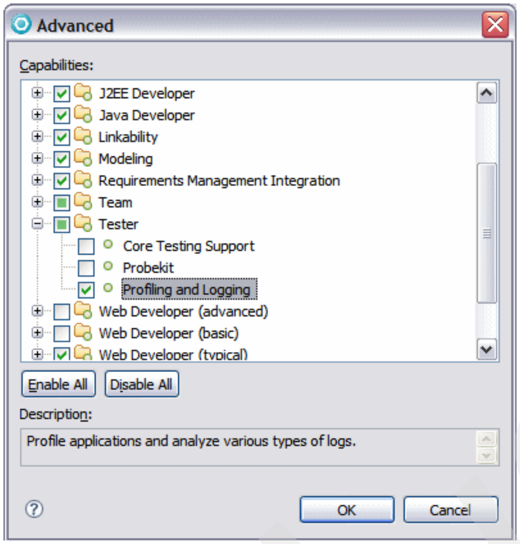
☑ Prompt when enabling capabilities

Capabilities:

☑ J2EE Developer (locked)
☑ Linkability
□ Web Developer (advanced) (
☑ Java Developer
▣ Tester
□ Web Developer (basic) (locke
☑ Web Developer (typical) (lock
☑ XML Developer
☑ Database Developer
☑ Enterprise Java Developer (l
☑ Web Service Developer (lock
☑ Requirements Management I

Figure 25-3 Enable Profiling and Logging capability (1)

- In the Advanced dialog expand **Tester**, and select **Profiling and Logging** (Figure 25-4) and click **OK**.



- ☑ J2EE Developer
- ☑ Java Developer
- ☑ Linkability
- ☑ Modeling
- ☑ Requirements Management Integration
- ■ Team
- ■ Tester
  - □ Core Testing Support
  - □ Probekit
  - ☑ Profiling and Logging
- □ Web Developer (advanced)
- □ Web Developer (basic)
- ☑ Web Developer (typical)

Profile applications and analyze various types of logs.

Figure 25-4 Enable Profiling and Logging capability (2)

> **Note:** If you want to use the `Probekit`, you have to enable this capability by selecting **Probekit** (Figure 25-4).

- In the Preferences dialog, click **OK**.

Importing the sample project interchange file

> **Tip:** If you have the RedBank Web application (`RAD7BankBasicWeb`) already in the workspace, skip this step.

To import the ITSO RedBank JSP and servlet Web application project interchange file, do these steps:

- Open the Web perspective Project Explorer view.
- Select **File** → **Import**.
- In the Import dialog, select **Project Interchange** and click **Next**.
- In the Import Projects dialog, click **Browse** and locate the file:

  ```
  c:\7501code\zInterchangeFiles\webapps\RAD7BankBasic.zip
  ```

- Select the **RAD7BankBasicEAR**, **RAD7BankBasicWeb**, and **RAD7Java** projects, and click **Finish**.

Publishing and running sample application

The sample application has to be published to the WebSphere Application Server, prior to running the application server in profile mode.

To publish and run the sample application on the WebSphere Application Server V6.1 test server, do these steps:

- Open the Web perspective Project Explorer view.
- Expand **RAD7BankBasicWeb** → **WebContent**.
- Right-click **index.html** and select **Run As** → **Run on Server**.
- In the Run on Server dialog, select **Choose an existing server**, select **WebSphere Application Server v6.1**, and click **Finish**.

This operation will start the server and publish the application to the server. The `index.html` page is displayed in a Web browser.

Profiling the sample application

In this section we demonstrate most of the profiling features and statistics for the ITSO RedBank Web application.

Starting server in profile mode

To start the WebSphere Application Server V6.1 in profile mode, do these steps:

- In the Servers view do these steps:
  - If the WebSphere Application Server V6.1 is started, right-click and select **Restart** → **Profile**.
  - If the WebSphere Application Server V6.1 is not started, right-click the server and select **Profile**.

It takes a while to start the server in profiling mode. In the Servers view you notice that the status of the server is changed to **Profiling** . Profiling .

If the server fails to start

On systems with Application Developer and the integrated agent controller (no separate install of the agent controller) we noticed that the server does not start, and an error is reported in the file:

```
<RAD_HOME>\runtimes\base_v61\profiles\AppSrv01\logs\server1\native_stderr.log
JVMJ9VM011W Unable to load piAgent: The specified module could not be
found. Could not create the Java virtual machine.
```

If you get this error, the hyades engine cannot be found. To fix this problem:

- Open **My Computer** → **Properties**. On the Advanced page, click **Environment Variables**. Find the Path variable and click **Edit**. Add this folder to the path:

  ```
  C:\<SDP_HOME>\SDP70Shared\plugins\org.eclipse.hyades.execution.win32.x86
  _4.2.3.v200701141614
  ```

  Click **OK** several times to close the dialogs.
- Stop the server in Application Developer, then close Application Developer.
- Start Application Developer, then start the server in **Profiling** mode.

Profile on server dialog

After the server has started, the Profile on server dialog is displayed (Figure 25-5).

> **Important:** If the Profile on server dialog fails to open, we can configure profiling manually. See “Configure profiling manually” on page 1169.

- Select **[PID: xxxx]** and click > to move the PID to the Selected agents pane.

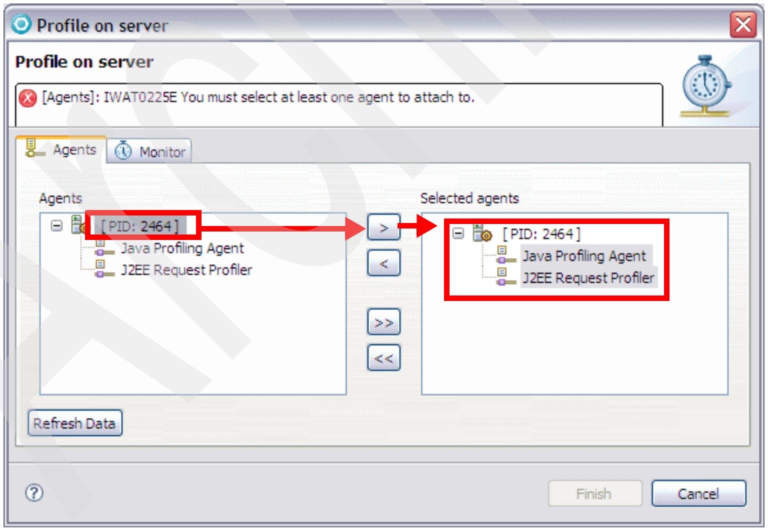
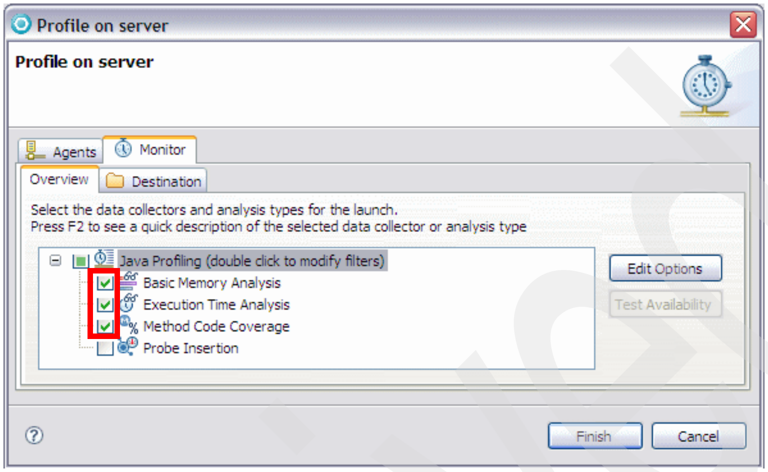

[Agents]: IWAT0225E You must select at least one agent to attach to.

Figure 25-5 Profile on server configuration (1)

- Select the **Monitor** tab and expand **Java Profiling**.
- Select **Basic Memory Analysis**, **Execution Time Analysis**, and **Method Code Coverage** (Figure 25-6).



Profile on server

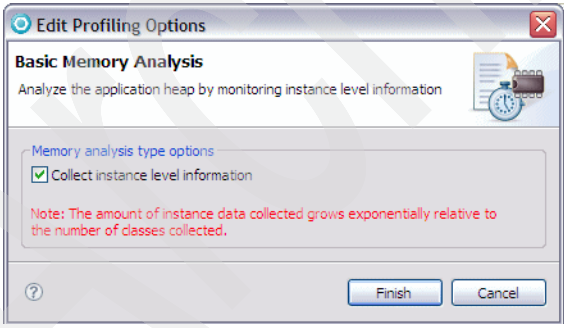
Profile on server

Select the data collectors and analysis types for the launch.
Press F2 to see a quick description of the selected data collector or analysis type

- ⊟ ■ Java Profiling (double click to modify filters)
  - ☑ Basic Memory Analysis
  - ☑ Execution Time Analysis
  - ☑ Method Code Coverage
  - □ Probe Insertion

Figure 25-6 Profile on server configuration (2)

- Select **Basic Memory Analysis** and click **Edit Options**.
- In the Edit Profiling Options dialog, select **Collect instance level information** and click **Finish** (Figure 25-7).



Edit Profiling Options

Basic Memory Analysis

Analyze the application heap by monitoring instance level information

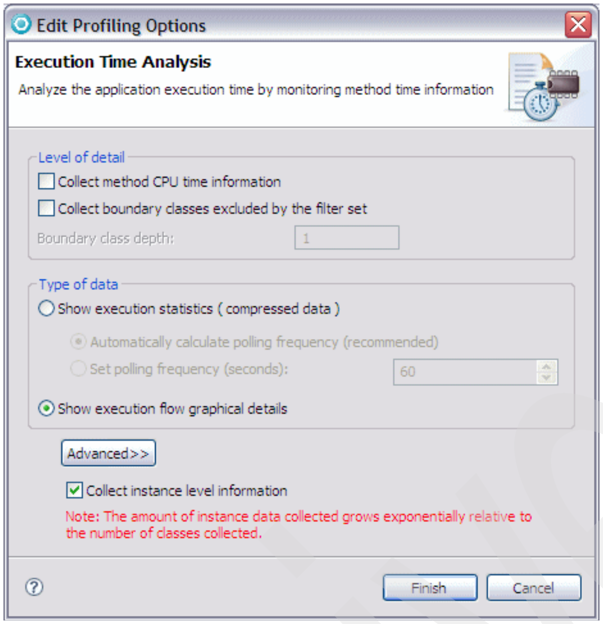
Memory analysis type options

☑ Collect instance level information

Note: The amount of instance data collected grows exponentially relative to the number of classes collected.

Figure 25-7 Edit Profiling Options: Basic Memory Analysis

- Select **Execution Time Analysis** and click **Edit Options**.
- In the Edit Profiling Options dialog, select **Show execution flow graphical details**. Click **Advanced** and select **Collect instance level information**. Click **Finish** (Figure 25-8).



Edit Profiling Options

Execution Time Analysis

Analyze the application execution time by monitoring method time information

Level of detail

□ Collect method CPU time information

□ Collect boundary classes excluded by the filter set

Boundary class depth: 1

Type of data

○ Show execution statistics (compressed data)

● Automatically calculate polling frequency (recommended)

○ Set polling frequency (seconds): 60

● Show execution flow graphical details

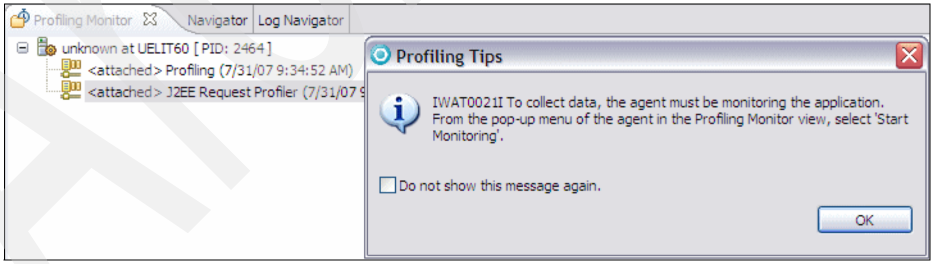
Advanced>>

☑ Collect instance level information

Note: The amount of instance data collected grows exponentially relative to the number of classes collected.

Figure 25-8 Edit Profiling Options - Execution Time Analysis

- In the Profile on server dialog, click **Finish**.
- Click **Yes** in the Confirm Perspective switch dialog.
- The Profiling and Logging perspective opens and the agent is displayed in the Profiling Monitor view.
- In the Profiling tips dialog, click **OK** (Figure 25-9).



unknown at UELIT60 [PID: 2464]

<attached> Profiling (7/31/07 9:34:52 AM)

<attached> J2EE Request Profiler (7/31/07 9

Profiling Tips

IWAT0021I To collect data, the agent must be monitoring the application. From the pop-up menu of the agent in the Profiling Monitor view, select 'Start Monitoring'.

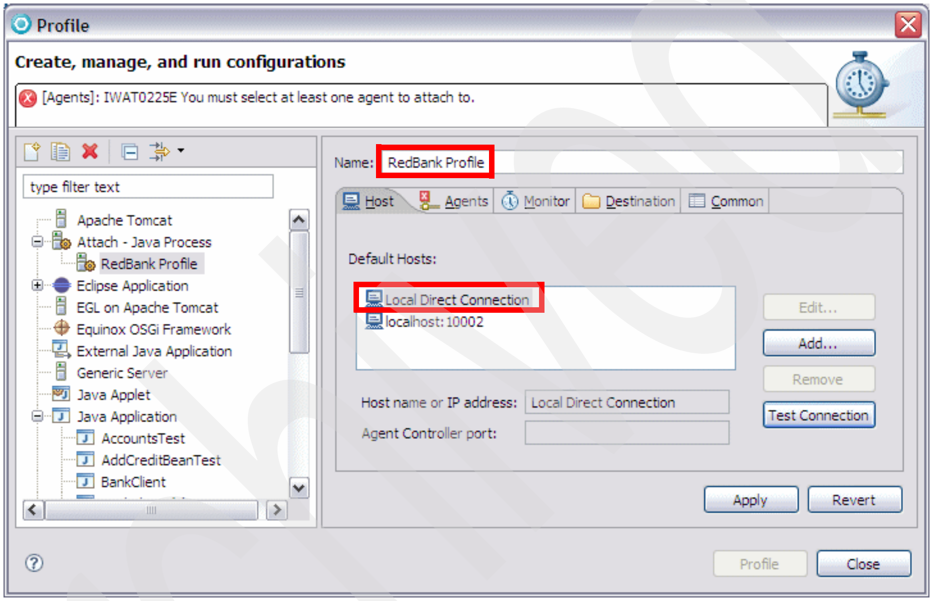
□ Do not show this message again.

Figure 25-9 Profiling Monitor with Profiling Tips dialog

Configure profiling manually

If the profile on server dialog does not appear, or if you want to repeat profiling, do these steps:

- Right-click the **RAD7BankBasicWeb** project and select **Profile As** → **Profile**. The Profile dialog opens.
- Right-click **Attach - Java Process** and select **New**.
- Change the name to **RedBank Profile** (Figure 25-10).



Profile

Create, manage, and run configurations

[Agents]: IWAT0225E You must select at least one agent to attach to.

Name: RedBank Profile

Default Hosts:

Host name or IP address: Local Direct Connection

Agent Controller port:

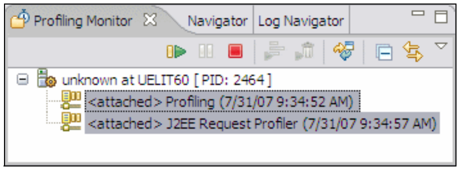
Figure 25-10 Profile configuration (1)

- Select **Local Direct Connection** and click **Test Connection**. You receive a successful message.
- Select the **Agents** tab. Select **[PID: xxxx]** and click > to move the PID to the Selected agents pane (this is the same as Figure 25-5 on page 1166).
- Select the **Monitor** tab and expand **Java Profiling**. Select **Basic Memory Analysis**, **Execution Time Analysis, and Method Code Coverage** (Figure 25-6 on page 1167). Configure Basic Memory Analysis and Execution Time Analysis (Figure 25-7 on page 1167 and Figure 25-8 on page 1168).
- In the Destination tab, you can see that a **ProfileProject** is created.
- Click **Apply**, then click **Profile**.
- Switch perspective when prompted. The agent appears in the Profiling Monitor view (Figure 25-9 on page 1168).

Collecting profiling data

To collect the profiling data, do these steps:

- In the Profiling Monitor view of the Profiling and Logging perspective, select the two attached profiler processes (Figure 25-11).
- Right-click and select **Start Monitoring** (or click the icon in the toolbar).



unknown at UELIT60 [PID: 2464]
<attached> Profiling (7/31/07 9:34:52 AM)
<attached> J2EE Request Profiler (7/31/07 9:34:57 AM)

Figure 25-11 Selecting active process in the Profiling Monitor view

Run part of the sample application to collect profiling data

> **Note:** This step requires that you have published the project to the server as described in “Publishing and running sample application” on page 1165.

To collect data, we will display the accounts of one customer, run a debit and a credit transaction, list the transactions, and update the customer name:

- Enter the following URL to launch the application in a Web browser:

```
http://localhost:9080/RAD7BankBasicWeb/index.html
```

- In the RedBank home page, click **redbank**.
- In the Login page, enter **`222-22-2222`** in the customer `SSN` field and click **Submit**.
- Select the second account.
- In the Account Details page, select **Withdraw** and withdraw an amount of $25.
- In the Account Details page, select **Deposit** and deposit $33.
- In the Account Details page, select **List Transactions**.
- In the List Transactions page, click **Account Details**.
- In the Account Details page, click **Customer Details**.
- In the Customer page, change the last name, and click **Update**.
- In the Customer page, click **Logout**.

Stop monitoring

In the profiling Monitor view, pause monitoring by selecting both processes and clicking the Pause Monitoring icon ⏸ (or right-click and select **Pause Monitoring**.

You can also keep the profiling running while you analyze the accumulated data.

Analyzing profiling data

We have now run the part of the sample application that we want to collect data for. In this section, we analyze the collected data for coverage statistics, memory statistics, and execution statistics.

To display the collected data, we use the toolbar icons (Figure 25-12). Alternatively, you can right-click the process and select **Open With** → statistic:

- Open Execution Flow
- Open Memory Statistics
- Open Execution Statistics
- Open Coverage Statistics
- Open Object References

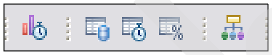
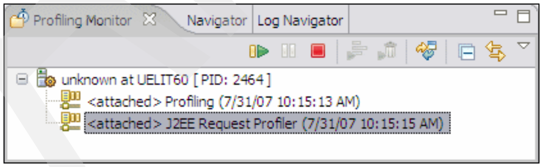


Figure 25-12 Profiling statistics icons

Coverage statistics

The Coverage Statistics view displays usage statistics for a selected type of object. Coverage statistics are available at the package, class, method, and instance level.

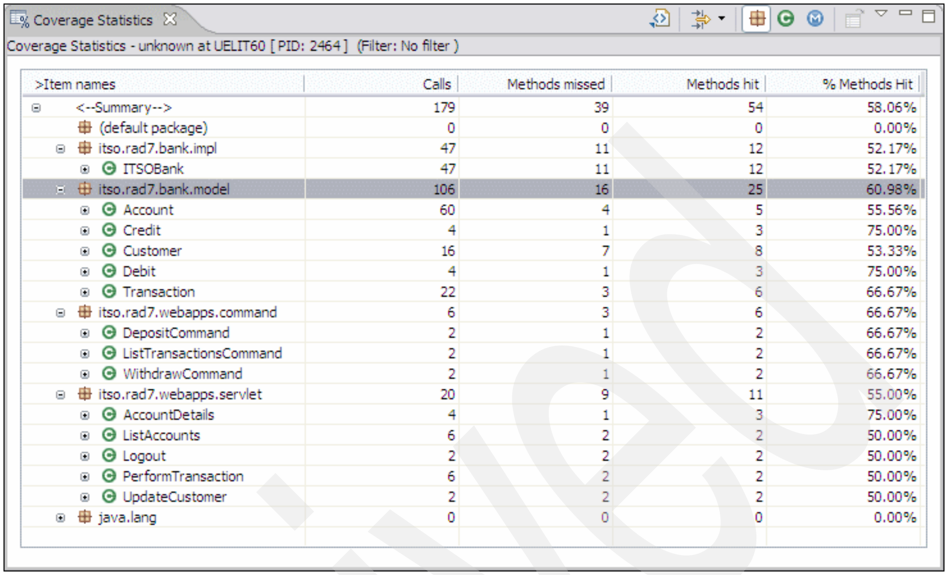
To analyze the coverage statistics, in the Profiling Monitor view, select the second attached profiler process and click the icon, or select **Open With** → **Coverage Statistics** (Figure 25-13).



unknown at UELIT60 [PID: 2464]
<attached> Profiling (7/31/07 10:15:13 AM)
<attached> J2EE Request Profiler (7/31/07 10:15:15 AM)

Figure 25-13 Selecting active process in the Profile Monitor view

The Coverage Statistics view opens (Figure 25-14).



Coverage Statistics - unknown at UELIT60 [PID: 2464] (Filter: No filter)

>Item names	Calls	Methods missed	Methods hit	% Methods Hit
<--Summary-->	179	39	54	58.06%
(default package)	0	0	0	0.00%
itso.rad7.bank.impl	47	11	12	52.17%
ITSOBank	47	11	12	52.17%
itso.rad7.bank.model	106	16	25	60.98%
Account	60	4	5	55.56%
Credit	4	1	3	75.00%
Customer	16	7	8	53.33%
Debit	4	1	3	75.00%
Transaction	22	3	6	66.67%
itso.rad7.webapps.command	6	3	6	66.67%
DepositCommand	2	1	2	66.67%
ListTransactionsCommand	2	1	2	66.67%
WithdrawCommand	2	1	2	66.67%
itso.rad7.webapps.servlet	20	9	11	55.00%
AccountDetails	4	1	3	75.00%
ListAccounts	6	2	2	50.00%
Logout	2	2	2	50.00%
PerformTransaction	6	2	2	50.00%
UpdateCustomer	2	2	2	50.00%
java.lang	0	0	0	0.00%

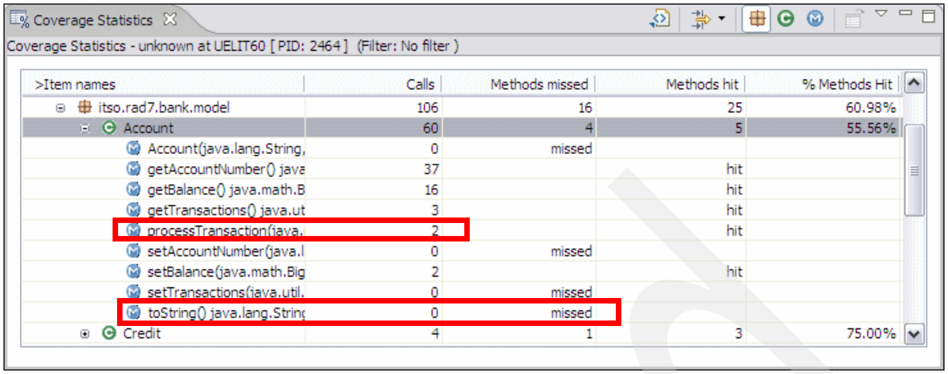
Figure 25-14 Coverage Statistics view

The Coverage Statistics view shows how many times a package or class has been called, and how many methods were hit and missed.

For each object type, the following statistics are displayed:

- **Calls**: The number of method calls.
- **Methods missed**: The number of methods that have not been called.
- **Methods hit**: The number of methods that have been called.
- **% Methods hit**: The percentage rate between methods hit and methods total (missed + hit).

You can drill down into the methods by expanding a class to see which methods were missed and how many times each method was called (Figure 25-15). This type of information is useful for function testing.



Coverage Statistics

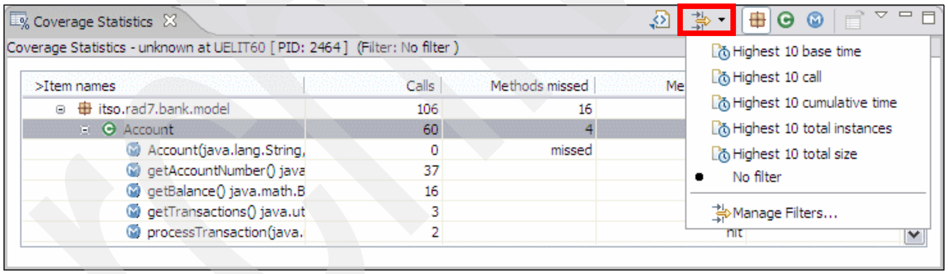
Coverage Statistics - unknown at UELIT60 [PID: 2464] (Filter: No filter)

>Item names	Calls	Methods missed	Methods hit	% Methods Hit
itso.rad7.bank.model	106	16	25	60.98%
Account	60	4	5	55.56%
Account(java.lang.String,	0	missed		
getAccountNumber() java	37		hit	
getBalance() java.math.B	16		hit	
getTransactions() java.ut	3		hit	
processTransaction(java.	2		hit	
setAccountNumber(java.l	0	missed		
setBalance(java.math.Big	2		hit	
setTransactions(java.util.	0	missed		
toString() java.lang.Strin(	0	missed		
Credit	4	1	3	75.00%

Figure 25-15 Coverage Statistics view

The display options of the Coverage Statistics view can be changed from the views' toolbar menu . You can select to view package, class, or method level information.

The view provides filtering based on the name of the package, class, or method (Figure 25-16). You can also filter on time or heap size and on particular attributes of the package, class, method, or object instance.



Coverage Statistics

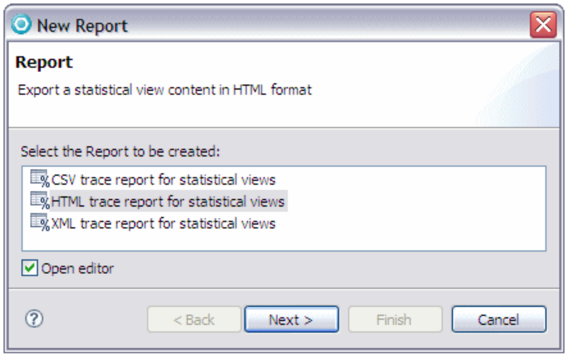
Coverage Statistics - unknown at UELIT60 [PID: 2464] (Filter: No filter)

>Item names	Calls	Methods missed	Me
itso.rad7.bank.model	106	16	
Account	60	4	
Account(java.lang.String,	0	missed	
getAccountNumber() java	37		
getBalance() java.math.B	16		
getTransactions() java.ut	3		
processTransaction(java.	2		

Figure 25-16 Coverage Statistics view filter option

You can examine the source code for a class or method by right-clicking it in the Coverage Statistics view and selecting **Open Source**.

The view also provides the reporting capability to export data in CSV, HTML, and XML format by clicking the icon. The New Report dialog allows you to select the report type and open the document (Figure 25-17).



Report

Export a statistical view content in HTML format

Select the Report to be created:

- CSV trace report for statistical views
- HTML trace report for statistical views
- XML trace report for statistical views

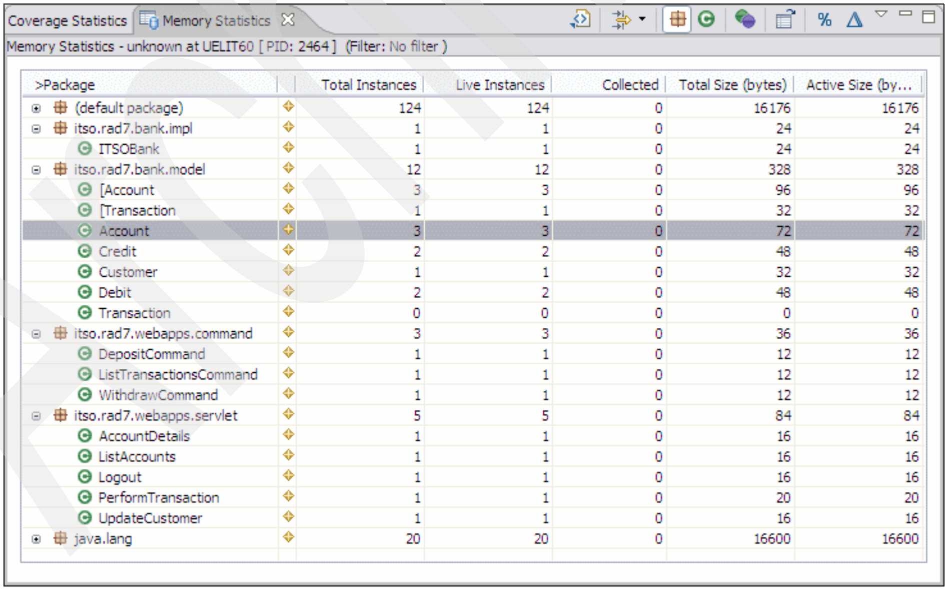
☑ Open editor

Figure 25-17 Coverage Statistics view report option

Memory statistics

The Memory Statistics view displays statistics about the application heap. It provides detailed information such as the number of classes loaded, the number of instances that are alive, and the memory size allocated by every class. Memory statistics are available at the package, class, and instance level.

In the Profiling Monitor view, select the second attached profiler process and click the icon, or select **Open With** → **Memory Statistics** (Figure 25-18).



Memory Statistics - unknown at UELIT60 [PID: 2464] (Filter: No filter)

>Package		Total Instances	Live Instances	Collected	Total Size (bytes)	Active Size (by...
(default package)	◆	124	124	0	16176	16176
itso.rad7.bank.impl	◆	1	1	0	24	24
ITSOBank	◆	1	1	0	24	24
itso.rad7.bank.model	◆	12	12	0	328	328
[Account	◆	3	3	0	96	96
[Transaction	◆	1	1	0	32	32
Account	◆	3	3	0	72	72
Credit	◆	2	2	0	48	48
Customer	◆	1	1	0	32	32
Debit	◆	2	2	0	48	48
Transaction	◆	0	0	0	0	0
itso.rad7.webapps.command	◆	3	3	0	36	36
DepositCommand	◆	1	1	0	12	12
ListTransactionsCommand	◆	1	1	0	12	12
WithdrawCommand	◆	1	1	0	12	12
itso.rad7.webapps.servlet	◆	5	5	0	84	84
AccountDetails	◆	1	1	0	16	16
ListAccounts	◆	1	1	0	16	16
Logout	◆	1	1	0	16	16
PerformTransaction	◆	1	1	0	20	20
UpdateCustomer	◆	1	1	0	16	16
java.lang	◆	20	20	0	16600	16600

Figure 25-18 Memory Statistics view

For each object type, the following statistics are displayed:

- **Total Instances**: Shows the total number of instances that had been created of the selected package, class, or method.
- **Live Instances**: Shows the number of instances of the selected package, class, or method, where no garbage collection has taken place.
- **Collected**: Shows the number of instances of the selected package, class, or method, that were removed during garbage collection.
- **Total Size**: Shows the total size in bytes of the selected package, class, or method, of all instances that were created for it.
- **Active Size**: Shows the total number of size of all live instances.

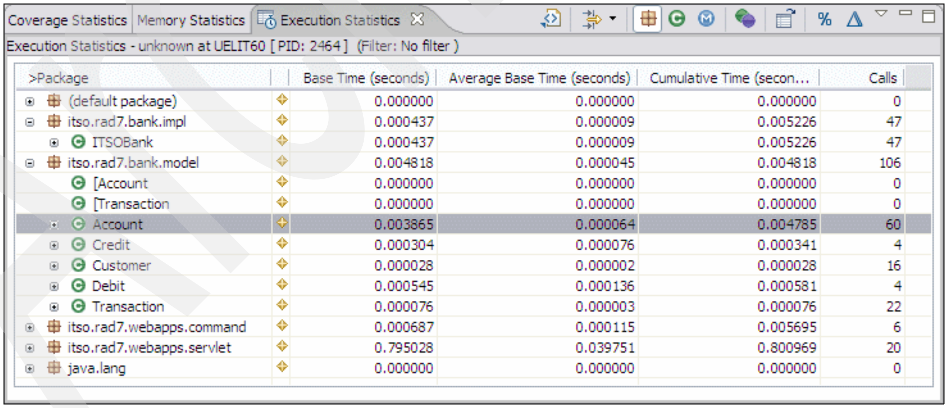
You can drill down to classes , or even instances of classes .

Filtering and reporting capabilities also exist in Memory Statistics view.

Execution statistics

The Execution Statistics view displays statistics about the application execution time. It provides data such as the number of methods called, and the amount of time taken to execute every method. Execution statistics are available at the package, class, method and instance level.

To analyze the execution statistics, in the Profiling Monitor view, select the second attached profiler process and click the icon, or select **Open With** → **Execution Statistics** (Figure 25-19).



Execution Statistics - unknown at UELIT60 [PID: 2464] (Filter: No filter)

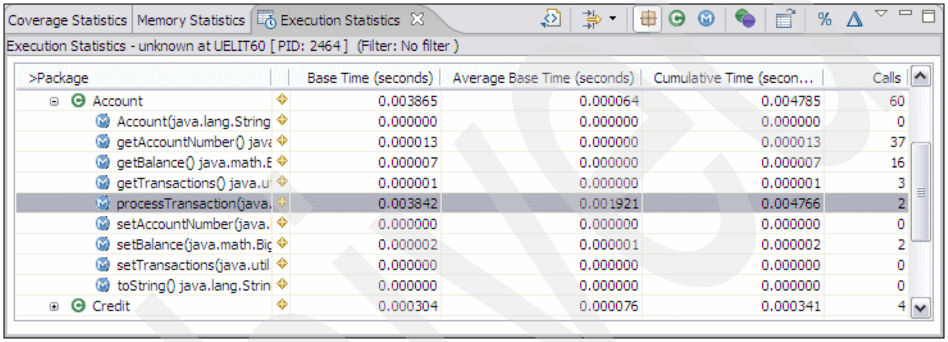
>Package		Base Time (seconds)	Average Base Time (seconds)	Cumulative Time (secon...	Calls
(default package)	◆	0.000000	0.000000	0.000000	0
itso.rad7.bank.impl	◆	0.000437	0.000009	0.005226	47
ITSOBank	◆	0.000437	0.000009	0.005226	47
itso.rad7.bank.model	◆	0.004818	0.000045	0.004818	106
[Account	◆	0.000000	0.000000	0.000000	0
[Transaction	◆	0.000000	0.000000	0.000000	0
Account	◆	0.003865	0.000064	0.004785	60
Credit	◆	0.000304	0.000076	0.000341	4
Customer	◆	0.000028	0.000002	0.000028	16
Debit	◆	0.000545	0.000136	0.000581	4
Transaction	◆	0.000076	0.000003	0.000076	22
itso.rad7.webapps.command	◆	0.000687	0.000115	0.005695	6
itso.rad7.webapps.servlet	◆	0.795028	0.039751	0.800969	20
java.lang	◆	0.000000	0.000000	0.000000	0

Figure 25-19 Execution Statistics view

For each object type, the following statistics are displayed:

- **Base Time**: The time taken to execute the invocation (excluding time spent in called methods).
- **Average Base Time**: The base time divided by the number of calls.
- **Cumulative Time**: The time taken to execute the invocation (including time spent in called method).
- **Calls**: The number of calls made to the package, class, or method.

You can expand a class to see the statistics for each method (Figure 25-20).



Execution Statistics - unknown at UELIT60 [PID: 2464] (Filter: No filter)

>Package		Base Time (seconds)	Average Base Time (seconds)	Cumulative Time (secon...	Calls
⊟ Account	◈	0.003865	0.000064	0.004785	60
Account(java.lang.String	◈	0.000000	0.000000	0.000000	0
getAccountNumber() java	◈	0.000013	0.000000	0.000013	37
getBalance() java.math.E	◈	0.000007	0.000000	0.000007	16
getTransactions() java.u	◈	0.000001	0.000000	0.000001	3
processTransaction(java.	◈	0.003842	0.001921	0.004766	2
setAccountNumber(java.	◈	0.000000	0.000000	0.000000	0
setBalance(java.math.Bic	◈	0.000002	0.000001	0.000002	2
setTransactions(java.util	◈	0.000000	0.000000	0.000000	0
toString() java.lang.Strin	◈	0.000000	0.000000	0.000000	0
⊞ Credit	◈	0.000304	0.000076	0.000341	4

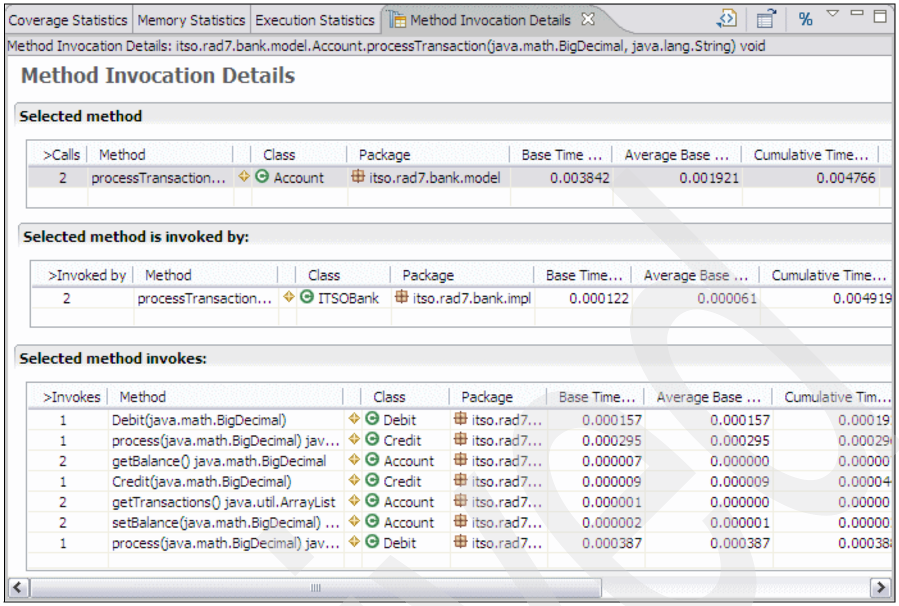
Figure 25-20 Execution statistics by methods of a class

Method invocation details

Right-click `processTransaction` and select **Show Method Invocation Details** (Figure 25-21).

The Method Invocation Details view provides statistical data on a selected method. The following data is displayed for the selected method:

- **Selected method** (`Account.processTransaction`): Shows details including the number of times the selected method is called, the class and package information, and the time taken by this method.
- **Selected method invoked by**: Shows details of each method that calls the selected method, including the number of calls to the selected method, and the number of times the selected method is invoked by the caller. In our case, the `processTransaction` method of the `ITSOBank` class invokes the selected method.
- **Selected method invokes**: Shows details of each method invoked by the selected method, for example, `Credit` constructor, `Account.getBalance`, and `Credit.process`.



Method Invocation Details: itso.rad7.bank.model.Account.processTransaction(java.math.BigDecimal, java.lang.String) void

Method Invocation Details

Selected method

>Calls	Method	Class	Package	Base Time ...	Average Base ...	Cumulative Time...
2	processTransaction...	Account	itso.rad7.bank.model	0.003842	0.001921	0.004766

Selected method is invoked by:

>Invoked by	Method	Class	Package	Base Time...	Average Base ...	Cumulative Time...
2	processTransaction...	ITSOBank	itso.rad7.bank.impl	0.000122	0.000061	0.004919

Selected method invokes:

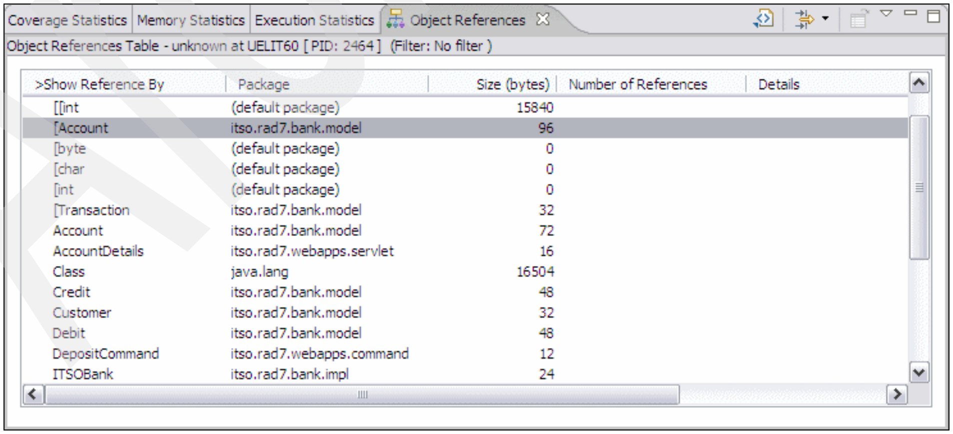
>Invokes	Method	Class	Package	Base Time...	Average Base ...	Cumulative Tim...
1	Debit(java.math.BigDecimal)	Debit	itso.rad7...	0.000157	0.000157	0.00019
1	process(java.math.BigDecimal) jav...	Credit	itso.rad7...	0.000295	0.000295	0.00029
2	getBalance() java.math.BigDecimal	Account	itso.rad7...	0.000007	0.000000	0.00000
1	Credit(java.math.BigDecimal)	Credit	itso.rad7...	0.000009	0.000009	0.00004
2	getTransactions() java.util.ArrayList	Account	itso.rad7...	0.000001	0.000000	0.00000
2	setBalance(java.math.BigDecimal) ...	Account	itso.rad7...	0.000002	0.000001	0.00000
1	process(java.math.BigDecimal) jav...	Debit	itso.rad7...	0.000387	0.000387	0.00038

Figure 25-21 Method Invocation Details view

Object references

The Object References view displays statistics about classes and objects, and how many references there are to these objects.

In the Profiling Monitor view, select the second attached profiler process and click the icon, or select **Open With** → **Object References** (Figure 25-22).



Object References Table - unknown at UELIT60 [PID: 2464] (Filter: No filter)

>Show Reference By	Package	Size (bytes)	Number of References	Details
[[int	(default package)	15840		
[Account	itso.rad7.bank.model	96		
[byte	(default package)	0		
[char	(default package)	0		
[int	(default package)	0		
[Transaction	itso.rad7.bank.model	32		
Account	itso.rad7.bank.model	72		
AccountDetails	itso.rad7.webapps.servlet	16		
Class	java.lang	16504		
Credit	itso.rad7.bank.model	48		
Customer	itso.rad7.bank.model	32		
Debit	itso.rad7.bank.model	48		
DepositCommand	itso.rad7.webapps.command	12		
ITSOBank	itso.rad7.bank.impl	24		

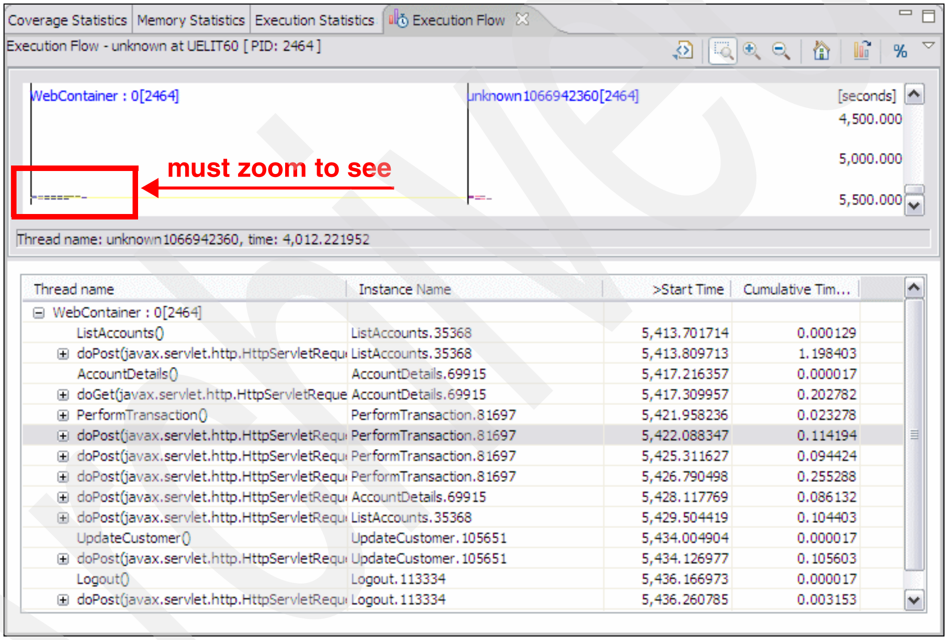
Figure 25-22 Object References view

The Object References view displays references by a set of objects. This is useful to study data structures, to find memory leaks, and to find unexpected references.For example, the total size of `Account` objects is 288 bytes.

Execution flow

The Execution Flow view and table both show a representation of the entire program execution. In this view, the threads of the program fit horizontally, and time is scaled so that the entire execution fits vertically. In the table, the threads are grouped in the first column and time is recorded in successive rows.

In the Profiling Monitor view, select the second attached profiler process and click the icon, or select **Open With** → **Execution Flow** (Figure 25-23).



Thread name: unknown1066942360, time: 4,012.221952

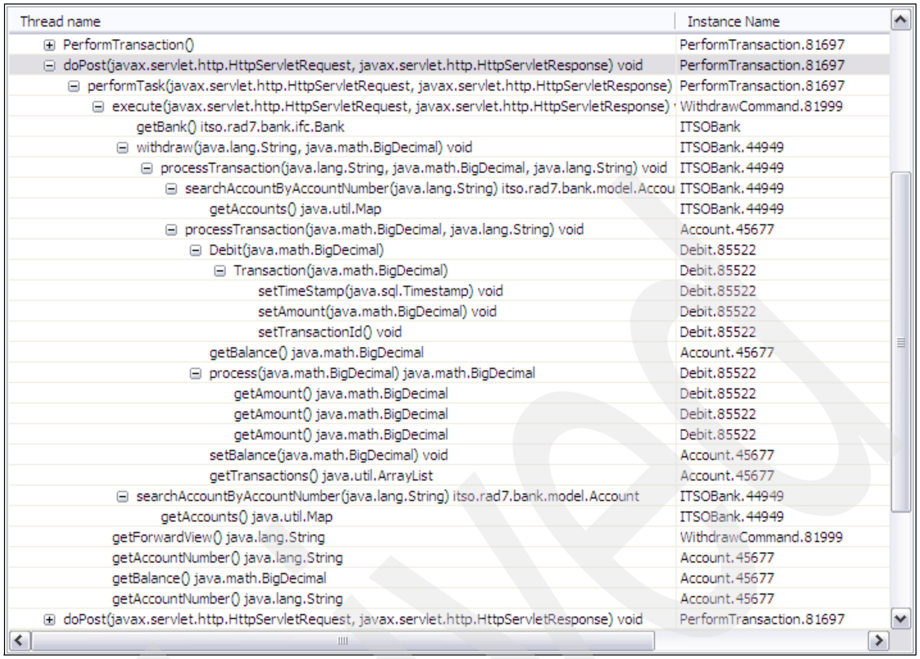
Thread name	Instance Name	>Start Time	Cumulative Tim...
⊟ WebContainer : 0[2464]			
ListAccounts()	ListAccounts.35368	5,413.701714	0.000129
⊞ doPost(javax.servlet.http.HttpServletRequ	ListAccounts.35368	5,413.809713	1.198403
AccountDetails()	AccountDetails.69915	5,417.216357	0.000017
⊞ doGet(javax.servlet.http.HttpServletReque	AccountDetails.69915	5,417.309957	0.202782
⊞ PerformTransaction()	PerformTransaction.81697	5,421.958236	0.023278
⊞ doPost(javax.servlet.http.HttpServletRequ	PerformTransaction.81697	5,422.088347	0.114194
⊞ doPost(javax.servlet.http.HttpServletRequ	PerformTransaction.81697	5,425.311627	0.094424
⊞ doPost(javax.servlet.http.HttpServletRequ	PerformTransaction.81697	5,426.790498	0.255288
⊞ doPost(javax.servlet.http.HttpServletRequ	AccountDetails.69915	5,428.117769	0.086132
⊞ doPost(javax.servlet.http.HttpServletRequ	ListAccounts.35368	5,429.504419	0.104403
UpdateCustomer()	UpdateCustomer.105651	5,434.004904	0.000017
⊞ doPost(javax.servlet.http.HttpServletRequ	UpdateCustomer.105651	5,434.126977	0.105603
Logout()	Logout.113334	5,436.166973	0.000017
⊞ doPost(javax.servlet.http.HttpServletRequ	Logout.113334	5,436.260785	0.003153

Figure 25-23 Execution Flow view

The top pane shows the execution stripes, but we have to zoom in to see any details. The time in seconds is displayed on the right side. All our actions occurred after 5 seconds of waiting to get the application started.

The bottom pane displays the action sequence: `ListAccounts`, `AccountDetails`, `PerformTransactions`, `ListAccounts`, `CustomerUpdate`, and `Logout`.

To see the details of an interaction, expand one of the `doPost` methods to see the complete execution flow (Figure 25-24).



```
Thread name                                                                                     Instance Name
  ⊞ PerformTransaction()                                                                        PerformTransaction.81697
  ⊟ doPost(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse) void  PerformTransaction.81697
    ⊟ performTask(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse) PerformTransaction.81697
      ⊟ execute(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse) '  WithdrawCommand.81999
          getBank() itso.rad7.bank.ifc.Bank                                                     ITSOBank
        ⊟ withdraw(java.lang.String, java.math.BigDecimal) void                                 ITSOBank.44949
          ⊟ processTransaction(java.lang.String, java.math.BigDecimal, java.lang.String) void   ITSOBank.44949
            ⊟ searchAccountByAccountNumber(java.lang.String) itso.rad7.bank.model.Accou         ITSOBank.44949
                getAccounts() java.util.Map                                                     ITSOBank.44949
            ⊟ processTransaction(java.math.BigDecimal, java.lang.String) void                   Account.45677
              ⊟ Debit(java.math.BigDecimal)                                                     Debit.85522
                ⊟ Transaction(java.math.BigDecimal)                                             Debit.85522
                    setTimeStamp(java.sql.Timestamp) void                                       Debit.85522
                    setAmount(java.math.BigDecimal) void                                        Debit.85522
                    setTransactionId() void                                                     Debit.85522
                getBalance() java.math.BigDecimal                                               Account.45677
              ⊟ process(java.math.BigDecimal) java.math.BigDecimal                              Debit.85522
                  getAmount() java.math.BigDecimal                                              Debit.85522
                  getAmount() java.math.BigDecimal                                              Debit.85522
                  getAmount() java.math.BigDecimal                                              Debit.85522
                setBalance(java.math.BigDecimal) void                                           Account.45677
                getTransactions() java.util.ArrayList                                           Account.45677
          ⊟ searchAccountByAccountNumber(java.lang.String) itso.rad7.bank.model.Account         ITSOBank.44949
              getAccounts() java.util.Map                                                       ITSOBank.44949
        getForwardView() java.lang.String                                                       WithdrawCommand.81999
        getAccountNumber() java.lang.String                                                     Account.45677
        getBalance() java.math.BigDecimal                                                       Account.45677
        getAccountNumber() java.lang.String                                                     Account.45677
  ⊞ doPost(javax.servlet.http.HttpServletRequest, javax.servlet.http.HttpServletResponse) void  PerformTransaction.81697
```

Figure 25-24 Execution Flow view expanded

Zoom into the top pane using the **Zoom In** icon 🔍 to see methods and their times (Figure 25-25). The time spent in methods in our application is too small to make the graph very readable.

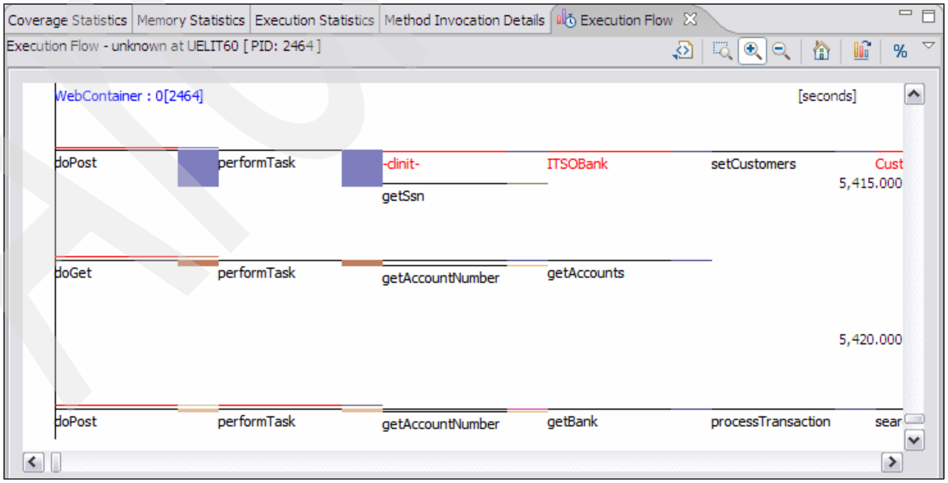


Figure 25-25 Execution Flow zoomed in

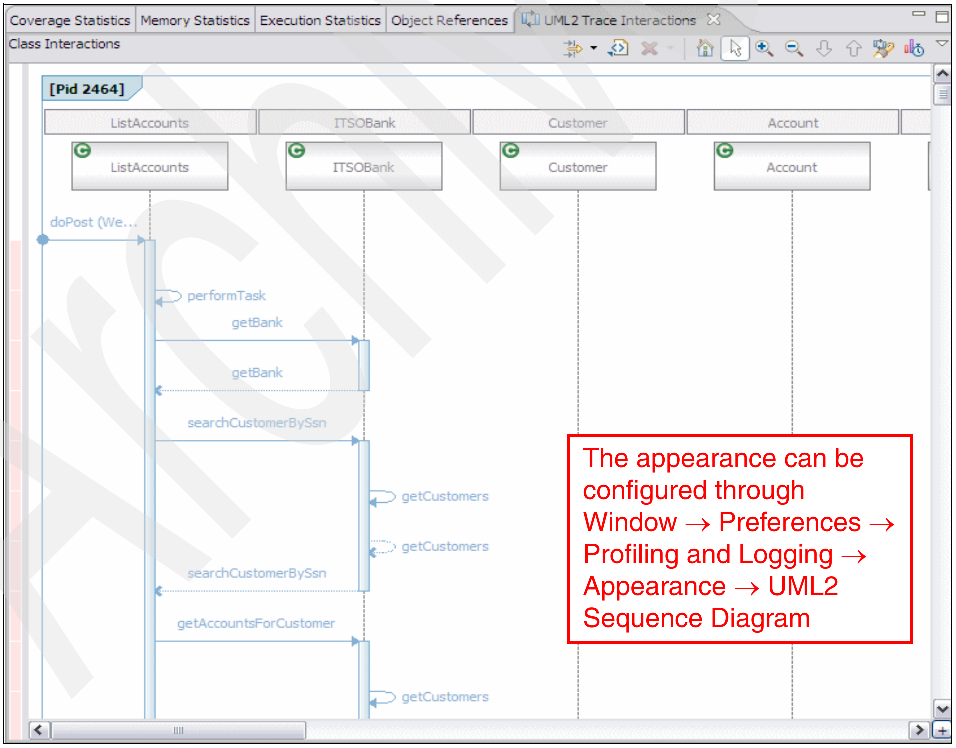
UML sequence diagrams

You can also analyze the graphical details of the execution flow using the data collected with Execution Time Analysis. These graphical details are displayed using the UML sequence diagram notation. The representation of time in these diagrams helps in determining bottlenecks in application performance as well as network communication. The following types of diagrams are available:

- **UML2 Class Interactions**: Shows interactions of classes that participate in the execution of an application.
- **UML2 Object Interactions**: Shows interactions of objects that participate in the execution of an application.
- **UML2 Thread Interactions**: Shows interactions of methods that execute in different threads, which participate in the execution of an application.

To display UML2 interaction diagrams, do these steps:

- In the Profiling Monitor view, right-click the second attached profiler process and select **Open With** → **UML2 Class interactions** (Figure 25-26).



The appearance can be configured through Window → Preferences → Profiling and Logging → Appearance → UML2 Sequence Diagram

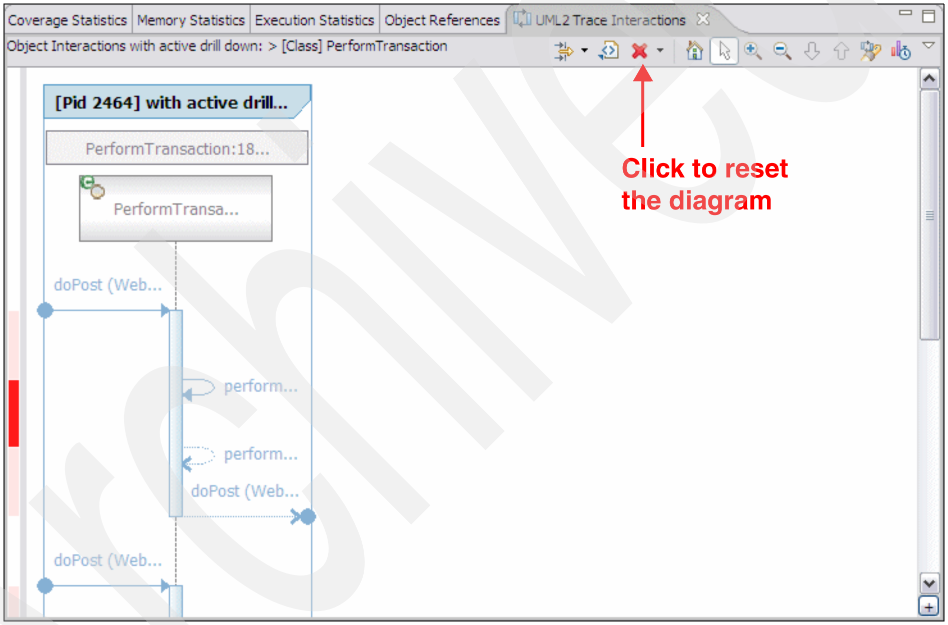
Figure 25-26 UML2 Class Interactions view

The other two UML diagrams are very similar: UML2 Object Interactions and UML2 Thread Interactions.

You can open the execution flow by clicking the **Open Execution Flow Table** icon (top-right in the toolbar).

You can drill down into a lifeline that allows you to view all the trace interactions within a particular lifeline. This feature helps to trace the root cause of a problem from a host, to a process, to a thread, and eventually to a class or an object.

For example, scroll right and locate the `PerformTransaction` class. Right-click the class and select **Drill down into selected lifeline**. Zoom in to enlarge the diagram (Figure 25-27).



Click to reset the diagram

Figure 25-27 UML2 CLass Interactions view: Drill down

You can highlight a call stack, which allows you to view all the methods invocations in a call stack. To highlight a call stack, right-click a method and select **Highlight call stack** (Figure 25-28). All method invocations in the call stack are highlighted.

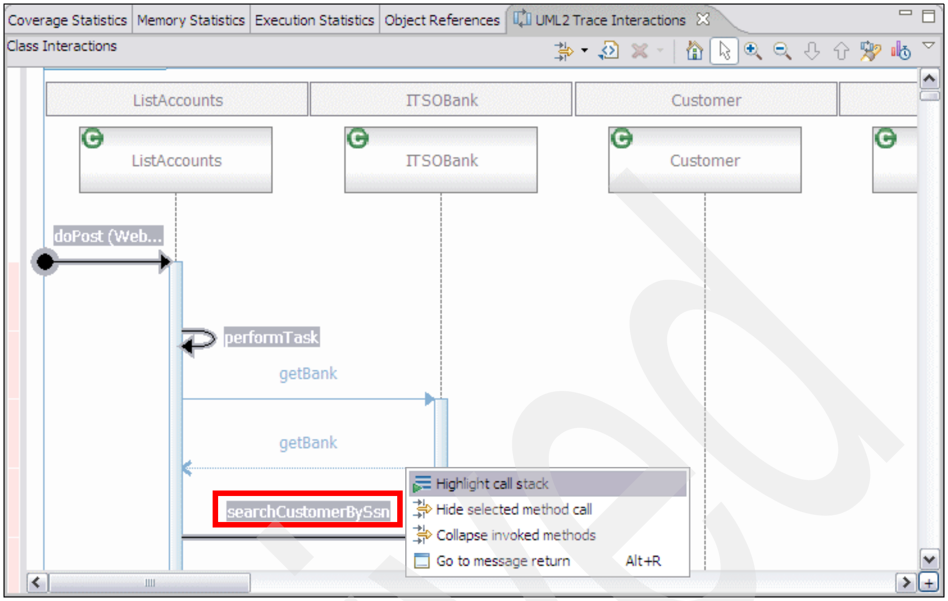


Figure 25-28 UML2 Class Interactions view: Highlight call stack

Refreshing the views and resetting data

You can keep the profiling running while you analyze the views. Now and then you might want to refresh the views with the latest data, or reset the data:

- Click the **Refresh Views** icon to refresh the views.
- Right-click the profiling process and select **Reset Data** to start a new collection of data and subsequent analysis.

Ending the profiling session

To end the profiling session, click the Terminate icon . To remove the profiling agent, right-click the agent and select **Delete**. You are prompted if you want to delete the data in the file system. The connection to the server is removed.

More information

In the Application Developer Online Help, select **Testing functionality and performance** → **Monitoring and profiling applications**.

Part 5

Team development

In this part of the book, we describe the tooling and technologies provided by Application Developer for developing applications in a team environment.

26

ClearCase integration

This chapter introduces the features and terminology of IBM Rational ClearCase with respect to Application Developer. In addition, we provide a basic scenario with two developers working in parallel on a common Web project using Application Developer and ClearCase. The focus of the example is to demonstrate the tooling of the ClearCase integration feature of the Application Developer.

> **Note:** Before starting this chapter, you should install ClearCase LT on your machine, as it is described in “Installing IBM Rational ClearCase LT” on page 1304.

The chapter is organized into the following sections:

- Introduction to IBM Rational ClearCase
- ClearCase integration feature of Application Developer
- Integration scenario overview
- Joining a UCM project
- ClearCase setup for a new project
- Development scenario

Introduction to IBM Rational ClearCase

This section provides an introduction to the ClearCase product as well as basic information on the ClearCase integration feature of Application Developer.

We have organized the section into the following topics:

- ClearCase overview
- ClearCase terminology
- ClearCase LT setup

ClearCase overview

ClearCase is a software configuration management (SCM) product that helps to automate the tasks required to write, release, and maintain software code.

ClearCase offers the essential functions of version control, workspace management, process configuration, and build management. By automating many of the necessary and error-prone tasks associated with software development, it helps teams of all sizes build high-quality software.

ClearCase incorporates Unified Change Management (UCM), Rational's best practices process for managing change at the activity level, and control for workflow.

UCM can be applied to projects out-of-the-box, enabling teams to get up and running quickly. However, it can be replaced with any other process that you already have in place at your site.

ClearCase provides support for parallel development. With automatic branching and merge support, it enables multiple developers to design, code, test, and enhance software from a common, integrated code base.

Snapshot™ views support a disconnected use model for working away from the office. All changes since the last snapshot are automatically updated once you are connected again.

IBM offers two versions of the ClearCase product:

- ClearCase
- ClearCase LT

ClearCase LT is a light version for support of small teams that do not need the full functionality of the complete ClearCase product (distributed servers, database replication, advanced build management, and transparent file access). A product license for *IBM Rational ClearCase LT, Version 7.0* is included with IBM Rational Application Developer, Version 7.0.

For the *full-sized* ClearCase version, the product also provides an add-on *MultiSite* feature.

More information on ClearCase products can be found at:

http://www.ibm.com/software/awdtools/clearcase/

ClearCase terminology

Here, we outline some of the key terminology used in ClearCase:

- **Activity**: A unit of work performed by an individual. In UCM, an activity tracks a change set, that is, a list of versions of files created to perform the work (for example, Developer 1 fixing problem report #123). When you work on an activity, all versions you create are associated with that activity.
- **Component**: A set of related directory and file elements. Typically, elements that make up a component are developed, integrated, and released together. In Application Developer, a component contains one or more projects.
- **Baseline**: A version of a project.
- **Development stream**: Each developer's own working area.
- **Integration stream**: A shared working area for the team, containing the versions of the components that are available to all developers.
- **Deliver stream**: The act of making a developer's development stream available to the integration stream, publishing a developer's work.
- **Rebase**: The act of retrieving a project to work on locally, or to synchronize your development stream with what is available in the integration stream.
- **Check in** and **check out**: A file that is to be edited must be checked out. This lets other developers know that the file is opened by another developer. Once a developer completes any edits on a file, it must be checked back in before making the files available to others.
- **Versioned object base (VOB)**: The permanent data repository where ClearCase stores files, directories, and meta data.
- **View**: A selection of resources in a VOB, a window to the VOB data.

ClearCase LT setup

How to install IBM Rational ClearCase LT is described in “Installing IBM Rational ClearCase LT” on page 1304.

Figure 26-1 shows a typical development environment when using ClearCase with two developers.

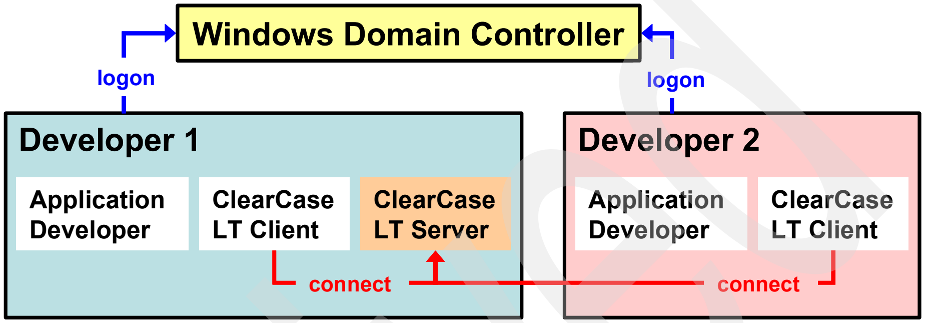


Figure 26-1 Sample ClearCase LT setup between two developers

In this chapter, we are simulating this setup by using two workspaces on the same machine.

ClearCase integration feature of Application Developer

Application Developer includes integration support for ClearCase and ClearCase LT, allowing easy access to ClearCase features. The ClearCase adapter is automatically activated when you start Application Developer the next time after ClearCase installation. In case you have customized your perspective, you might not see the ClearCase menu item and the ClearCase toolbar. Right-click in the toolbar and select **Customize Perspective**. In the Customize Perspective dialog, open the **Commands** tab, select **ClearCase**, and click **OK**.

ClearCase Help in Application Developer

Application Developer provides documentation for using ClearCase:

- To access the help documentation in Application Developer, select **Help** → **Help Contents** → **Working in a team environment**.
- To access the ClearCase Help system, select **ClearCase** → **ClearCase Help** or click ? in the toolbar.

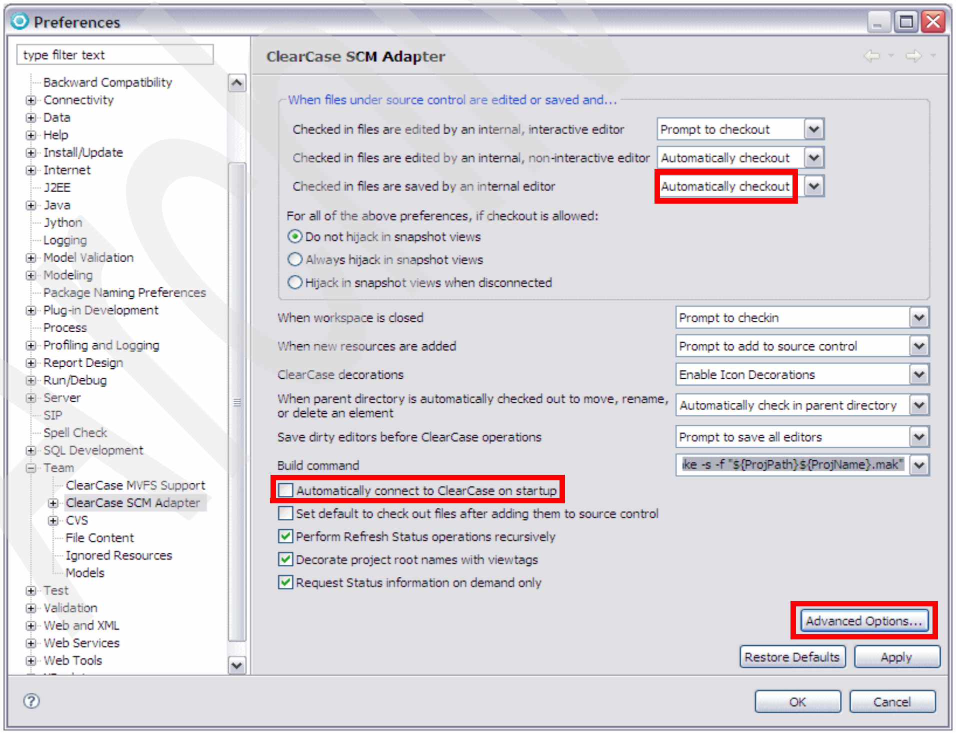
Tip: most of the ClearCase icons in the toolbar becomes active when you connect to ClearCase sever by clicking in the toolbar.

ClearCase preferences

Ensure that the ClearCase SCM Adapter capability is enabled. Refer to “Setting up the ClearCase server environment” on page 1192 for details.

There are a number of ClearCase preferences that you can modify by selecting **Window** → **Preferences** → **Team** → **ClearCase SCM Adapter** (Figure 26-2).

We recommend that you check out files from ClearCase before you edit them. However, if you edit a file that is under ClearCase control but is not checked out, Application Developer can automatically check it out for you, if you select **Automatically checkout** for Checked in files that are saved by an internal editor. You might want to specify to automatically connect to ClearCase when you start Application Developer by selecting **Automatically connect to ClearCase on startup**.



ClearCase SCM Adapter

When files under source control are edited or saved and...

Checked in files are edited by an internal, interactive editor: Prompt to checkout

Checked in files are edited by an internal, non-interactive editor: Automatically checkout

Checked in files are saved by an internal editor: Automatically checkout

For all of the above preferences, if checkout is allowed:

- ● Do not hijack in snapshot views
- ○ Always hijack in snapshot views
- ○ Hijack in snapshot views when disconnected

When workspace is closed: Prompt to checkin

When new resources are added: Prompt to add to source control

ClearCase decorations: Enable Icon Decorations

When parent directory is automatically checked out to move, rename, or delete an element: Automatically check in parent directory

Save dirty editors before ClearCase operations: Prompt to save all editors

Build command: ke -s -f "${ProjPath}${ProjName}.mak"

- □ Automatically connect to ClearCase on startup
- □ Set default to check out files after adding them to source control
- ☑ Perform Refresh Status operations recursively
- ☑ Decorate project root names with viewtags
- ☑ Request Status information on demand only

Figure 26-2 ClearCase SCM Adapter Preferences dialog

If you click **Advanced Options** and select the **Operations** tab, there is a preference for generating backup copies when you cancel a checkout. This is enabled by default, and specifies that ClearCase generates backup copies when you perform an undo checkout operation. The backup files will have a `.keep` extension.

> **Note:** You must be connected to ClearCase for the Advanced Options button to be active.

The ClearCase online help in Rational Developer contains a detailed description of each option of the preferences page.

Enable ClearCase capability

By default, the ClearCase SCM Adapter capability is disabled in Application Developer. To enable, do these steps:

- Select **Window** → **Preferences** → **General** → **Capabilities**.
- Click **Advanced** and expand **Team**.
- Select **ClearCase SCM Adapter** and **Core Team Support**.
- Click **OK**, then **Apply** and **OK**.

Integration scenario overview

This section describes a scenario with two developers, developer 1 and 2, working on a Web project called `RAD7ITSOExampleWeb`. Developer 1 is assigned the role of project integrator and is responsible for setting up the environment.

Table 26-1 Development activities

Developer 1 activities	Developer 2 activities
► Creates a new ClearCase project, ITSO_Project ► Joins the ClearCase project by creating views ► Creates a new Web project, `RAD7ITSOExampleWeb` ► Moves the project under ClearCase source control ► Adds a servlet, ServletA ► Delivers the work to the integration stream ► Makes a baseline	

Developer 1 activities	Developer 2 activities
	► Joins the ClearCase project by creating views ► Rebases his view to match the integration stream ► Imports the project into his workspace
► Checks out ServletA ► Makes changes ► Checks in ServletA	► Checks out ServletA ► Makes changes ► Checks in ServletA ► Delivers the work to the integration stream
► Delivers the work to the integration stream ► Resolves conflicts by using the merge tool	

The setup of this scenario and its flow are shown in Figure 26-3. ClearCase terminology is used for the tasks.

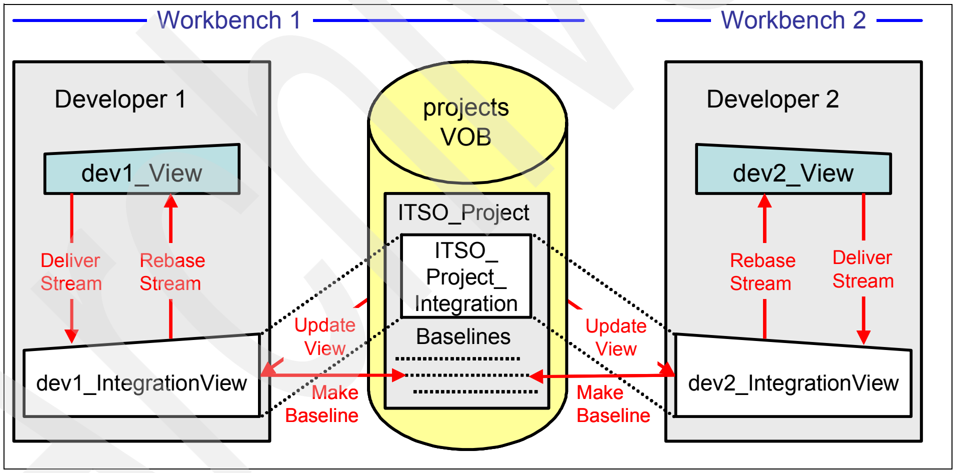


Figure 26-3 Scenario setup

Note that the integration view is like a window to the integration stream. The integration stream should be reserved only for code that has passed the developer's inspection and is sharable to the entire team.

When finished with the changes, the developer delivers his or her development stream back to the integration stream. A project integrator (or any of the developers) can then make a new baseline freezing the latest code changes.

ClearCase setup for a new project

In this example scenario, developer 1 and developer 2 work on the same machine by switching workspaces to simulate multiple users. Alternatively, you could have developer 1 working on a machine where a ClearCase LT server is installed, and other developers working on machines where only the ClearCase LT client is installed. The steps are basically the same in both cases.

Setting up the ClearCase server environment

Before we can use ClearCase, we have to set up the ClearCase server environment:

- Create a component for storing the project baseline
- Create a VOB and one component
- Create new UCM project

Create a component for storing the project baseline

Using a composite baseline to represent the project is easier than keeping track of a set of baselines, one for each component. Although you can store a composite baseline and elements in the same component, it is cleaner to dedicate one component for storing the project baseline. To ensure that nobody creates elements in this component, create the component without a VOB root directory. A component that has no VOB root directory cannot store its own elements.

> **Note:** If you are creating only one VOB with a single VOB-level component, and you do not have more than one component in the project, then it means that you do not have a composite baseline, and therefore, a root-less component is not required.

To create a component to storing the project baseline, do these steps:

- As developer 1, select **Start** → **Programs** → **IBM Rational** → **IBM Rational ClearCase** → **Project Explorer**. The path might be different on your system.
- Expand **process** → **Components**, right-click **Components** and select **New** → **Component without a VOB**.
- In the Create Component without a VOB dialog:
  - Enter `itso_example_baseline_component` in the Name field.
  - Enter `Component for storing the project baseline of the ITSO example` as the description.
  - Click **OK**.

Create a VOB and one component

This section describes how to create a component for storing the files that your team develops.

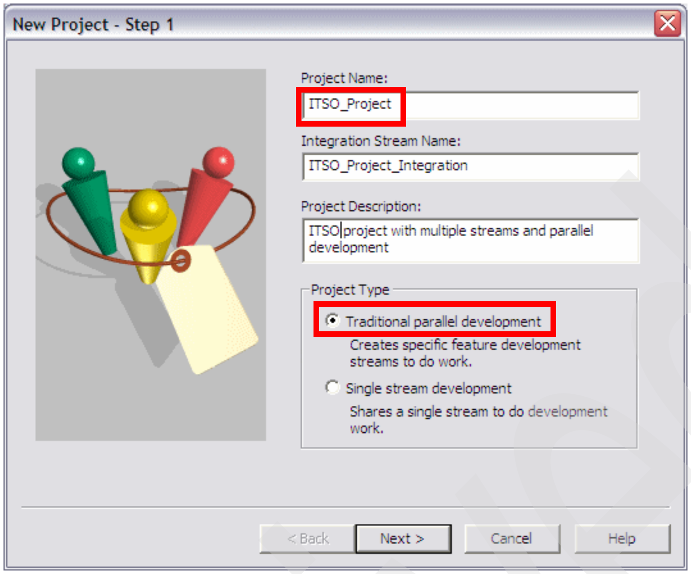
To create a VOB and its one component, do these steps:

- As developer 1, select **Start** → **Programs** → **IBM Rational** → **IBM Rational ClearCase** → **Administration** → **Create VOB**. The path might be different on your system.
- In the VOB Creation Wizard—Name and Major Parameters dialog, enter `itso_example_vob` as the name and `The main VOB that contains the ITSO example project elements` as the description, and click **Next**.
- In the VOB Creation Wizard—Components dialog, select **Create VOB as a single VOB-level component**, and click **Finish**.
- In the VOB Creation Wizard—Confirmation dialog, review the details, and click **OK**.
- In the Creating VOB Summary, click **Close**.

Create new UCM project

To create a new UCM project, do these steps:

- As developer 1, select **Start** → **Programs** → **IBM Rational** → **IBM Rational ClearCase** → **Project Explorer**.
- Select **File** → **New** → **Folder** and call the new folder `Projects`. You do not have to create a project folder, but it is a good idea.
- Right-click **Projects** and select **New** → **Project**.
- In the New Project—Step 1 dialog, enter the following data, as shown in Figure 26-4, and click **Next**:
  - Project Name: `ITSO_Project`
  - Integration Stream Name: `ITSO_Project_Integration` {default}
  - Project Description: `ITSO project with multiple streams and parallel development`
  - Select **Traditional parallel development** {default}



New Project - Step 1

Project Name: ITSO_Project

Integration Stream Name: ITSO_Project_Integration

Project Description: ITSO project with multiple streams and parallel development

Project Type

● Traditional parallel development

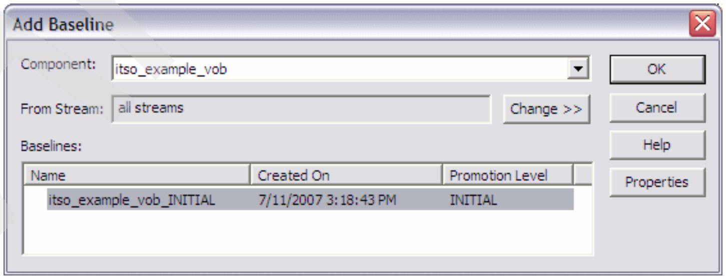
Creates specific feature development streams to do work.

○ Single stream development

Shares a single stream to do development work.

Figure 26-4 New Project wizard—Step 1

- In the New Project—Step 2 dialog, ensure that **No** is selected, and click **Next**. Since we are creating a project from scratch, and we do not have any other existing projects at that time, we do not select a project to seed this project as the baseline.
- In the New Project—Step 3 dialog, click **Add** and make the following steps, as shown in Figure 26-5:
  - In the Add Baseline dialog, click **Change >>** and select **All Streams**.
  - Select the component `itso_example_vob` from the Component drop-down list, and select `itso_example_vob_INITIAL` under Baselines.
  - Click **OK**.



Add Baseline

Component: itso_example_vob

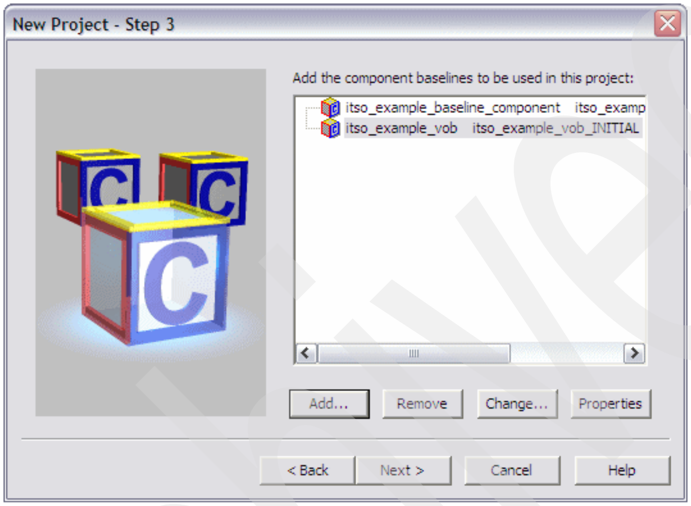
From Stream: all streams

Baselines:

Name	Created On	Promotion Level
itso_example_vob_INITIAL	7/11/2007 3:18:43 PM	INITIAL

Figure 26-5 New Project wizard—Add Baseline dialog

- Click **Add** once more:
  - In the Add Baseline dialog, click **Change >>** and select **All Streams.**
  - Select the component `itso_example_baseline_component` from the Component drop-down list and select `itso_example_baseline_component_INITIAL` under Baselines.
  - Click **OK**. In the New Project wizard—Step 3 dialog, two baselines are listed (Figure 26-6).

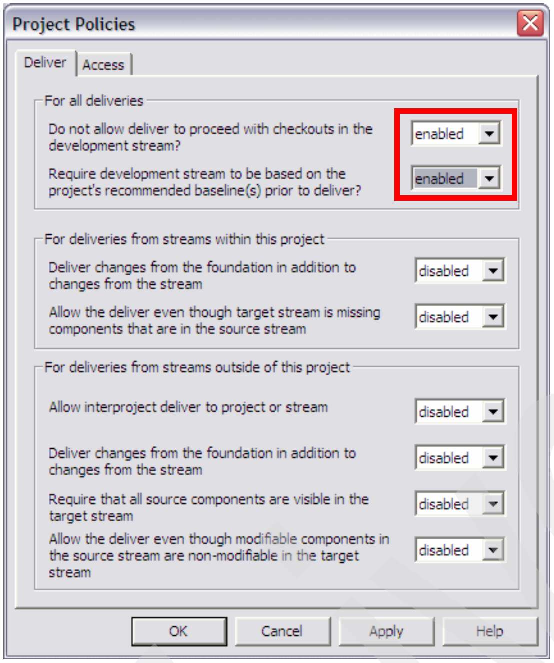

New Project - Step 3

Add the component baselines to be used in this project:

itso_example_baseline_component itso_examp
itso_example_vob itso_example_vob_INITIAL

Figure 26-6 New Project—Step 3 after adding component baselines

  - Click **Next**.
- In the New Project—Step 4 dialog, select `itso_example_vob` in the Make the following components modifiable list and click **Policies**:
  - In the Project Policies **Deliver** tab enable the first two options, and click **OK** (Figure 26-7):
    - The first option ensures that there are no elements left checked out when the developer tries to deliver the current stream. This helps reduce the number of abandoned elements. It enforces the idea if there is an element checked out, either it contains useful code (so we should check it in) or the changes should be undone.
    - The second option ensures that the developers always have the latest stable code in their private work areas.



For all deliveries

Do not allow deliver to proceed with checkouts in the development stream? enabled

Require development stream to be based on the project's recommended baseline(s) prior to deliver? enabled

For deliveries from streams within this project

Deliver changes from the foundation in addition to changes from the stream disabled

Allow the deliver even though target stream is missing components that are in the source stream disabled

For deliveries from streams outside of this project

Allow interproject deliver to project or stream disabled

Deliver changes from the foundation in addition to changes from the stream disabled

Require that all source components are visible in the target stream disabled

Allow the deliver even though modifiable components in the source stream are non-modifiable in the target stream disabled

Figure 26-7 Project Policies—Deliver tab

– Click **Next**.

- In the New Project—Step 5 dialog, select **No** and click **Finish**.
- Click **OK** in the confirmation dialog.

ClearCase now creates the `ITSO_Project` project and it shows up in the Project Explorer in the `Projects` folder.

Joining a UCM project

The next step for developer 1 is to join the project and create a new Web project in Application Developer.

- Start Application Developer and enter `c:\dev1_workspace` as the name of the workspace for developer 1.

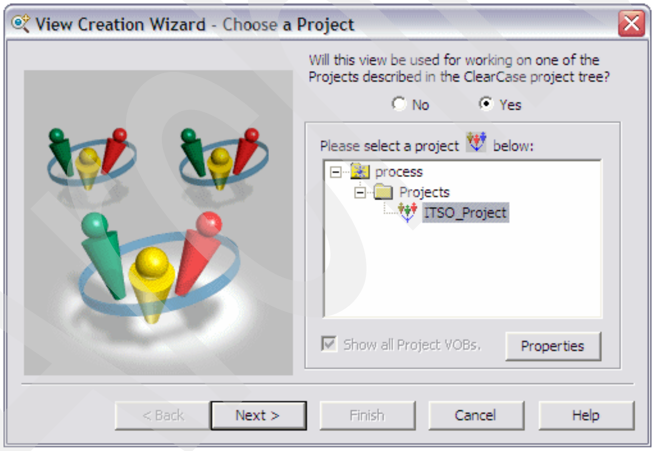
  Do not enable the default workspace option because we are switching workspaces several times during this exercise. Click **OK**.

- – If the Workspace Launcher dialog does not appear, change your Application Developer startup settings under **Window** → **Preferences** → **General** → **Startup and Shutdown**. Make sure the **Prompt for workspace on startup** option is selected.
- – If the Auto Launch Configuration Change Alert dialog opens during startup, informing of an update to your auto launch settings, click **Yes**.

- ► Close the Welcome view.
- ► Enable ClearCase capability (refer to “Enable ClearCase capability” on page 1190).
- ► Select **ClearCase** → **Connect to Rational ClearCase** from the workspace menu or click in the toolbar (unless you specified to automatically connect to ClearCase when Application Developer starts).

Developer 1 creates a development stream and view

- ► Select **ClearCase** → **Create New View**.
- ► In the View Creation Wizard—Choose a Project dialog, select **Yes** to indicate that we are working on a ClearCase project. Select **process** → **Projects** → **ITSO_Project**, and click **Next** (Figure 26-8).



View Creation Wizard - Choose a Project

Will this view be used for working on one of the Projects described in the ClearCase project tree?

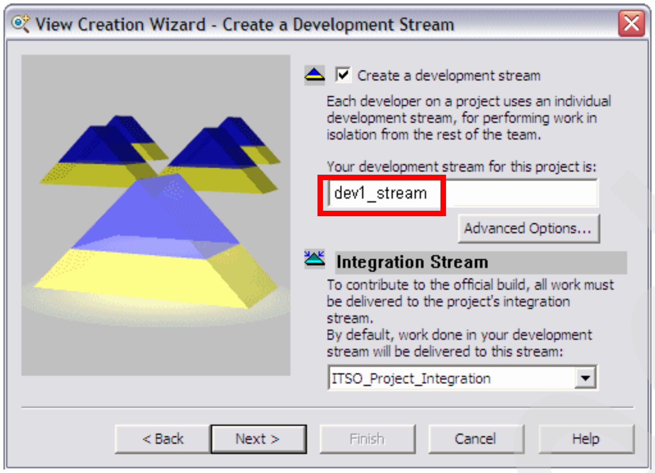
○ No ● Yes

Please select a project below:

☑ Show all Project VOBs.

Figure 26-8 View Creation Wizard—Choose a Project

- ► In the View Creation Wizard—Create a Development Stream dialog, enter **`dev1_stream`** as the development stream name and make sure the integration stream name is `ITSO_Project_Integration` (Figure 26-9), and click **Next**.



View Creation Wizard - Create a Development Stream

☑ Create a development stream

Each developer on a project uses an individual development stream, for performing work in isolation from the rest of the team.

Your development stream for this project is: dev1_stream

Integration Stream

To contribute to the official build, all work must be delivered to the project's integration stream.
By default, work done in your development stream will be delivered to this stream: ITSO_Project_Integration

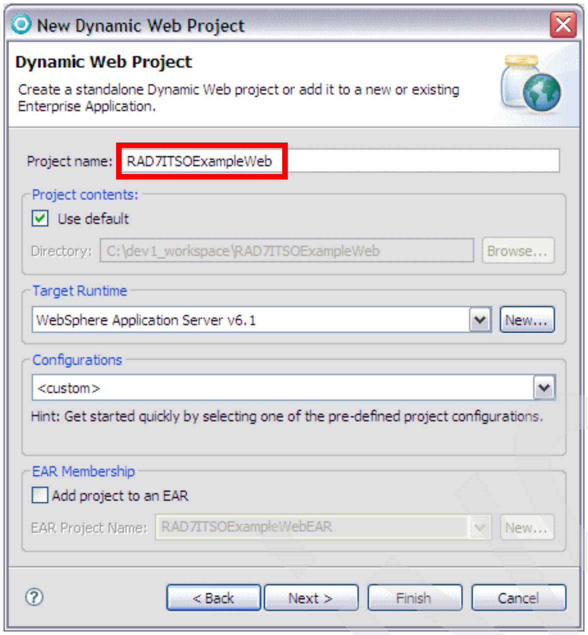
Figure 26-9 View Creation Wizard—Create a Development Stream

- In the View Creation Wizard—Choose Location for a Snapshot View (Development View) dialog, change the location for the root of the view to `c:\clearstorage\views\dev1_View`, and click **Next**.
- In the View Creation Wizard—Choose Location for a Snapshot View (Integration View) dialog, change the location to `c:\clearstorage\views\dev1_IntegrationView`, and click **Next**.
- In the View Creation Wizard—Choose Components dialog, keep `itso_example_vob` selected, and click **Finish**.
- In the Confirm dialog, click **OK**.
- The View Creation Status dialog is displayed after the views have been created. Click **OK**.

Developer 1 creates a Web project

Developer 1 has now created the necessary views and joined the project. The next task is to start actual development and add a new project under ClearCase control. We create a dynamic Web project containing just one servlet for this purpose:

- As developer 1, select **File** → **New** → **Project**. Select **Dynamic Web Project** and click **Next**.
- In the New Dynamic Web Project dialog, enter the following data (Figure 26-10).


New Dynamic Web Project

Dynamic Web Project

Create a standalone Dynamic Web project or add it to a new or existing Enterprise Application.

Project name: RAD7ITSOExampleWeb

Project contents:

☑ Use default

Directory: C:\dev1_workspace\RAD7ITSOExampleWeb

Target Runtime: WebSphere Application Server v6.1

Configurations: <custom>

Hint: Get started quickly by selecting one of the pre-defined project configurations.

EAR Membership

□ Add project to an EAR

EAR Project Name: RAD7ITSOExampleWebEAR

Figure 26-10 New Dynamic Web Project wizard

- Project name: **RAD7ITSOExampleWeb**
- Project contents: Select Use default {default}
- Target Runtime: WebSphere Application Server v6.1 {default}
- Configuration: <custom> {default}
- EAR Membership: Clear **Add project to an EAR**.
- Click **Finish**.

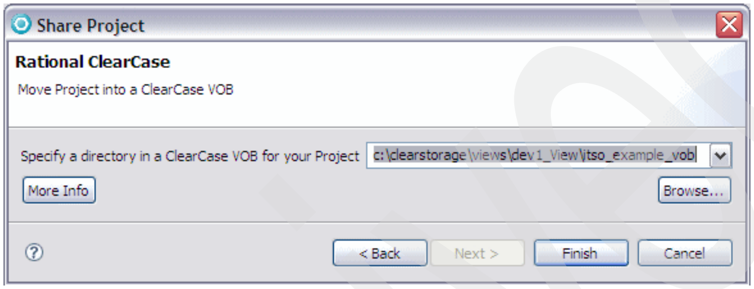
- When prompted, click **Yes** to switch to the Web perspective.
- Close the Web Diagram editor.

> **Important note for the scenario:** Sometimes, when using Application Developer, the user is prompted to check out the `.compatibility` file in a project. This file is used to handle backwards compatibility of projects to previous versions of Application Developer. It seems that some simple activities such as building a project modify this file, and therefore the ClearCase plug-in will offer to check out this resource. For the purposes of demonstrating ClearCase, it is sufficient to cancel this dialog.

Developer 1 adds the project to ClearCase source control

To add the `RAD7ITSOExampleWeb` project under ClearCase control, do these steps:

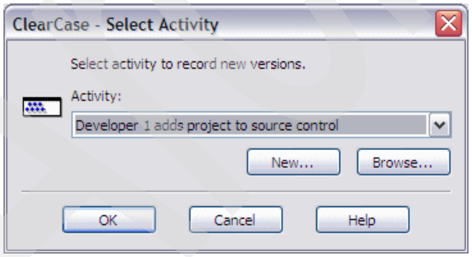
- As developer 1, right-click the **RAD7ITSOExampleWeb** project in the Project Explorer and select **Team** → **Share Project**.
- In the Share Project dialog, select **ClearCase SCM Adapter** and click **Next**.
- In the Share Project—Move Project into a ClearCase VOB dialog, click **Browse**, select `c:\clearstorage\views\dev1_View\itso_example_vob`, and click **OK**. Click **Finish** (Figure 26-11).



Specify a directory in a ClearCase VOB for your Project: c:\clearstorage\views\dev1_View\itso_example_vob

Figure 26-11 Share Project—Moving Project into a ClearCase VOB

- In the Add Element(s) to Source Control dialog, keep all resources selected and make sure that **Keep checked out** is cleared. Click **OK**.
- In the ClearCase—Select Activity dialog, click **New** and enter `Developer 1 adds project to source control`, and click **OK**. Click **OK** to continue (Figure 26-12).



Select activity to record new versions.

Activity: Developer 1 adds project to source control

Figure 26-12 ClearCase—Select Activity dialog

- The Web project is now added to ClearCase source control.

- In Figure 26-13 you can see that the icons belonging to the Web project now have a blue background, and the project name has a view name attached to it. This indicates that the resources are under ClearCase source control.

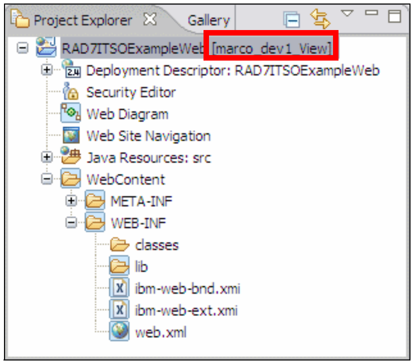


Figure 26-13 Package Explorer with resources under ClearCase source control

- The project contents have been moved from `c:\dev1_workspace\` to `c:\clearstorage\views\dev1_View\itso_example_vob\`. Open Windows Explorer and verify this result.

> **Tip:** It can be useful to create a baseline at this time (see “Developer 1 makes a baseline” on page 1205) and recommend the baseline to all other developers.

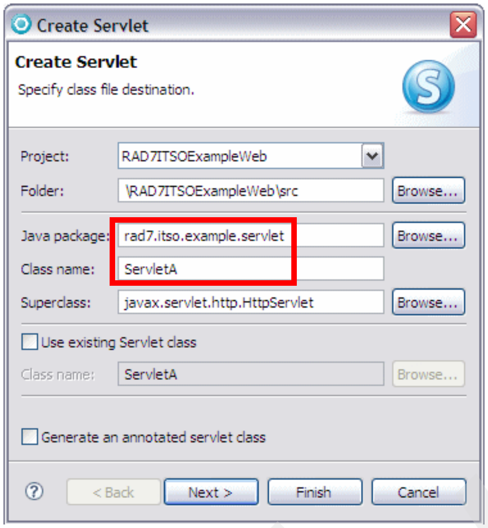
Development scenario

To show how to work with ClearCase, we use a simple scenario where two developers work in parallel on a common project. We use a servlet to illustrate handling a situation when adding an element (the servlet) generates a potential update to some other element(s), such as the deployment descriptor.

Developer 1 adds a servlet

To create a servlet in the `RAD7ITSOExampleWeb` project, do these steps:

- As developer 1, right-click **RAD7ITSOExampleWeb** and select **New** → **Servlet**.
- In the Create Servlet wizard, enter the following data (Figure 26-14).



Create Servlet

Create Servlet

Specify class file destination.

Project: RAD7ITSOExampleWeb

Folder: \RAD7ITSOExampleWeb\src

Java package: rad7.itso.example.servlet

Class name: ServletA

Superclass: javax.servlet.http.HttpServlet

☐ Use existing Servlet class

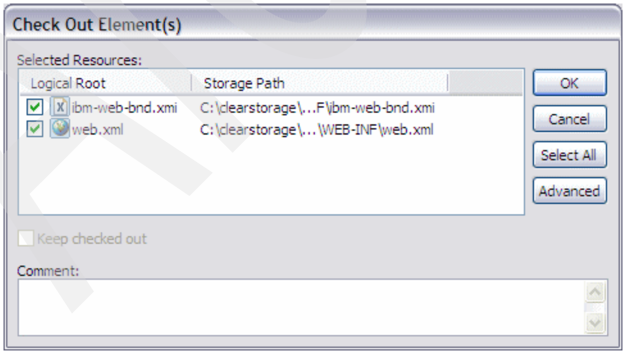
Class name: ServletA

☐ Generate an annotated servlet class

Figure 26-14 Create Servlet wizard

- Project: `RAD7ITSOExampleWeb` {default}
- Folder: `\RAD7ITSOExampleWeb\src` {default}
- Java package: **`rad7.itso.example.servlet`**
- Class name: **`ServletA`**
- Superclass: `javax.servlet.http.HttpServlet` {default}
- Click **Finish**.

- Adding a servlet to the project causes an update to the deployment descriptor and the binding and extension information. You are prompted to check out these files. In the Check Out Element(s) dialog, ensure that all files are selected and click **OK** (Figure 26-15).



Check Out Element(s)

Selected Resources:

Logical Root	Storage Path
☑ ibm-web-bnd.xmi	C:\clearstorage\...F\ibm-web-bnd.xmi
☑ web.xml	C:\clearstorage\...\WEB-INF\web.xml

☐ Keep checked out

Comment:

Figure 26-15 Checking out dependent elements while creating a servlet

- In the ClearCase—Select Activity dialog, click **New** and enter `Developer 1 adds ServletA` as the name of the activity. Click **OK** and again **OK**.
- In the Add Element(s) to Source Control dialog, make sure the `ServletA.java` is selected and that **Keep checked out** is cleared. Click **OK**.
- In the ClearCase—Select Activity dialog, select `Developer 1 adds ServletA` and click **OK**.
- In the Add Element(s) to Source Control dialog, make sure that all package folders are selected and that **Keep checked out** is cleared. Click **OK**.
- In the ClearCase—Select Activity dialog, select `Developer 1 adds ServletA` and click **OK**.
- Close the Java editor of `ServletA`.

Developer 1 checks in all dependent files

There are still dependent files checked out. Expand **RAD7ITSOExampleWeb** → **WebContent** → **WEB-INF** and note the green check marks, as highlighted in Figure 26-16.

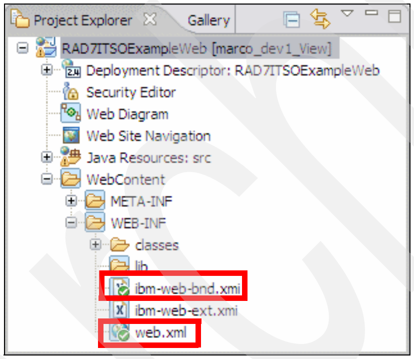
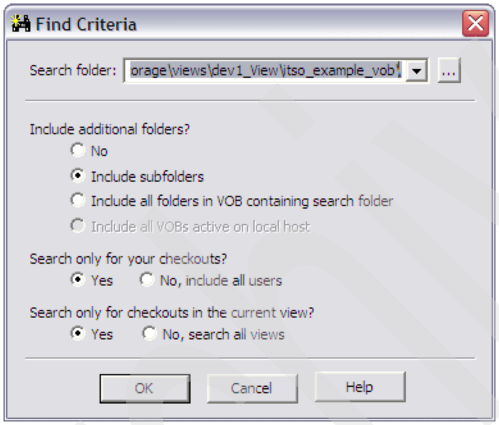


Figure 26-16 Package Explorer with checked out files

Before we deliver the work to the integration stream, we want to check them back in. Do these steps:

- Right-click **web.xml** and select **Team** → **Check in** or select it and click in the toolbar.
- In the Check in Element(s) dialog, ensure that the element is selected and click **OK**. The green check mark on the resource icon is removed, indicating that the file is no longer checked out.

- Before we can deliver the project to the stream, the `ibm-web-bnd.xmi` file must be checked in as well. As its content has not changed, we will simply undo the checkout. Right-click **ibm-web-bnd.xmi** and select **Team** → **Undo Check Out** or select it and click in the toolbar.
- In the Undo Check out Element(s) dialog, ensure that the element is selected and click **OK**. The green check mark on the resource icon is removed, indicating that the file is no longer checked out.
- Before delivering to the stream, it is also good practice to make sure that nothing else is checked out. Select **ClearCase** → **Find Checkouts**.
- In the Find Criteria dialog, enter `c:\clearstorage\views\dev1_View\itso_example_vob\` as Search Folder. Keep the other defaults, and click **OK** (Figure 26-17).



Find Criteria

Search folder: orage\views\dev1_View\itso_example_vob\

Include additional folders?
○ No
● Include subfolders
○ Include all folders in VOB containing search folder
○ Include all VOBs active on local host

Search only for your checkouts?
● Yes ○ No, include all users

Search only for checkouts in the current view?
● Yes ○ No, search all views

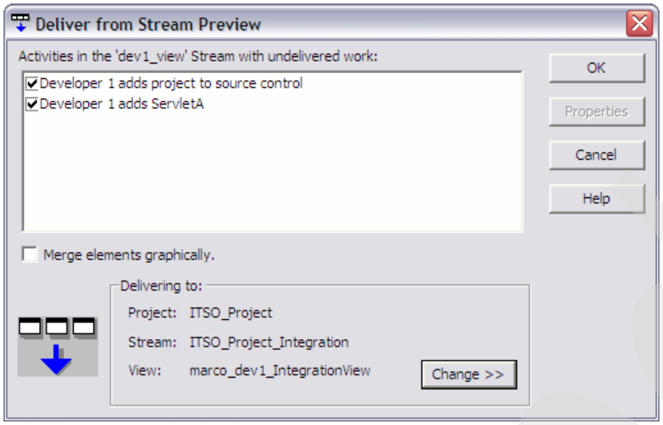
Figure 26-17 Find Criteria dialog [Find checkouts]

- No checked out files are found. Click **OK** to dismiss the dialog and then close the Find Checkouts dialog.

Developer 1 delivers work to the integration stream

To deliver the work into the integration stream, do these steps:

- Select **ClearCase** → **Deliver Stream** or click in the toolbar.
- In the Deliver from Stream Preview dialog, select all activities and make sure that the view to deliver to is `<userid>_dev1_IntegrationView`, and click **OK** (Figure 26-18).


Deliver from Stream Preview

Activities in the 'dev1_view' Stream with undelivered work:

☑ Developer 1 adds project to source control
☑ Developer 1 adds ServletA

☐ Merge elements graphically.

Delivering to:

Project: ITSO_Project
Stream: ITSO_Project_Integration
View: marco_dev1_IntegrationView

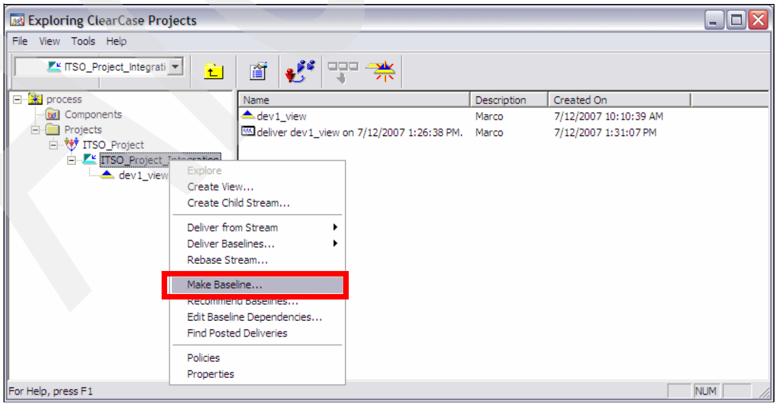
Figure 26-18 Deliver from Stream Preview dialog

- After a while, the Deliver from Stream—Merges Complete dialog opens. Clear **Open a ClearCase Explorer window** and click **OK**.
- In the Delivering to View dialog, click **Complete** and after the work is done **Close**.

Developer 1 makes a baseline

To make a baseline, do these steps:

- As developer 1, select **Start** → **Programs** → **IBM Rational** → **IBM Rational ClearCase** → **Project Explorer**. The path might be different on your system.
- In the left pane right-click **ITSO_Project_Integration** (under **process** → **Projects** → **ITSO_Project**) and select **Make Baseline** (Figure 26-19).


Exploring ClearCase Projects

Name	Description	Created On
dev1_view	Marco	7/12/2007 10:10:39 AM
deliver dev1_view on 7/12/2007 1:26:38 PM.	Marco	7/12/2007 1:31:07 PM

For Help, press F1

Figure 26-19 ClearCase Project Explorer—Make Baseline selection

- In the Make Baseline dialog, keep the defaults, and click **OK**.
- Click **OK** on the confirmation dialog (one new baseline was created), and click **Close** in the Make Baseline dialog.
- Close the ClearCase Project Explorer. Developer 1 has now finished the current task. Developer 2 now joins the project and make changes to the `ServletA`.

Developer 2 joins the project

Developer 2 joins the ClearCase project and adds it to the workspace:

- In the Application Developer, select **File** → **Switch Workspace**.
- In the Workspace Launcher—Select a workspace dialog, enter `c:\dev2_workspace` as the workspace for developer 2 and click **OK**.
- Close the Welcome view.
- Enable ClearCase capability (refer to “Enable ClearCase capability” on page 1190).
- Select **ClearCase** → **Connect to Rational ClearCase** or click in the toolbar (unless you specified to automatically connect to ClearCase when Application Developer starts).

Developer 2 creates a development stream and view

Developer 2 follows these steps:

- Select **ClearCase** → **Create New View**.
- In the Choose a Project dialog, select **Yes** to indicate that we are working on a ClearCase project. Select **process** → **Projects** → **ITSO_Project** and click **Next**.
- In the Create a Development Stream dialog, enter **`dev2_stream`** as the development stream name and make sure the integration stream name is `ITSO_Project_Integration`. Click **Next**.
- In the Review Types of Views dialog, ensure that Create a Development view is selected, and click **Next**.
- In the Choose Location for a Snapshot View (Development View) dialog, change the location to `c:\clearstorage\views\dev2_View`, and click **Next**.
- In the Choose Components dialog, keep `itso_example_vob` selected, and click **Finish**.
- In the Confirm dialog, click **OK**.

- The View Creation Status dialog is displayed after the views have been created. Click **OK**.

Developer 2 imports the project into the workspace

Developer 2 works on the same Web project as developer 1 and has to import it. To import the `RAD7ITSOExampleWeb` project to the workspace, do these steps:

- As developer 2, select **ClearCase** → **Rebase Stream** or click in the toolbar to update your development stream with the contents of the integration stream.
- In the Rebase Stream dialog, select **process** → **Projects** → **ITSO_Project** → **ITSO_Project_Integration** → **dev2_stream**, and click **OK** (Figure 26-20).

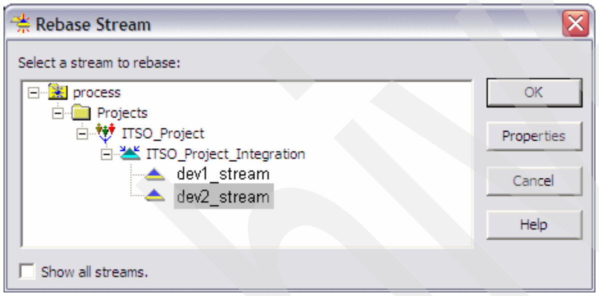


Figure 26-20 Rebase Stream dialog

- In the Rebase Stream Preview dialog, do these steps:
  - Click **Advanced**.
  - In the Change Rebase Configuration dialog, click **Add**.
  - In the Add Baseline dialog, select `itso_example_vob` in the Component drop-down list and select `ITSO_Project_{date}` from the Baselines list. Click **OK**.
  - In the confirmation dialog, click **OK**.
  - In the Change Rebase Configuration dialog, select `itso_example_vob ITSO_Project_{date}` as the baseline to rebase to, and click **OK**.
  - Verify that `<userid>_dev2_View` is selected in the Merge work to the following view field, and click **OK**.
- In the Rebase Stream—Hijacked Files Warning dialog, click **OK**.
- If you receive a warning saying *No versions require merging in stream dev2_view*, dismiss the dialog. We have not made any changes, and do not want to check in any changes.

- In the Rebasing in View dialog, click **Complete** to perform the rebase action. After this is done, click **Close**.

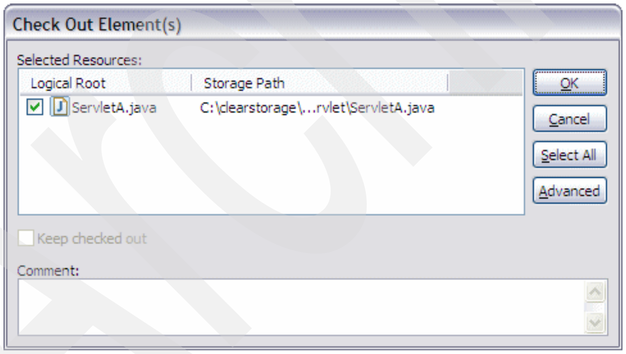
  The contents of the integration view have now been copied to the `dev2_View`, but do not yet appear in the workspace.

- Select **File** → **Import** → **General** → **Existing Projects into Workspace** and click **Next**.
- In the Import dialog, click **Browse**, select the `c:\clearstorage\views\dev2_View\itso_example_vob\RAD7ITSOExampleWeb` directory, click **OK**, and then click **Finish**.

Developer 2 modifies the servlet

As we only want to show how to work with ClearCase, we do not need to add any real code to the servlet. Adding a simple comment to the servlet will work just as well. Do these steps:

- Expand **RAD7ITSOExampleWeb** → **Java Resources: src** → **rad7.itso.example.servlet** and open **ServletA.java**.
- Start entering some text inside any Javadoc comment area, indicating that developer 2 made this change. As soon as you start typing in the editor, the `ServletA` must be checked out. The dialog asks you to check out the `ServletA.java` file. Click **OK** (Figure 26-21).



Check Out Element(s)

Selected Resources:

Logical Root	Storage Path
☑ ServletA.java	C:\clearstorage\...rvlet\ServletA.java

□ Keep checked out

Comment:

Figure 26-21 Check Out Elements dialog

- In the ClearCase—Select Activity dialog, create a new activity `Developer 2 updates ServletA` and click **OK** to confirm. The file is now checked out, which is indicated by the green check mark on the servlet's icon .
- After making your changes (for example, adding your name as the author), save the servlet, and then close the editor.

- The changed file must now be checked in. Right-click **ServletA.java** and select **Team** → **Check In** or select it and click in the toolbar.
- In the Check in Element(s) dialog, click **OK**. The green check marks of the resources icons are removed, indicating that the elements are no longer checked out.

Developer 2 delivers work to the integration stream

Developer 2 is ready to integrate the work. Do these steps:

- As developer 2, select **ClearCase** → **Deliver Stream** or click in the toolbar.
- In the Deliver from Stream Preview dialog, select all activities and make sure that the view to deliver to is `<userid>_dev1_IntegrationView`, and click **OK**.
- After a while, the Deliver from Stream—Merges Complete dialog opens. Clear **Open a ClearCase Explorer window**, and click **OK**.
- In the Delivering to View dialog, click **Complete**, and after the work is done, click **Close**.

Developer 1 modifies the servlet

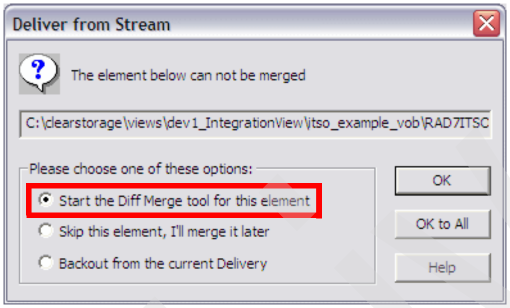
`ServletA` has now been updated once in the integration stream since the original baseline was created. Let us see what happens when another developer makes changes to the same element and makes delivery:

- In Application Developer, select **File** → **Switch Workspace**.
- In the Workspace Launcher—Select a workspace enter `c:\dev1_workspace` as the workspace for developer 1 and click **OK**.
- Select **ClearCase** → **Connect to Rational ClearCase** or click in the toolbar (unless you specified to automatically connect to ClearCase when Application Developer starts).
- Expand **RAD7ITSOExampleWeb** → **Java Resources: src** → **rad7.itso.example.servlet** and open **ServletA.java**.
- Make an update to the comment as developer 1. The servlet will be checked out again.
- In the ClearCase—Select Activity dialog, create a new activity, `Developer 1 updates ServletA`, and click **OK** to confirm.
- After making your changes (use some other text than previously), save the servlet and then close the editor.
- Check in the updated servlet.

Developer 1 delivers new work to the integration stream

Developer 1 is now ready to integrate the work as well. Do these steps:

- As developer 1, select **ClearCase** → **Deliver Stream** or click in the toolbar.
- In the Deliver from Stream Preview dialog, select all activities and make sure that the view to deliver to is `<userid>_dev1_IntegrationView`, and click **OK**.
- ClearCase notifies you that there is a conflict and the element cannot be merged automatically (Figure 26-22).



Deliver from Stream

The element below can not be merged

C:\clearstorage\views\dev1_IntegrationView\itso_example_vob\RAD7ITSC

Please choose one of these options:

● Start the Diff Merge tool for this element
○ Skip this element, I'll merge it later
○ Backout from the current Delivery

Figure 26-22 Deliver from Stream alert

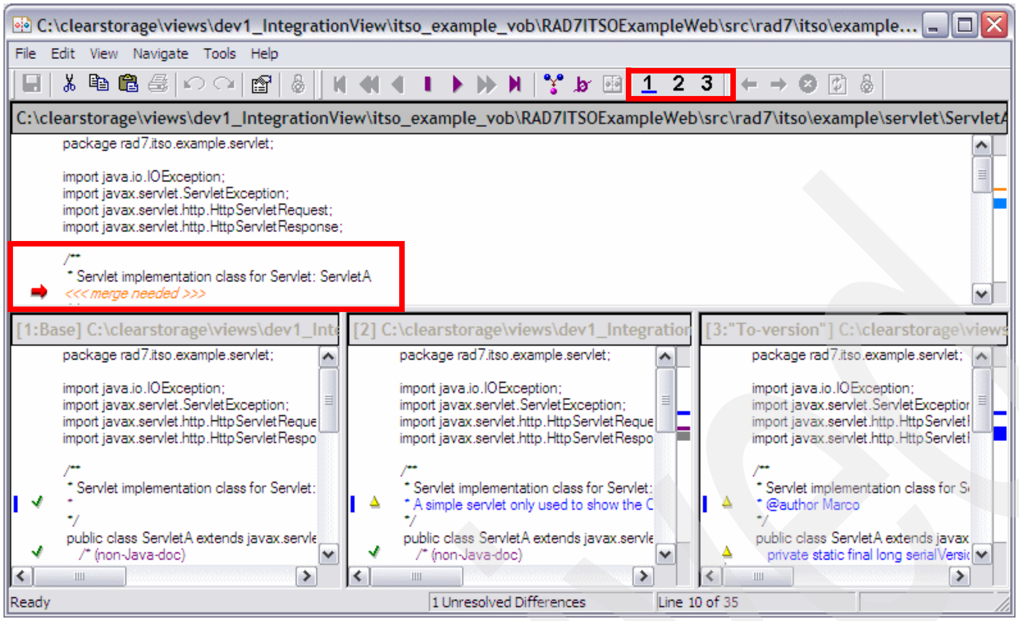
- Select **Start the Diff Merge tool for this element** and click **OK**. Click **OK** again at the Diff Merge Status dialog.
- The Diff Merge tool is now launched (Figure 26-23).

  Take a few moments to become familiar with the information displayed and the options available to you:

  – The top panel shows the merge result and areas of conflict. Click a line to focus on that particular conflict.

  – The lower panels show the different versions that are available.

  – The number icons on the toolbar (1, 2, 3) are used to indicate which version should be used in the merged result.


C:\clearstorage\views\dev1_IntegrationView\itso_example_vob\RAD7ITSOExampleWeb\src\rad7\itso\example...
File Edit View Navigate Tools Help
1 2 3
Ready
1 Unresolved Differences
Line 10 of 35

C:\clearstorage\views\dev1_IntegrationView\itso_example_vob\RAD7ITSOExampleWeb\src\rad7\itso\example\servlet\ServletA

```
package rad7.itso.example.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
 * Servlet implementation class for Servlet: ServletA
<<< merge needed >>>
```

[1:Base] C:\clearstorage\views\dev1_Inte

```
package rad7.itso.example.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletReque
import javax.servlet.http.HttpServletRespo

/**
 * Servlet implementation class for Servlet:
 *
 */
public class ServletA extends javax.servle
  /* (non-Java-doc)
```

[2] C:\clearstorage\views\dev1_Integration

```
package rad7.itso.example.servlet;

import java.io.IOException;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletReque
import javax.servlet.http.HttpServletRespo

/**
 * Servlet implementation class for Servlet:
 * A simple servlet only used to show the C
 */
public class ServletA extends javax.servle
  /* (non-Java-doc)
```

[3:"To-version"] C:\clearstorage\views

```
package rad7.itso.example.servlet;

import java.io.IOException;
import javax.servlet.ServletExceptior
import javax.servlet.http.HttpServletl
import javax.servlet.http.HttpServletl

/**
 * Servlet implementation class for S
 * @author Marco
 */
public class ServletA extends javax
   private static final long serialVersic
```

Figure 26-23 Diff Merge tool

- In the toolbar, select **2** and **3** to determine that in this case the implementation of developer 1 and developer 2 should be merged. Verify the change in the merge panel.
- Close the Merge Tool by saving the changes (select **File** → **Save**) and close the tool.
- After a while, the Deliver from Stream—Merges Complete dialog opens. Clear **Open a ClearCase Explorer window** and click **OK**.
- In the Delivering to View dialog, click **Complete** and after the work is done **Close**.

Tip: For more complex projects, it is good practice to use a new workspace, import the projects (from ClearCase), and verify that nothing has been broken by the integration.

Version tree

Now all the changes to `ServletA` have been applied to the integration stream. You can verify this by looking at the element in the version tree.

To invoke the Version Tree Browser, do these steps:

- Right-click **ServletA.java** in the Project Explorer and select **Team** → **Show Version Tree**.
- The ClearCase Version Tree Browser is launched (Figure 26-24).

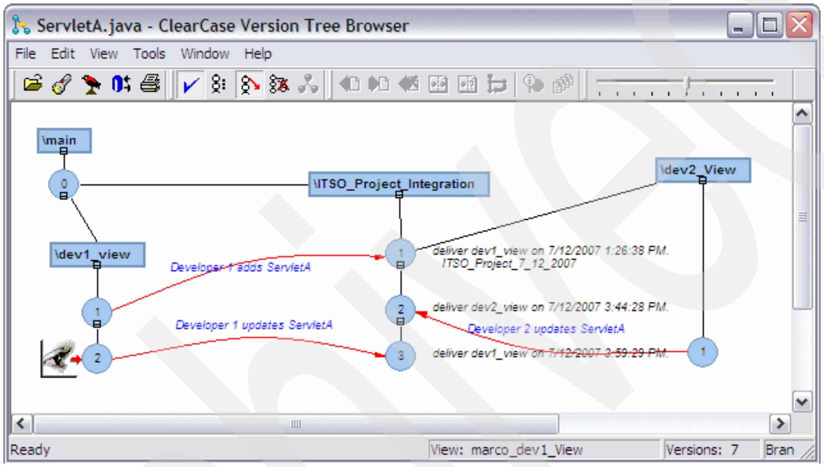

ServletA.java - ClearCase Version Tree Browser
File
Edit
View
Tools
Window
Help
\main
0
\ITSO_Project_Integration
\dev2_View
\dev1_view
Developer 1 adds ServletA
1
deliver dev1_view on 7/12/2007 1:26:38 PM.
ITSO_Project_7_12_2007
1
2
deliver dev2_view on 7/12/2007 3:44:28 PM.
Developer 1 updates ServletA
Developer 2 updates ServletA
2
3
deliver dev1_view on 7/12/2007 3:59:29 PM.
1
Ready
View: marco_dev1_View
Versions: 7
Bran

Figure 26-24 ClearCase Version Tree Browser [ServletA]

Summary

In this chapter we described how to set up ClearCase LT and use the product in a team under Application Developer V7. We used a very simple scenario with two developers to illustrate the major activities when interacting with ClearCase LT.

More information

For more information on ClearCase LT and its integration with Application Developer, refer to these resources:

- *Software Configuration Management: A Clear Case for IBM Rational ClearCase and ClearQuest® UCM*, SG24-6399
- Application Developer Help, under Working in a team environment

27

CVS integration

This chapter provides an introduction to the widely adopted open source version control system known as Concurrent Versions System (CVS) and the tools within Rational Application Developer V7.0 to integrate with it. Through an example installation and implementation of CVS followed by usage scenarios, including two developers working on a simulated project, we demonstrate the main features of using CVS from within Application Developer.

The chapter is organized into the following sections:

- Introduction to CVS
- CVSNT Server installation and implementation
- CVS client configuration for Application Developer
- Configuring CVS in Application Developer
- Development scenario
- CVS resource history
- Comparisons in CVS
- Annotations in CVS
- Branches in CVS
- Working with patches
- Disconnecting a project
- Team Synchronizing perspective
- More information

Introduction to CVS

Concurrent Versions System (CVS) is a popular open source Software Configuration Management (SCM) system for source code version control. It can be used by individual developers or very large teams and can be configured to run across the Web or any configuration where the users have TCP/IP access to a CVS Server. CVS allows users to work on the same file simultaneously without locking and provides a facility to merge changes and resolve conflicts when they arise. For these main reasons and the fact that it is free and relatively easy to install and configure, CVS has become very popular both for open source and commercial projects.

The Mircosoft Windows version of CVS—known as CVSNT—while still retaining the original functionality has split off from the original UNIX version and has been enhanced even further, to the extent that enhancements in CVSNT have been ported back to the UNIX version. It can be run on any Windows NT® or later system and allows clients to connect to the CVS Server from many different environments.

Finally, it is important to note that CVS only implements version control and does not handle other aspects of SCM, such as requirements management, defect tracking, and build management. There are other open source projects for performing these functions, and also, the Rational suite of products has a large set of tools that perform the same functions in an integrated way. (See Chapter 26, "ClearCase integration" on page 1185 for an explanation of Rational ClearCase, which also provides source code version control.)

CVS features

Some of the main features of CVS are as follows:

- Multiple client-server protocols over TCP/IP, a feature that allows developers access to the latest code from a wide variety of clients located anywhere with access to the CVS Server.

Note: Application Developer V7.0 supports four authentication protocols when establishing a connection to a CVS Server:

- **pserver (password server)**—This is the simplest but least secure communication method. Using this mechanism the user name and password is passed to the CVS Server using a defined protocol. The downside is that the password is transmitted in clear text over the network, making it vulnerable to network sniffing tools.
- **ext (external)**—This method allows the user to specify an external program to connect to the repository. On the preferences page **Window** → **Preferences** → **Team** → **CVS** → **Ext Connection Method**, a user can specify the local application to run, to send commands to the CVS Server and the parameters to pass to it. Each CVS operation performed is then done through this executable, and passwords are encrypted using whatever mechanism the executable uses. Generally this mechanism requires shell accounts on the server machine so that the clients can make use the remote shell applications.
- **extssh (external program using secure shell)**—This method is similar to the **ext** method from above, but uses a built-in SSH client supplied with Application Developer to perform encryption. The preferences page **Window** → **Preferences** → **Team** → **CVS** → **SSH2 Connection Method** provides a set of options for configuring the ssh security including the ssh application path and some private keys.
- **pserverssh2 (password server using secure shell)**—This method uses the **pserver** mechanism of storing the users and passwords within the CVS Server as above, but uses ssh encryption (as configured in the SSH2 connection method preferences page) to communicate the user and password information to the server.

For more information on these configurations, refer to the CVS home page and Application Developer help. The option **pserver** is the easiest to configure and is used in the example in this chapter.

- All the versions of a file are stored in a single repository file using forward-delta versioning, which stores only the differences between sequential versions.
- Developers are insulated from each other. Every developer works in their own directory, and CVS merges the work in the repository when they are ready to commit. Conflicts can be resolved as development progresses (using synchronize) and must be resolved before any piece of work is committed to the repository.

Important: CVS and Application Developer have a slightly different interpretation of what the term conflict means:

- In CVS, a conflict means that two changes have been made to the same line or set of lines in the same source code file. In these cases an automatic merge is not possible and some manual merging is required.
- For Application Developer, a conflict means only that there exists a locally modified version of a resource for which a more recent revision is available in the branch in the repository. In these cases an automatic merge might resolve the issue, or if there are changes to the same set of lines then (as is the case for resolving CVS conflicts), manual merging is required.

- It uses an unreserved checkout approach to version control that helps avoid artificial conflicts common when using an exclusive checkout model.
- It keeps shared project data in repositories. Each repository has a root directory on the file system.
- CVS maintains a history of the source code revisions. Each change is stamped with the time it was made and the user name of the person who made it. We recommend that developers also provide a description of the change. Given this information, CVS can help developers find answers to questions such as: Who made the change, when was it made, the reasons why, and what specifically was changed.

CVS support within Application Developer

Application Developer provides a fully integrated CVS client, the main features of which are demonstrated throughout this chapter. At the highest level, there are two perspectives in Application Developer that are important when working with CVS:

- **CVS Repository Exploring perspective**—Provides an interface for creating new links to a repository, browsing the content of the repository, and checking out content (files, folders, projects) to the workspace.
- **Team Synchronizing perspective**—Provides an interface for synchronizing code on the workspace with code in a repository. The perspective is also used by other version control systems, including IBM Rational ClearCase.

In addition to these two perspectives, there are numerous menu options, context menu options, and other features throughout Application Developer to help users work with a CVS repository.

New features available with Application Developer V7.0

The underlying CVS client provided with Application Developer V7.0 is very similar to that provided with V6.0. However, V7.0 is based on Eclipse 3.2 (Version 6.0 was based on Eclipse 3.0) and some of the views have been refined with extra options available. These are as follows:

- **Commit comment templates**—This feature allows programmers to set up a template for their check-in comments. These can contain standard information headings (such as defect number or requirement number) to speed up the check-in process and make sure nothing important is left out of the CVS comment history.
- **Improved patch support**—CVS now has the ability to make patches from multiple projects, to exclude some files from a patch, and to save a patch to the clip-board, file system, or workspace.
- **Improved handling for conflicts when CVS updates**—Now, by default, any non-conflicting files are automatically updated. This can be configured on the **Team** → **CVS** → **Update/Merge** preferences page.
- **CVS shows model content in synchronizations**—(self-explanatory).
- **Expand All button available in CVS Repositories view**—This new menu option fetches the entire CVS tree and displays it expanded in a single operation with the CVS Server.
- **History view improvements**—The history view now combines the local workspace history with the CVS history. New icons have also been added to show different aspects of a given files CVS history. These include:
  - **Link Mode**—Here the History view is linked with the main editor and automatically shows the history for the file being edited.
  - **Local, remote, or both filter**—The history view can be configured to show only local changes, only remote (repository) changes, or both.
  - **Pin history**—This will cause the History view to be locked and any other histories retrieved for other files will be opened in a new History view.
  - **Filters**—It is now possible to filter the CVS history by date, author, or text within the check-in comments.
- **Compare with operation now includes local and remote history**—Previously there were two options on the context menu for performing compares. Now the history list presented includes both local and remote changes.
- **Proxy support for CVS pserver connections**—It is now possible to attach to a CVS Server using pserver protocol through a proxy (this is configured on the **Team** → **CVS** → **Proxy Settings** preferences page).

- **CVS Quick Diff annotations**—A new version of the CVS Annotations view has been provided. The view uses a color coded scheme to show who did what to a file and in which revision.

CVSNT Server installation and implementation

The following CVSNT Server installation and implementation section is organized as follows:

- Installing the CVS Server.
- Configuring the CVS Server repository.
- Creating the Windows users and groups used by CVS.
- Verifying the CVSNT installation.
- Creating CVS users.

Installing the CVS Server

The CVS Server distribution for Linux, UNIX, and Windows platforms is available at the CVS project site, as is installation and usage documentation:

http://www.march-hare.com/cvspro/

At the time of writing, the CVSNT branch 2.5.x branch was the most stable version.

Note: Application Developer does not directly support CVSNT (version 2.5.03), and therefore CVSNT has to be configured to act like a standard CVS Server by setting the **Compatibility** options. This is done in the sample under the step titled “Configuring the CVS Server repository” on page 1219.

We used CVSNT V2.5.03.2382 for the sample because all of our sample applications where developed on the Windows platform, and the CVS home page stated that this was the most stable recent version. We found CVSNT easy to use and did not experience any significant problems using CVSNT, however, the correct configuration was required.

For information on CVS compatibility with Eclipse versions, refer to the following URL:

http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/platform-vcm-home/docs/online/html-cvs/cvs-compatibility.html

To install CVS on the Windows platform, do these steps:

- Before installing CVSNT, refer to these useful installation tips:

  `http://www.cvsnt.org/wiki/InstallationTips`

- Download the CVSNT Server V2.5.03 (`cvsnt-2.5.03.2382.msi`) from the above URL to a temporary directory (for example, `c:\temp`).

  > **Important:** The CVSNT software requires a user performing the installation has local system privileges to install and configure a service in Windows.

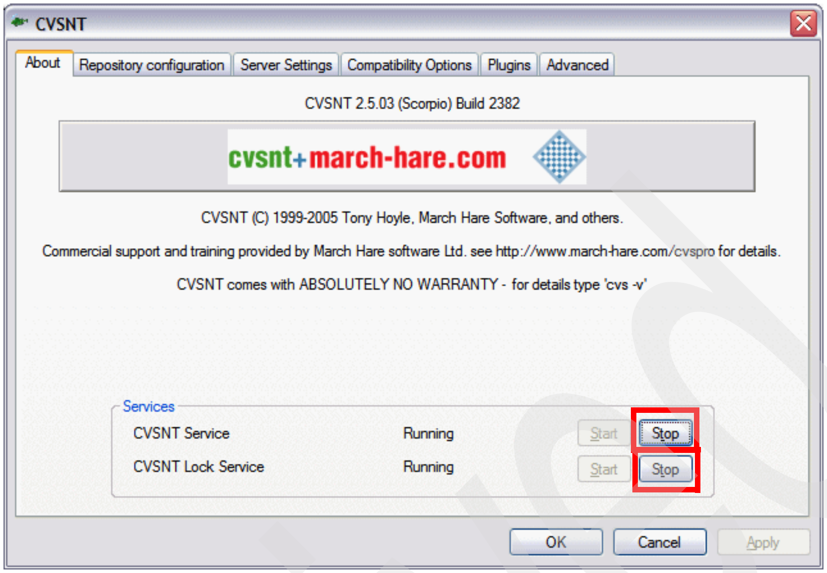
- Execute the CVSNT installer by double-clicking the downloaded `cvsnt-2.5.03.2382.msi` file from the temporary directory.
- In the Welcome dialog, click **Next**.
- In the License Agreement dialog, review the terms, select **I accept the agreement**, and click **Next**.
- In the **Choose Setup Type** dialog, select **Typical**.
- In the **Ready to Install** dialog, click **Install**.
- In the **Completing the CVSNT 2.5.03.2382 Setup Wizard** dialog, click **Finish**.

When the installation is complete, the installation program prompts to see if you want to to perform a restart. We recommend that you restart your system. This step guarantees that the environment variables are set up properly and the CVSNT Windows services are started.

Configuring the CVS Server repository

After you have installed the CVS Server and restarted the system, do the following steps to create and configure the CVS Server repository:

- Manually create the common root directory. For example, the directory `c:\rep7501` can be created using Windows Explorer.
- Select **Start** → **Programs** → **CVSNT** → **Service control panel** to start the CVSNT control application.
- The CVS services must be stopped to create a new repository. From the CVSNT control panel window, stop clicking the services by clicking Stop (Figure 27-1).
  - Click **Stop** under CVS Service.
  - Click **Stop** under CVS Lock Service.



CVSNT 2.5.03 (Scorpio) Build 2382

CVSNT (C) 1999-2005 Tony Hoyle, March Hare Software, and others.

Commercial support and training provided by March Hare software Ltd. see http://www.march-hare.com/cvspro for details.

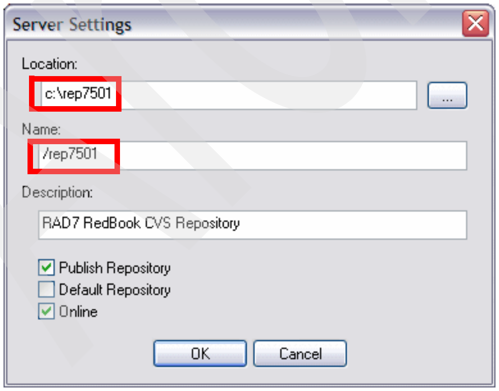
CVSNT comes with ABSOLUTELY NO WARRANTY - for details type 'cvs -v'

Services

CVSNT Service	Running
CVSNT Lock Service	Running

Figure 27-1 Stop the CVSNT services

- Select the **Repository Configuration** tab, and click **Add**.
- In the Server Settings dialog, enter the following values (Figure 27-2):
  - Location: `c:/rep`7501 (we created this directory manually)
  - Name: `/rep7501`
  - Description: `RAD 7 Redbook CVS repository`
  - Leave the other selections with their defaults.
  - Click **OK**.



Location: c:\rep7501

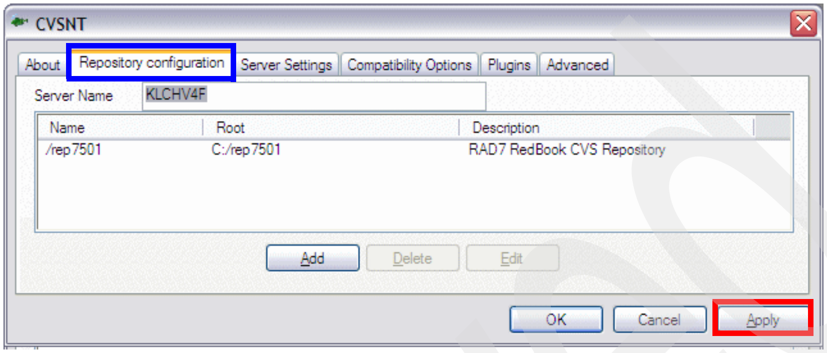
Name: /rep7501

Description: RAD7 RedBook CVS Repository

☑ Publish Repository
☐ Default Repository
☑ Online

Figure 27-2 Add repository

- When prompted with the message *c:/rep7501 exists, but is not a valid CVS repository. Do you want to initialize it?*, click **Yes**.
- The new CVS repository is created. When completed, the Repository page is shown as in Figure 27-3. Click **Apply**.

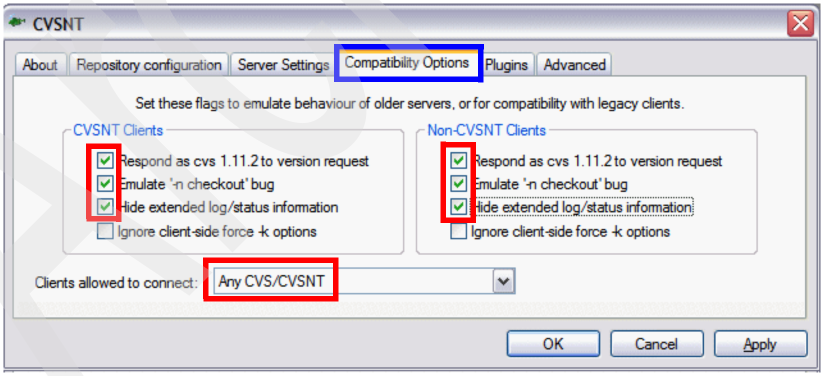


Server Name: KLCHV4F

Name	Root	Description
/rep7501	C:/rep7501	RAD7 RedBook CVS Repository

Figure 27-3 CVSNT service configuration (Repository page)

- Select the **Compatibility Options** tab (Figure 27-4):
  - For both type of clients, use the following options:
    - Select **Respond as cvs 1.11.2 to version request**.
    - Select **Emulate '-n checkout' bug**.
    - Select **Hide extended log/status information**.
    - Clear **Ignore client-side force -k options**
  - For Clients allowed to connect, select **Any CVS/CVSNT**.



Set these flags to emulate behaviour of older servers, or for compatibility with legacy clients.

CVSNT Clients
☑ Respond as cvs 1.11.2 to version request
☑ Emulate '-n checkout' bug
☑ Hide extended log/status information
☐ Ignore client-side force -k options

Non-CVSNT Clients
☑ Respond as cvs 1.11.2 to version request
☑ Emulate '-n checkout' bug
☑ Hide extended log/status information
☐ Ignore client-side force -k options

Clients allowed to connect: Any CVS/CVSNT

Figure 27-4 Compatibility options

These settings ensure that CVSNT is compatible with clients such as Application Developer and other CVS clients.

- Click **OK** to close the CVS control panel.

Creating the Windows users and groups used by CVS

The following instructions configure the user account for CVSNT and associate the CVSNT application with that user.

Add a Windows user (cvsadmin)

To add a Windows CVS administrator user, do these steps:

- From the Windows desktop, right-click **My Computer** (or whatever the label of the computer is on the desktop), and select **Manage**.
- From the **Computer Management** application, expand **Local Users and Groups** and select **Users**.
- From the menu bar, click **Action** → **New User**.
- In the New user window, enter the following values:
  - User name: `cvsadmin`
  - Password: `<password>`. Remember this password for later!
  - Leave Full name blank.
  - Description: User account for the CVSNT application.
  - Clear **User must change password at next logon**.
  - Clear **Account is disabled**.
  - Click **Create** and click **Close.**
- Do not exit the Computer Management tool.

Add Windows user (cvsadmin) to the Administrators group

To add the Windows user to a group that has the sufficient permissions, do these steps:

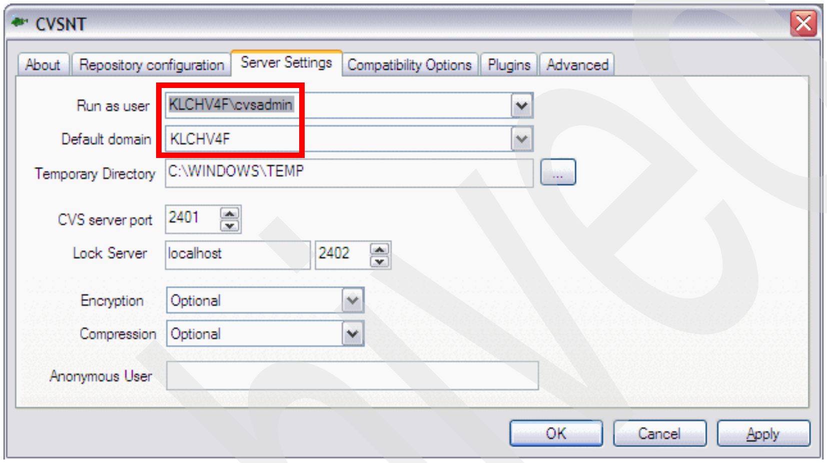
- From the **Computer Management** application, click **Groups** and double-click **Administrators**.
- Click **Add**.
- Enter **cvsadmin** in the **Enter the object names to select** field and click **Check Names.** The user name is verified and prefixed by the local machine name. Click **OK** to add cvsadmin to the Administrators group.
- Click **OK** to exit the Administrators Properties window.
- Exit the Computer Management dialog.

We created a new administrative user for the CVS Server machine to start and stop the CVS administration processes.

Configure the CVS administration user

The CVS services have to be associated with the cvsadmin user to complete the configuration. Do these steps:

- Restart the CVSNT Control Panel application, **Start** → **Programs** → **CVSNT** → **CVSNT Control Panel**.
- Select the **Server Settings** tag and select **cvsadmin** (prefixed by the CVS server machine name) for Run as user and the local machine name for Default domain (Figure 27-5).



Run as user: KLCHV4F\cvsadmin
Default domain: KLCHV4F
Temporary Directory: C:\WINDOWS\TEMP
CVS server port: 2401
Lock Server: localhost 2402
Encryption: Optional
Compression: Optional
Anonymous User:

Figure 27-5 CVSNT Server Settings

- Click **Apply** and **OK**.

Verifying the CVSNT installation

To verify that the CVSNT installation is running as required, do these steps:

- Restart the system, which ensures that the environment variables are loaded and the CVSNT services are started.
- After the system has been restarted, verify that the following CVSNT Windows services have started, by starting the **CVSNT Control Panel**, viewing the **About** tab and checking the status of the following services:
  - CVSNT
  - CVSNT Lock Service

  Both of these services should show as running.

Creating CVS users

To create CVS users to access the files in the repository, do these steps:

- Open a Windows command prompt.
- Set the `cvsroot` environment variable with the following command:

  ```
  set cvsroot=:pserver:cvsadmin@KLCHV4F.ITSOSJ.SANJOSE.IBM.COM:/rep7501
  ```

  Where `KLCHV4F.ITSOSJ.SANJOSE.IBM.COM` is the host name of the CVS Server and `/rep7501` is the directory the repository is located on the host machine.

  > **Note:** The full host name must be specified, localhost does not work. To find out the name of a machine you are using, bring up the context menu of the computer from the Windows desktop, select **Properties** and select the **Computer Name** tab. The full computer name is shown on this page.

- Log on to the CVS repository machine to manage the users, using the following command:

  ```
  cvs login cvsadmin
  ```

- You are prompted to enter the CVS password.

  Enter the password for the cvsadmin user created in "Creating the Windows users and groups used by CVS" on page 1222.

- Enter the following CVS commands to add users:

  ```
  cvs passwd -a -r cvsadmin <cvs user id>
  ```

  – **`cvs passwd -a`** is the command to add a new user and password, or change a password if that user already exists.

  – **`-r cvsadmin`** indicates the alias or native user name that the user will run under when connecting to the repository (we set this to be cvsadmin for the user created in "Creating the Windows users and groups used by CVS" on page 1222).

  – ***`<cvs user id>`*** is the user ID to be added.

  For example, to add user `cvsuser1`, enter the following command:

  ```
  cvs passwd -a -r cvsadmin cvsuser1
  ```

  > **Note:** The first occurrence of a user being added creates the file `passwd` in the directory, `c:\rep7501\CVSROOT`. The new user is appended to this file. We recommend that this file not be edited directly by a text editor. It also must not be placed under CVS control.

- A prompt opens to enter the password:

  ```
  Adding user cvsuser1@klchv4f.itsosj.sanjose.ibm.com
  New password: ********
  Verify password: ********
  ```

  Note that the userid and password created is completely specific to the CVS repository and is unrelated to the windows password.

- Repeat the previous step for additional CVS users.
- Provide the development team members their CVS account information, host name, and connection type to the CVS server, so that they can establish a connection from Application Developer. For example:
  - Account info: Developer CVS user ID (for example, `cvsuser1`)
  - CVS Server host name: (for example, `KLCHV4F.ITSOSJ.SANJOSE.IBM.COM`)
  - Connection type: `pserver`
  - Password: `<password>`
  - Repository path: `/rep7501`

Any subsequent changes to the users or passwords must be done by the administrator using the same commands.

CVS client configuration for Application Developer

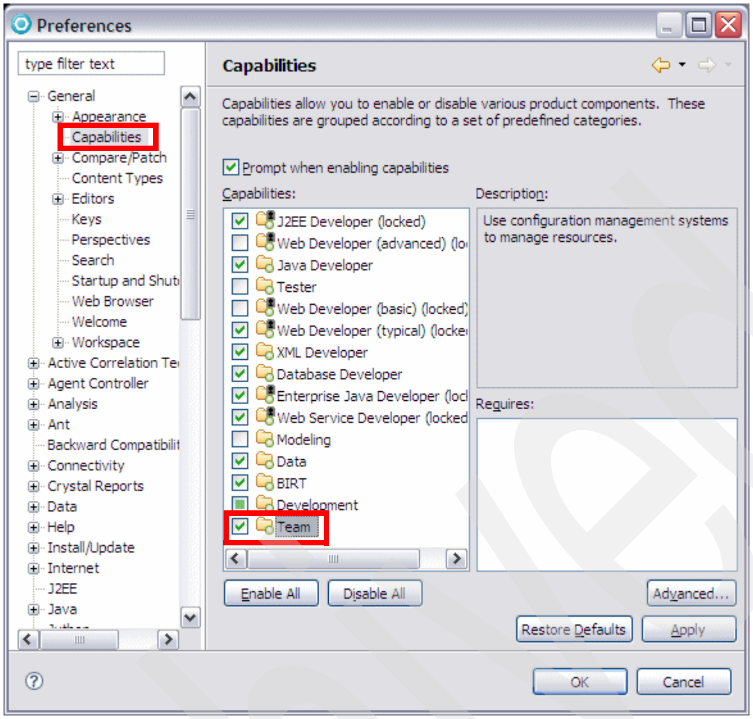
This section describes how to create a client connection within Application Developer to a CVS Server. Typically these activities are done in Application Developer from a new workspace.

Configuring the CVS team capability

Application Developer, by default, does not enable the team capabilities when creating a new workspace for a user.

To enable team capabilities, do these steps:

- Open Application Developer.
- Select **Windows** → **Preferences**.
- Select and expand the **General** and select **Capabilities**.
- Select the **Team** capability.
- Click **Advanced** to selectively enable CVS and ClearCase support.
- Click **Apply** and **OK (**Figure 27-5).



Capabilities allow you to enable or disable various product components. These capabilities are grouped according to a set of predefined categories.

☑ Prompt when enabling capabilities

Capabilities:

- ☑ J2EE Developer (locked)
- ☐ Web Developer (advanced) (lo
- ☑ Java Developer
- ☐ Tester
- ☐ Web Developer (basic) (locked)
- ☑ Web Developer (typical) (locke
- ☑ XML Developer
- ☑ Database Developer
- ☑ Enterprise Java Developer (loc
- ☑ Web Service Developer (locked
- ☐ Modeling
- ☑ Data
- ☑ BIRT
- ■ Development
- ☑ Team

Description:

Use configuration management systems to manage resources.

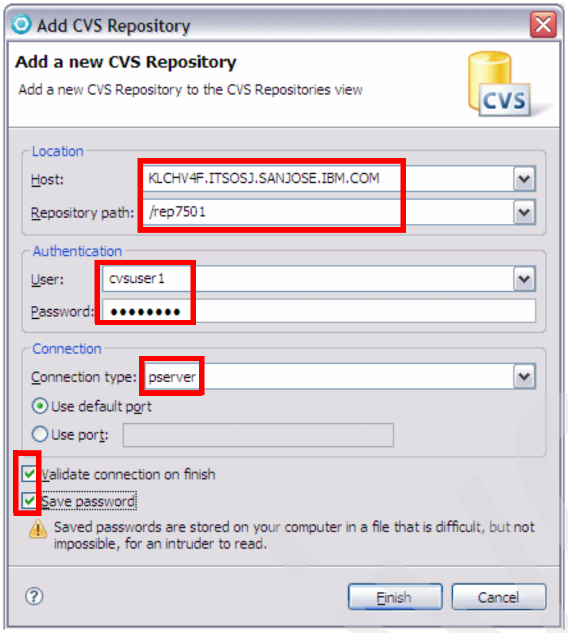
Requires:

Figure 27-6 Setting the Team capability to be available

Accessing the CVS repository

To access a repository that has been configured on a server for users to perform their version management, do these steps:

- Select **Windows** → **Open Perspective** → **Other** → **CVS Repository Exploring**. Click **OK**.
- In the CVS Repositories view, right-click, and select **New** → **Repository Location**.
- Add the parameters for the repository location as shown in Figure 27-7, where Host and Repository path should reflect where the CVS repository is, and User and Password are the name of the user of the work space. Select **pserver** as the connection type.
- Select **Validate Connection Finish** and **Save Password**, and then click **Finish**.



Add CVS Repository

Add a new CVS Repository

Add a new CVS Repository to the CVS Repositories view

Location

Host: KLCHV4F.ITSOSJ.SANJOSE.IBM.COM

Repository path: /rep7501

Authentication

User: cvsuser1

Password: ••••••••

Connection

Connection type: pserver

● Use default port

○ Use port:

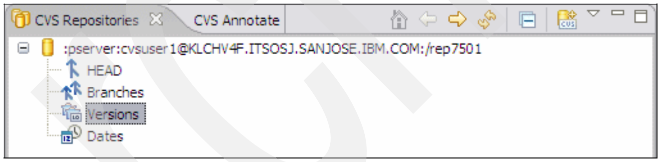
☑ Validate connection on finish

☑ Save password

Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read.

Figure 27-7 Add the CVS repository to the workspace

If everything worked correctly, you can to see a repository location in the CVS Repositories view. The entry can be expanded to show HEAD, Branches, Versions, and Dates (Figure 27-8).



:pserver:cvsuser1@KLCHV4F.ITSOSJ.SANJOSE.IBM.COM:/rep7501
- HEAD
- Branches
- Versions
- Dates

Figure 27-8 CVS Repositories view

Configuring CVS in Application Developer

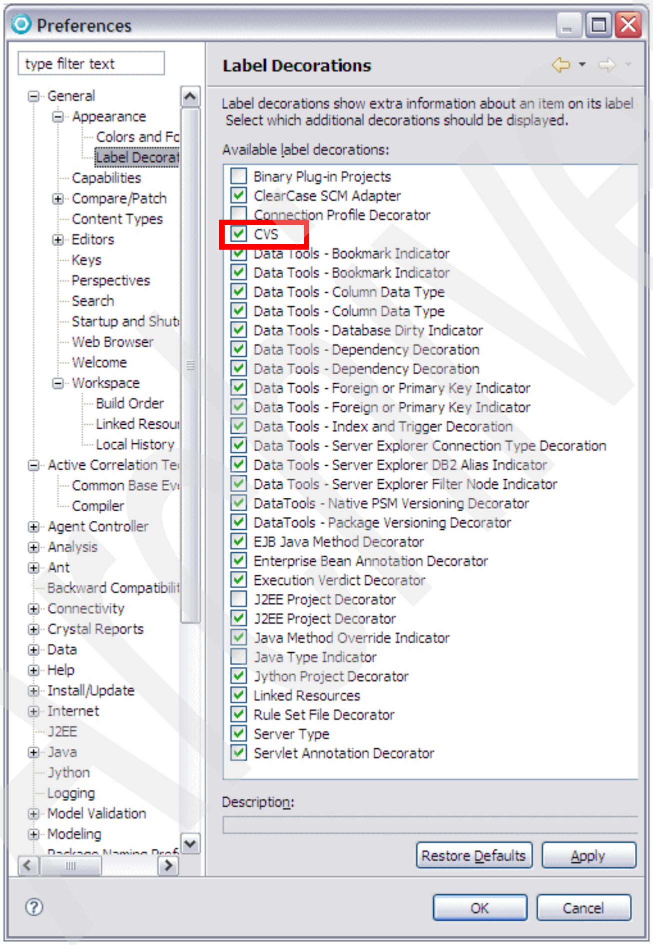
Within Application Developer, it is possible to set the following CVS related settings to guide the integration of CVS:

- Label decorations
- File content
- Ignored resources
- CVS-specific settings
- Configuring CVS keyword substitution

Label decorations

Label decorations are set to be on for CVS by default. This means that the CVS properties of a particular file are shown on its label or icon. For example, if a file has changed from the version in the repository, it will have a **>** symbol next to its file name.

To view or change the label decorations, select **Windows** → **Preferences** and expand **General** → **Appearance** and select **Label Decorations**. By default, the **CVS** labels are selected (Figure 27-9)



Label Decorations

Label decorations show extra information about an item on its label
Select which additional decorations should be displayed.

Available label decorations:

- □ Binary Plug-in Projects
- ☑ ClearCase SCM Adapter
- □ Connection Profile Decorator
- ☑ CVS
- ☑ Data Tools - Bookmark Indicator
- ☑ Data Tools - Bookmark Indicator
- ☑ Data Tools - Column Data Type
- ☑ Data Tools - Column Data Type
- ☑ Data Tools - Database Dirty Indicator
- ☑ Data Tools - Dependency Decoration
- ☑ Data Tools - Dependency Decoration
- ☑ Data Tools - Foreign or Primary Key Indicator
- ☑ Data Tools - Foreign or Primary Key Indicator
- ☑ Data Tools - Index and Trigger Decoration
- ☑ Data Tools - Server Explorer Connection Type Decoration
- ☑ Data Tools - Server Explorer DB2 Alias Indicator
- ☑ Data Tools - Server Explorer Filter Node Indicator
- ☑ DataTools - Native PSM Versioning Decorator
- ☑ DataTools - Package Versioning Decorator
- ☑ EJB Java Method Decorator
- ☑ Enterprise Bean Annotation Decorator
- ☑ Execution Verdict Decorator
- □ J2EE Project Decorator
- ☑ J2EE Project Decorator
- ☑ Java Method Override Indicator
- □ Java Type Indicator
- ☑ Jython Project Decorator
- ☑ Linked Resources
- ☑ Rule Set File Decorator
- ☑ Server Type
- ☑ Servlet Annotation Decorator

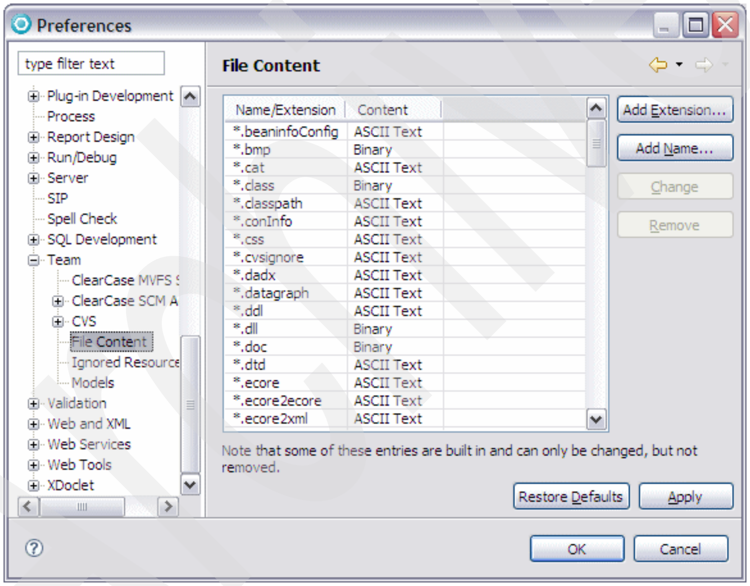
Description:

Figure 27-9 CVS decoration preferences

File content

The file content of resources can be configured to be stored as either ASCII or binary. When working with a file extension that is not defined in the file content list stored in Application Developer, files of this type are saved into the repository as binary by default. When a resource is stored as a binary, CVS cannot show line-by-line comparisons between versions. However, files that have binary content cannot be stored as ASCII CVS files. Also. once a file is created as one type, it cannot be changed. Therefore, it is important to make sure that each file content type is configured correctly in Application Developer before adding a new project to the repository.

To verify that a resource in the workspace is stored in the repository correctly, select **Windows** → **Preferences** and expand **Team** → **File Content** (Figure 27-10). Verify that the file extensions that you are using are present and stored in the repository as desired.



Name/Extension	Content
*.beaninfoConfig	ASCII Text
*.bmp	Binary
*.cat	ASCII Text
*.class	Binary
*.classpath	ASCII Text
*.conInfo	ASCII Text
*.css	ASCII Text
*.cvsignore	ASCII Text
*.dadx	ASCII Text
*.datagraph	ASCII Text
*.ddl	ASCII Text
*.dll	Binary
*.doc	Binary
*.dtd	ASCII Text
*.ecore	ASCII Text
*.ecore2ecore	ASCII Text
*.ecore2xml	ASCII Text

Note that some of these entries are built in and can only be changed, but not removed.

Figure 27-10 Team File Content preferences

If a particular file extension is not in the list, then this extension has to be added, unless the resource is stored in the default binary format. Application Developer prompts the user for the resource type when performing the first check-in (see Figure 27-17 on page 1239) if it encounters a new type, or the new type can be added manually in the Preferences page.

A common file that is often suppled with a source code distribution is a `Makefile.mak` file, which is usually an ASCII file.

To demonstrate adding this file type extension (that is not present in this list), do these steps:

- Select **Windows** → **Preferences** and expand **Team** → **File Content**.
- Click **Add Extension**.
- Enter the extension name `mak` and click **OK**.
- Find the extension in the list, and in the Content column, and select **ASCII** from the drop-down.

> **Tip:** The content can also be changed by highlighting the extension and clicking **Change**. This toggles the setting between ASCII and Binary.

- Click **Apply** and then **OK**.

Ignored resources

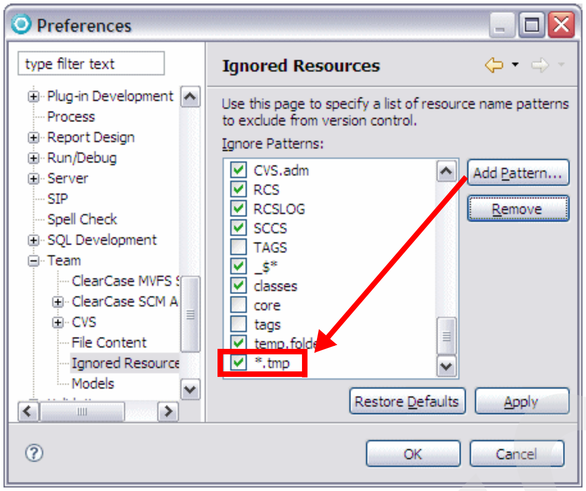
Resources that are created or changed dynamically through mechanisms such as compilation or builds are not recommended to be saved in the repository. This can include class files, executables, and Enterprise JavaBean stub and implementation code.

Application Developer stores a list of these resources that are ignored when performing CVS operations. This is accessed by selecting **Windows** → **Preferences** and expanding **Team** → **Ignored Resources**.

Resources can be added to this list by specifying the pattern that will be ignored. The two wild card characters are an asterisk (*)—which indicates a match of zero or many characters—and a question mark (?)—which indicates a match of one character. For example, a pattern of `_EJS*.java` would match any file that begins with `_EJS` and had zero to many characters and ends in `.java`.

The following example shows the addition of the filename pattern `*.tmp` to the ignored resources list:

- Select **Windows** → **Preferences** and expand **Team** → **Ignored Resources**.
- Click **Add Pattern**.
- Enter the pattern `*.tmp` and click **OK**.
- Ensure that the resource (`*.tmp`) is selected and added to the **Ignored Resources** list (Figure 27-11). `*.tmp` resources are now ignored by CVS in Application Developer.



Use this page to specify a list of resource name patterns to exclude from version control.

Ignore Patterns:

- ☑ CVS.adm
- ☑ RCS
- ☑ RCSLOG
- ☑ SCCS
- □ TAGS
- ☑ _$*
- ☑ classes
- □ core
- □ tags
- ☑ temp.folde
- ☑ *.tmp

Figure 27-11 Resources that will be ignored when saving to the repository

To remove a pattern from the ignore list, select it and click **Remove**. Alternatively, you can temporarily disable ignoring the file pattern by clearing its check box in the list.

Additionally, there are two further facilities that can be used to exclude a file from version control:

- Resources marked as derived are automatically not checked into the CVS repository by Application Developer. This field is set by builders in the Eclipse framework, such as the Java builder. To determine if a resource is derived or not, right-click the resource and select **Properties**, or look in the Properties view. The Derived field is shown under **Info**. It is also possible to change the Derived field value in the properties dialog.
- Use of a `.cvsignore` file. This file contains a list of files or directories that should not be placed into the repository. CVS checks this file and does not add to CVS any files which are in this list.

  A file can be added to the list by right-clicking the file on the Project Explorer and selecting **Team** → **Add to .cvsignore**.

Further details on the syntax of `.cvsignore` can be found at:

http://www.cvsnt.org/manual/html/cvsignore.html

CVS-specific settings

The CVS settings in Application Developer are extensive and cannot be covered in full here. Some of the more important settings are highlighted in Table 27-1 with short descriptions. A complete description of the remaining settings can be obtained from the Application Developer help system.

Table 27-1 Category of CVS settings available

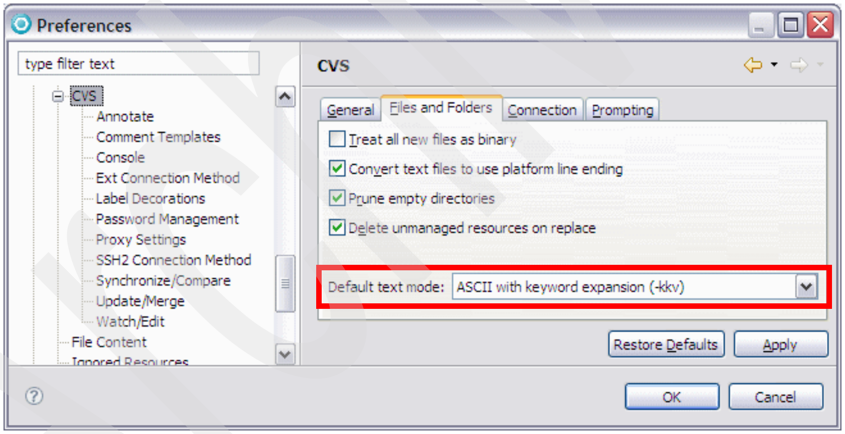
Category	Menu location	Description
General CVS Settings	**Windows → Preferences → Team → CVS**	Settings for the behavior in communicating with CVS, handling the files and projects received from CVS and when to prompt the user for certain activities.
Console	**Windows → Preferences → Team → CVS → Console**	Various settings for the CVS console view including a flag to for whether to display CVS commands to the console.
Ext Connection Method	**Windows → Preferences → Team → CVS → Ext Connection Method**	Settings to identify the ssh external program and associated parameters when using the ext protocol to communicate with the CVS Server.
Label Decorations	**Windows → Preferences → Team → CVS → Label Decorations**	Settings for how to display the CVS state of resources in Application Developer.
Password Management	**Windows → Preferences → Team → CVS → Password Management**	Manages which repositories have passwords saved within Application Developer.
SSH2 Connection Method	**Windows → Preferences → Team → CVS → SSH2 Connection Method**	Configuration settings for SSH2 protocol to the CVS repository.
Update/ Merge	**Windows → Preferences → Team → CVS → Update/Merge**	Settings for guiding the Application Developer process when merging is required during synchronization.
Watch/Edit	**Windows → Preferences → Team → CVS → Watch/Edit**	Settings for the CVS watch and edit functionality which allows users to be informed (via e-mail) when a file has been edited or committed by another user.

CVS keyword substitution

In addition to storing the history of changes to a given source code file in the CVS repository, it is also possible to store meta-information (such as author, date/time, revision and change comments) in the contents of the file. Typically a standard header template is configured in Application Developer, which is then applied to each file when CVS operations (usually check-in and checkout) are performed. The template can include a set of CVS keywords inside a heading, which are expanded out when a file is checked in or out. This is known as keyword expansion.

Keyword expansion is an effective mechanism for developers to quickly identify what version a resource is in the repository versus what a user has checked out locally on their workspace.

Application Developer, by default, has the keyword substitution set to *ASCII with keyword expansion (-kkv)* under the selection **Windows** → **Preferences** → **Team** → **CVS** and the **File and Folders tab** (Figure 27-12). This setting expands out keyword substitution based on the interpretation by CVS, and is performed wherever the keywords are located in the file.



☐ Treat all new files as binary
☑ Convert text files to use platform line ending
☑ Prune empty directories
☑ Delete unmanaged resources on replace

Default text mode: ASCII with keyword expansion (-kkv)

Figure 27-12 CVS Keyword expansion setting

Some of the available keywords (case sensitive) are listed in Table 27-2.

Table 27-2 CVS keywords

Keyword	Description
$Author$	Expands to the name of the author of the change in the file, for example: `$Author: itsodev $`
$Date$	Expands to the date and time of the change in UTC, for example: `$Date: 2007/06/27 18:21:32 $`
$Header$	Contains the CVS file in repository, revision, date (in UTC), author, state and locker, for example: `$Header: /rep7501/XMLExample/.project,v 1.1 2007/06/27 18:21:32 itsodev Exp itso $`
$Id$	Like $Header$ except without the full path of the CVS file, for example: `$Id: .project,v 1.1 2007/06/27 18:21:32 itsodev Exp itso $`
$Log$	The log message of this revision. This does not get replaced but gets appended to existing log messages.
$Name$	Expands to the name of the sticky tag, which is a file retrieved by date or revision tags, for example: `$Name: version_1_3 $`
$Revision$	Expands to the revision number of the file, for example: `$Revision: 1.1 $`
$Source$	Expands to the full path of the RCS file in the repository, for example: `$Source: /rep7501/XMLExample/.project,v $`

To ensure consistency between multiple users working on a team, it is recommended that a standard header is defined for all Java source files. A simple example is shown in Example 27-1.

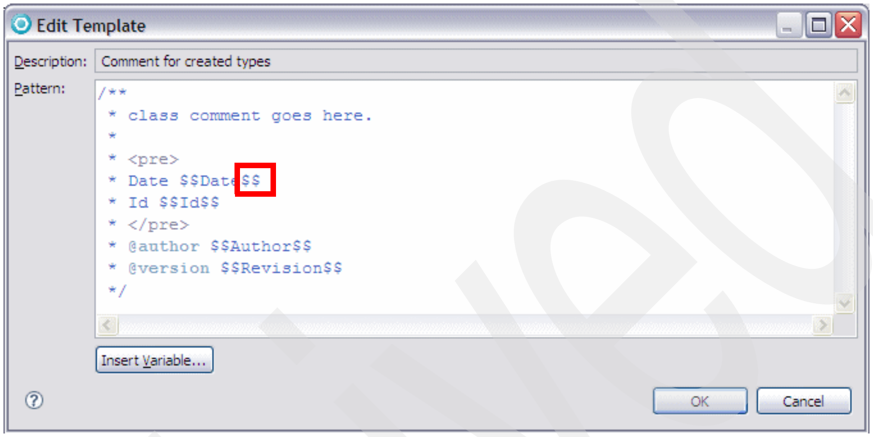
Example 27-1 Example of CVS keywords used in Java

```
/**
 * class comment goes here.
 *
 * <pre>
 * Date $Date$
 * Id $Id$
 * </pre>
 * @author $Author$
 * @version $Revision$
 */
```

To ensure consistency across all files created, each user would have to cut and paste this into their document. Fortunately, Application Developer offers a means to ensure this consistency.

To set up a standard template, do these steps:

- Select **Windows** → **Preferences** → **Java** → **Code Style** → **Code Templates**.
- Expand **Comments** → **Files** and click **Edit**.
- Cut and paste or type what comment header you require (Figure 27-13).



Description: Comment for created types

Pattern:

```
/**
 * class comment goes here.
 *
 * <pre>
 * Date $$Date$$
 * Id $$Id$$
 * </pre>
 * @author $$Author$$
 * @version $$Revision$$
 */
```

Figure 27-13 Setup of a common code template for Java files

- Click **OK** to complete the editing, then click **Apply** followed by **OK**.

> **Note:** The double dollar sign ($$) is required because Application Developer treats a single dollar ($) as one of its own variables. The double dollar ($$) is used as a means of escaping the single dollar so that it can be post processed by CVS.

This sets up a standard CVS template. The next time a new class is created, checked in, and then checked out, the header is displayed (Example 27-2).

Example 27-2 Contents of Java file after check in and check out from CVS

```
/**
 * class comment goes here.
 *
 * <pre>
 * Date $Date: 2004/10/29 18:21:32 $
 * Id $Id: $Id: Example.java,v 1.1 2004/10/29 18:21:32 itsodev Exp itso $
 * </pre>
 * @author $Author: itsodev $
 * @version $Revision: 1.1 $
 */
```

Development scenario

> **Note:** The example in this chapter calls for two simulated developer systems. For demonstration purposes, this can be accomplished by having two workspaces on the same machine. Refer to “Workbench basics” on page 74 for detailed instructions on setting up multiple workspaces.

To show you how to work with CVS in Application Developer, we follow a simple but typical development scenario, shown in Table 27-3.

Two developers, cvsuser1 and cvsuser2, work together to create a servlet *ServletA* and a view bean *View1*.

Table 27-3 Sample development scenario

Step	Developer 1 (cvsuser1)	Developer 2 (cvsuser2)
1	Creates a new Dynamic Web Project `ISTOCVSGuide` and adds it to the version control and the repository. Creates a servlet `ServletA` and commits it to the repository.	
2	**2b**: Updates the servlet `ServletA`.	**2a**: Imports the `ISTOCVSGuide` CVS module as a Workbench project. Creates a view bean `View1`, adds it to the version control, and synchronizes the project with the repository.
3	**3a**: Synchronizes the project with the repository to commit the changes to repository (servlet) and receives changes from the repository (view bean).	**3b**: Synchronize the project with the workspace to receive the servlet.
4	**4a**: Continues changing and updating the servlet. Synchronizes the project with the repository to commit his changes to repository and merges changes. **4c**: Synchronize with the repository to pick up the merged servlet.	**4a**: Begins changes to servlet in parallel with developer 1. **4b**: Synchronizes the project after cvsuser1 has committed and has to merge code from the workspace and the CVS repository.
5	Assigns version number the project.	

Steps 1 through 3 are serial development with no parallel work on the same file being done. During steps 4 and 5 both developers work in parallel, resulting in conflicts. These conflicts are resolved using the CVS tools in Application Developer.

In the sections that follow, we perform each of the steps and explain the team actions in detail.

Create and share the project (step 1 - cvsuser1)

Application Developer offers a perspective specifically designed for viewing the contents of CVS servers: The CVS Repository Exploring Perspective.

Add a CVS repository

For this section, ensure that you have completed, “CVSNT Server installation and implementation” on page 1218 and “Accessing the CVS repository” on page 1226.

- ► The CVS Repositories view now contains the repository location (Figure 27-14).

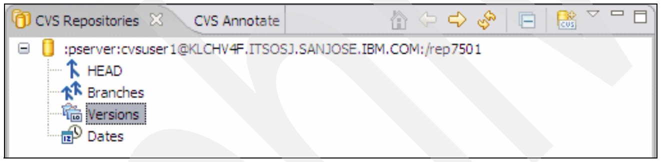


Figure 27-14 CVS Repositories view

Expanding a location in the CVS Repository view reveals branches and versions. A special branch, called HEAD, is shown outside the main branches folder because of its importance. It is the main integration branch, holding the project’s current development state.

The CVS Repositories view can be used to check out repository resources as projects on the Workbench. You can also configure branches and versions, view resource histories, and compare resource versions and revisions.

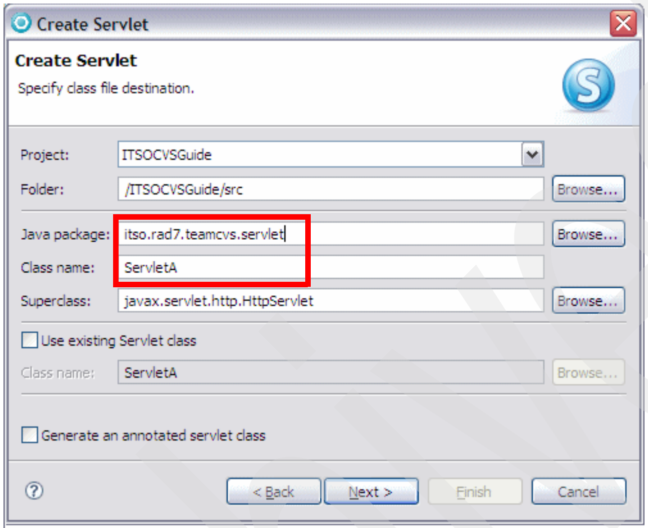
First, a project must be created and shared before full use can be made of the repository.

Create a project and servlet

To create a project and a servlet, do these steps:

- ► Switch to the Web perspective and create a new Dynamic Web Project by selecting **File** → **New** → **Dynamic Web Project**.

- Type in the name of the project, `ITSOCVSGuide`, and click **Finish**.
- In the Project Explorer, expand **ITSOCVSGuide** → **Deployment Descriptor** → **Servlets**. Right-click and select **New** → **Servlet**.
- Specify the package to be `itso.rad7.teamcvs.servlet`.
- Enter `ServletA` for the Class Name field and click **Next** (Figure 27-15).



Create Servlet

Create Servlet

Specify class file destination.

Project: ITSOCVSGuide

Folder: /ITSOCVSGuide/src

Java package: itso.rad7.teamcvs.servlet

Class name: ServletA

Superclass: javax.servlet.http.HttpServlet

☐ Use existing Servlet class

Class name: ServletA

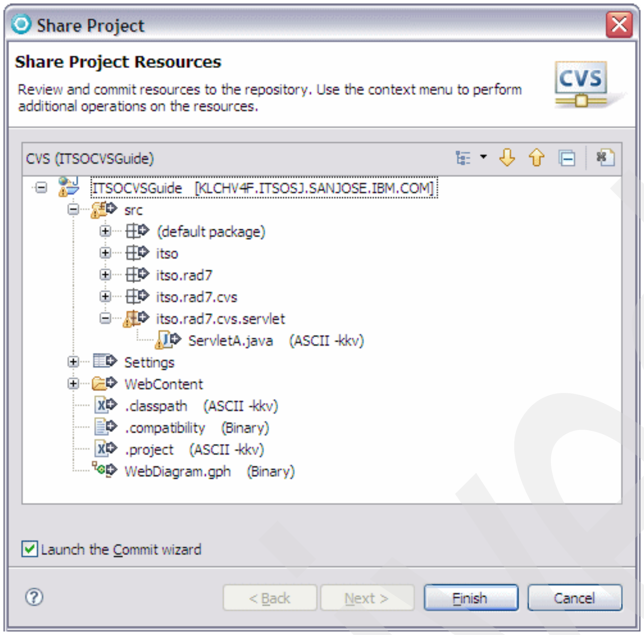
☐ Generate an annotated servlet class

Figure 27-15   Specifying the package name for servlet

Ads the project to the repository

To add the Web project source code to the repository, do these steps:

- In the Project Explorer, right-click the project **ITSOCVSGuide** and select **Team** → **Share Project**.
- In the **Share Project** dialog, select **CVS**, and click **Next**.
- In the Share Project with CVS Repository dialog, select **Use existing repository location** and the repository that is added previously. Click **Next**.
- In the Enter Module Name dialog, select **Use project name as module name**, and click **Next**. A status window might appear as the resources are added to the repository.
- The Share Project Resources dialog (Figure 27-16) opens, listing the resources to be added. Click **Finish**.



Share Project Resources

Review and commit resources to the repository. Use the context menu to perform additional operations on the resources.

CVS (ITSOCVSGuide)

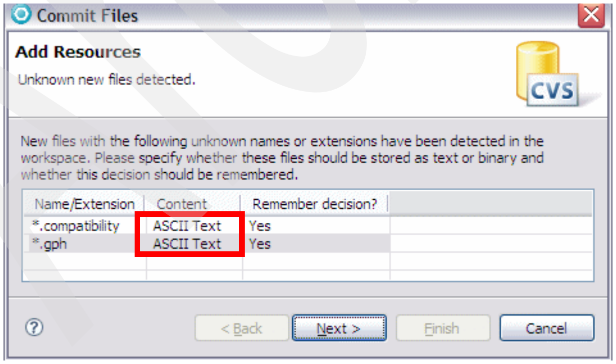
```
ITSOCVSGuide  [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
  src
    (default package)
    itso
    itso.rad7
    itso.rad7.cvs
    itso.rad7.cvs.servlet
      ServletA.java   (ASCII -kkv)
  Settings
  WebContent
  .classpath   (ASCII -kkv)
  .compatibility   (Binary)
  .project   (ASCII -kkv)
  WebDiagram.gph   (Binary)
```

☑ Launch the Commit wizard

Figure 27-16   Verification of resources add under CVS revision control

- A dialog opens, informing you that new file types are being added, which are not configured as types in the Application Developer preferences (see Figure 27-17). These are `*.compatibility` (used for allowing previous versions of Application Developer use the V7 project files) and `*.gph` (used to store the Web Diagram files).

  Set both these file types to ASCII Text, and click **Next.**



Add Resources

Unknown new files detected.

New files with the following unknown names or extensions have been detected in the workspace. Please specify whether these files should be stored as text or binary and whether this decision should be remembered.

Name/Extension	Content	Remember decision?
*.compatibility	ASCII Text	Yes
*.gph	ASCII Text	Yes

Figure 27-17   Adding new file types on check-in

- In the commit files dialog, enter `Initial version` in the comment field and click **Finish**.
- A status window opens, showing the progress as the initial versions of a new project are checked into the repository. Because the `ITSOCVSGuide` is a relatively small project, this process should only take a few seconds. For larger projects where the initial check-in might take some time, the **Run Background** button allows the user to continue working in their workspace while the CVS check-in process completes.

After this has been completed, the `ITSOCVSGuide` project is checked into the repository and available for other developers to use.

Adding a shared project to the workspace (step 2 - cvsuser2)

The purpose of any source code repository is to allow multiple developers to work as a team on the same project. The `ITSOCVSGuide` project has been created in one developer's workspace and shared using CVS. Now we want to add the same project to a second developer's workspace.

- The second developer must add the CVS repository location to their workspace using the CVS Repositories view in the CVS Repository Exploring perspective, as described in "Add a CVS repository" on page 1237.

  The difference is now that the HEAD branch in the repository, if expanded, contains the `ITSOCVSGuide` project (Figure 27-18).

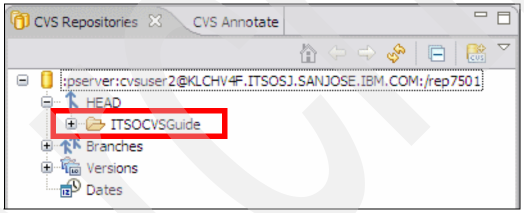


Figure 27-18 CVS Repository with ITSOCVSGuide project

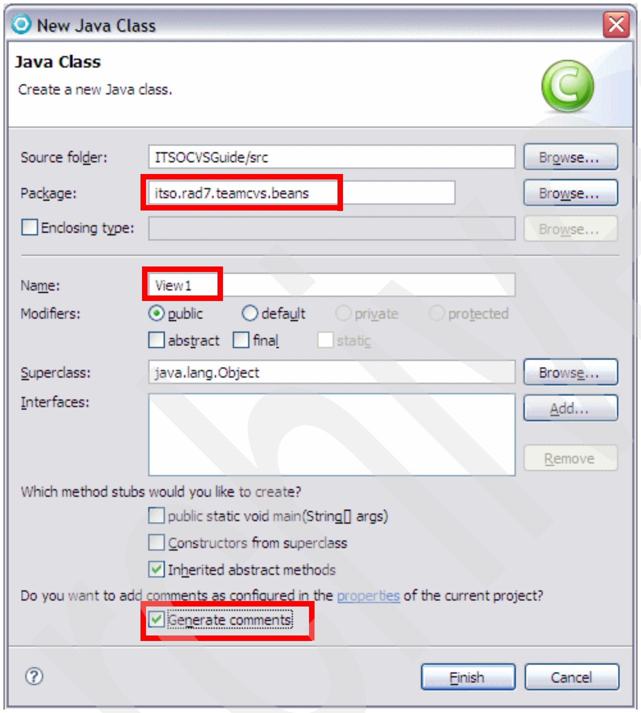
- Select the ITSOCVSGuide module, right-click, and select **Check Out**. The current project in the HEAD branch is checked out to the workspace.

Develop the view bean

Now that both developers have exactly the same synchronized HEAD branch of the `ITSOCVSGuide` project on their workspaces, it is time for the second developer to create the view bean `View1`.

- Open the Web perspective.

- Expand **ITSOCVSGuide** → **JavaSource: src** To add a new package right-click the **src** folder and select **New** → **Package** and enter **itso.rad7.teamcvs.beans** as the name. Click **Finish**.
- Right-click the new package and select **New** → **Class,** enter the name of the class to be **`View1`**.
- Make sure the **Generate Comments** option is selected (Figure 27-19).
- Click **Finish.**



New Java Class

Java Class

Create a new Java class.

Source folder: ITSOCVSGuide/src

Package: itso.rad7.teamcvs.beans

□ Enclosing type:

Name: View1

Modifiers: ● public ○ default ○ private ○ protected

□ abstract □ final □ static

Superclass: java.lang.Object

Interfaces:

Which method stubs would you like to create?

□ public static void main(String[] args)

□ Constructors from superclass

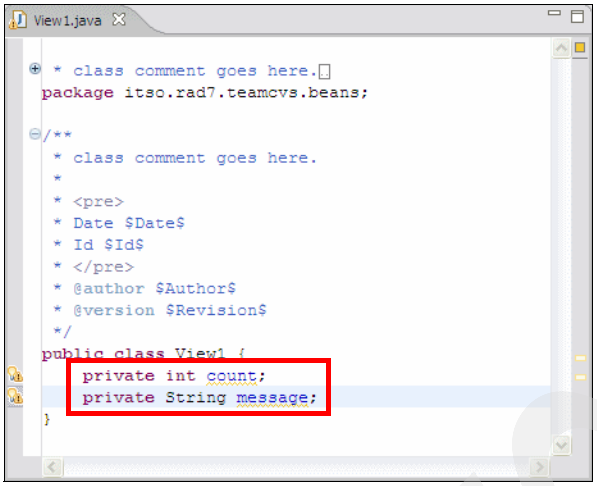
☑ Inherited abstract methods

Do you want to add comments as configured in the properties of the current project?

☑ Generate comments

Figure 27-19 Creating the View1 view bean

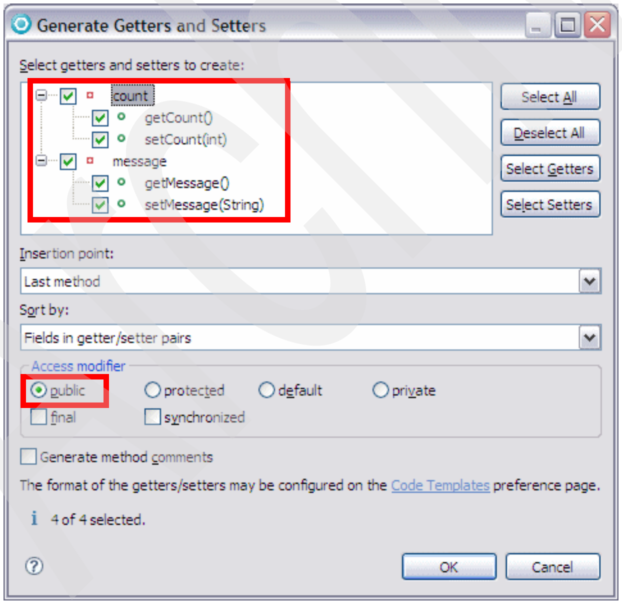
- Open the Java source code for `View1.java` and create two private attributes in the class an integer named `count` and a String named `message` (Figure 27-20).



```
 * class comment goes here.
package itso.rad7.teamcvs.beans;

/**
 * class comment goes here.
 *
 * <pre>
 * Date $Date$
 * Id $Id$
 * </pre>
 * @author $Author$
 * @version $Revision$
 */
public class View1 {
    private int count;
    private String message;
}
```

Figure 27-20 View1 code with new attributes

- Double-click the **count** attribute to highlight it, right-click, select **Source** → **Generate Getters and Setters,** select the **message fields** as well, and verify that the Access Modifier section has **public** set (Figure 27-21). Click **OK**.



Generate Getters and Setters

Select getters and setters to create:

☑ count
  ☑ getCount()
  ☑ setCount(int)
☑ message
  ☑ getMessage()
  ☑ setMessage(String)

Insertion point: Last method

Sort by: Fields in getter/setter pairs

Access modifier

● public ○ protected ○ default ○ private

□ final □ synchronized

□ Generate method comments

The format of the getters/setters may be configured on the Code Templates preference page.

4 of 4 selected.

Figure 27-21 Creating setters and getters for class View1

- Save the Java file.

Tip: In the Project Explorer view, the greater than sign (>) in front of a resource name means that the particular resource is not synchronized with the repository. The question mark symbol (?) indicates that the file is not in the repository. These visual cues can be used to determine when a project requires synchronization.

Synchronizing with the repository

To update the repository with these changes, do these steps:

- Right-click the **ITSOCVSGuide** project and select **Team** → **Synchronize with Repository**.
- A dialog prompts you to change to the Synchronize view. Click **Yes**. The project is compared with the repository, and the differences are displayed in the Synchronize view (Figure 27-22).

Note: While using Application Developer V7.0.0.2, we found that the sometimes the **.compatibility** and **.classpath** files were marked as changed, although no changes were directly applied to these resources. For the example, these files can either be ignored, checked back into the repository each time, or added to the list of CVS ignored resources.

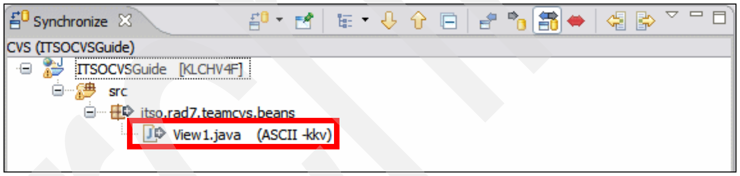
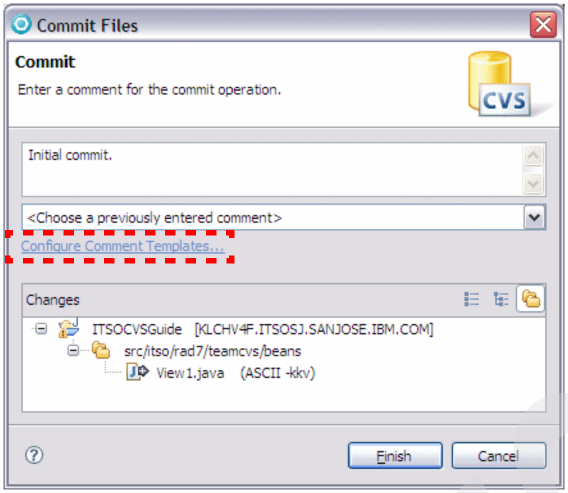


Figure 27-22 Synchronizing ITSOCVSGuide after creating the viewbean View1

This view allows you to update resources in the Workbench with newer content from the repository, commit resources from the Workbench to the repository, and resolve conflicts that might occur in the process. The arrow icons with a plus sign ⇨ indicate that the files do not exist in the repository. Because the class `View1.java` is new and not yet checked in it shows the plus sign in Figure 27-22.

- Add these new resources to version control by right-clicking **ITSOCVSGuide** in the Synchronize view, and selecting **Commit**.
- In the Commit Files dialog, enter `Initial Commit` for the commit comment and check the files displayed on bottom half of the window to make sure the changes are as expected (Figure 27-23).



Commit Files

Commit

Enter a comment for the commit operation.

Initial commit.

<Choose a previously entered comment>

Configure Comment Templates...

Changes

ITSOCVSGuide [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]

src/itso/rad7/teamcvs/beans

View1.java (ASCII -kkv)

Figure 27-23 Verifying committing of resources into repository

> **Note:** A new feature with Application Developer V7.0 is **Commit Comment Templates**, where the user can specify some common text within commit comments, and re-use the same format for all commits. To create such comments, click **Commit Comment Template** to create a comment, then select it from the drop-down list in this dialog.

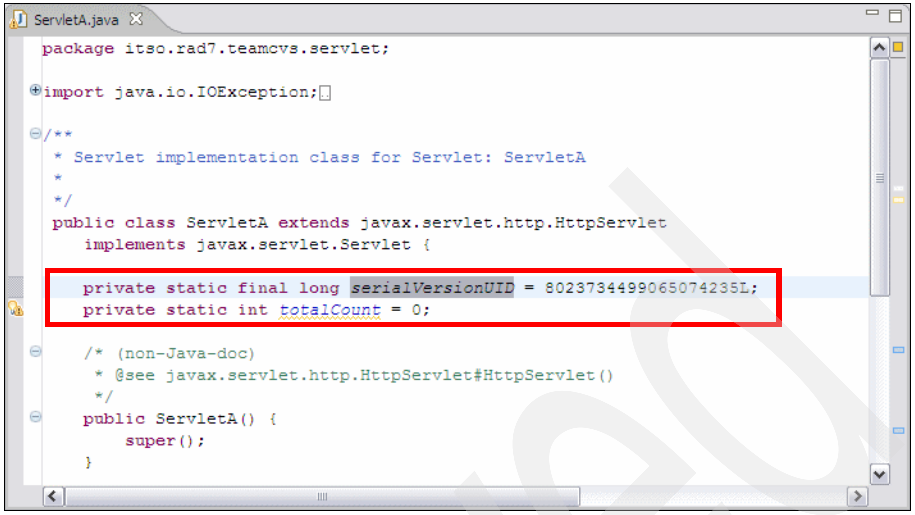
- Click **Finish** and the changes are committed to the repository.

Modifying the servlet (step 2 - cvsuser1)

While activities in “Adding a shared project to the workspace (step 2 - cvsuser2)” occur, our original user, cvsuser1, is working on developing the servlet further.

In the first workspace created for cvsuser1, do these steps:

- In the Web perspective expand **ITSOCVSGUIDE** → **Java Resources: src** → **itso.rad7.teamcvs.servlet** and open **ServletA.java** in the editor.
- Create a static attribute called `totalCount` of type int and initialized to zero, as in Figure 27-24.
- Also perform a **Quick Fix** to generate a **serialVersionUID**.



```
package itso.rad7.teamcvs.servlet;

import java.io.IOException;

/**
 * Servlet implementation class for Servlet: ServletA
 *
 */
public class ServletA extends javax.servlet.http.HttpServlet
   implements javax.servlet.Servlet {

   private static final long serialVersionUID = 8023734499065074235L;
   private static int totalCount = 0;

   /* (non-Java-doc)
    * @see javax.servlet.http.HttpServlet#HttpServlet()
    */
   public ServletA() {
       super();
   }
```

Figure 27-24 Addition of servlet attributes

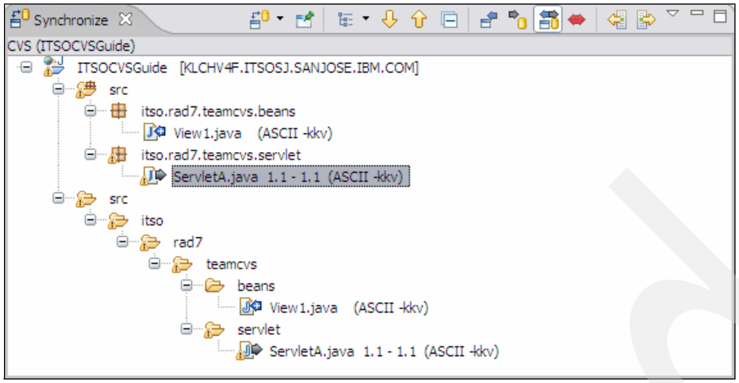
> **Note:** Files that are not saved have an asterisk (*) in front of their names in the title bar of the window. This assists in identifying resources that have to be saved.

- Save and close the file.

Synchronizing with the repository (step 3 - cvsuser1)

The first user now synchronizes with the repository and receives the changes of `cvsuser2` (adding the `View1` bean) and provides the opportunity to check-in the changes made to `Servlet1`. Do these steps:

- In the Project Explorer, right-click the **ITSOCVSGuide** project and select **Team** → **Synchronize with Repository**.
- Click **Yes** to switch to the Synchronize view.
- Expand the **ITSOCVSGuide** → **JavaSource** tree to view the changes (Figure 27-25).



```
CVS (ITSOCVSGuide)
ITSOCVSGuide   [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
    src
        itso.rad7.teamcvs.beans
            View1.java   (ASCII -kkv)
        itso.rad7.teamcvs.servlet
            ServletA.java  1.1 - 1.1  (ASCII -kkv)
    src
        itso
            rad7
                teamcvs
                    beans
                        View1.java   (ASCII -kkv)
                    servlet
                        ServletA.java  1.1 - 1.1  (ASCII -kkv)
```

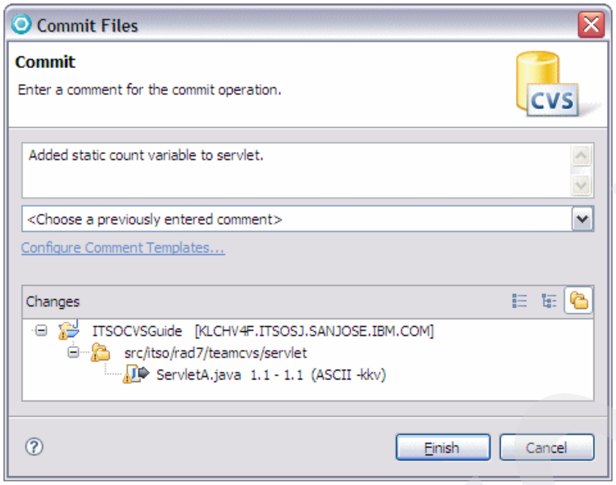
Figure 27-25 User cvsuser1 merging with CVS repository

> **Note:** The symbol → in the diagram indicates that an existing resource differs from what is in the repository. The symbol ← indicates that a new resource is in the repository that does not exist on the local workspace.

- To obtain updated resources from the CVS repository, right-click the project and select **Update**. This brings a copy of the `View1.java` file into this workspace.

> **Note:** When using Application Developer V7.0.0.2 sometimes a Java file that has been changed appears twice in the Synchronize view. It might appear under the representation of the Java package and also under a folder representation of the packages. Figure 27-25 shows an example of this with `View1.java` and `ServletA.java` both showing twice. While this looks untidy, if either of the representations of the file are updated, then both disappear from the Synchronize view as expected.

- Verify that the changes do not cause problems with existing resources in the local workspace, by checking the Problems View. In this case, there are none. Right-click the `ITSOCVSGuide` project and select **Commit**.
- In the Commit dialog, add the comment `Added static count variable to servlet.` and click **OK** (Figure 27-26). This checks the changes made to `Servlet1` into the repository.



Enter a comment for the commit operation.

Added static count variable to servlet.

<Choose a previously entered comment>

Configure Comment Templates...

```
ITSOCVSGuide  [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
  src/itso/rad7/teamcvs/servlet
    ServletA.java  1.1 - 1.1  (ASCII -kkv)
```

Figure 27-26 Adding comment for changes to Servlet1

The repository now has the latest changes to the code from both developers. The user cvsuser1 is in sync with the repository; however, cvsuser2,has not yet received the changes to the servlet.

Synchronizing with the repository (step 3 - cvsuser2)

The second workspace used by cvsuser2 should also synchronize with the repository and receive the changes of `cvsuser1`.

This brings the changes made to `ServletA` into the workspace and makes sure that both workspaces are completely up to date.

Parallel development (step 4 - cvsuser1 and cvsuser2)

The previous steps have highlighted development and repository synchronization with two people working on two parts of a project. It highlights the need to synchronize between each phase in the development before further work is performed.

The following scenario demonstrates two developers working simultaneously on the same file, starting from the same revision. Each user's sequence of events is described in the sections below; and a summary time-line is shown in the time-line in Figure 27-27.

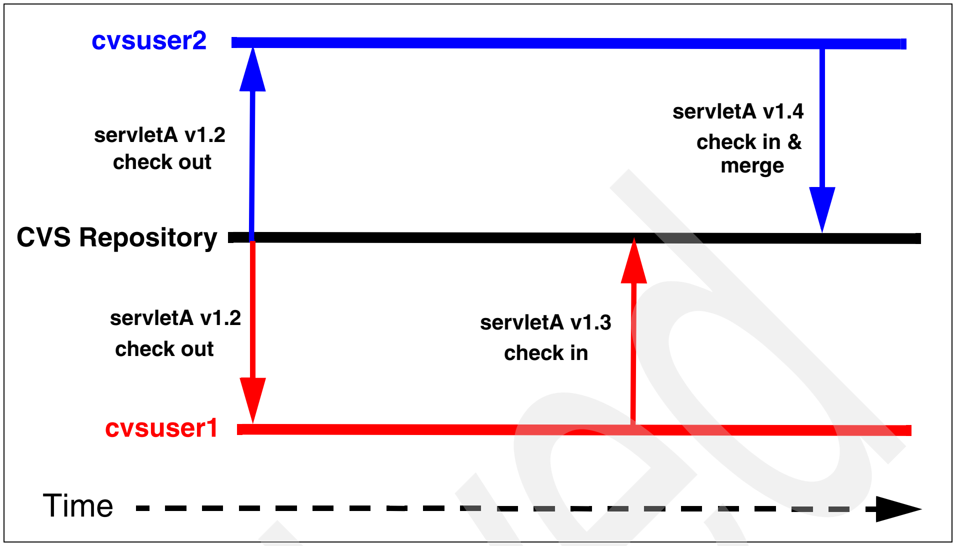


Figure 27-27 Parallel concurrent development of same resource by multiple developers

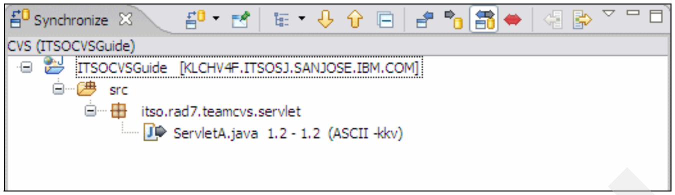
User cvsuser1 updates and commits changes

In this scenario, user cvsuser1 modifies the doPost method to log information for an attribute. The following procedure demonstrates how to synchronize the source code and commit the changes to CVS.

- In the Project Explorer, expand **ITSOCVSGuide** → **Java Resources: src** → **itso.rad7.teamcvs.servlet** and open **ServletA**.
- Navigate to the doPost method by scrolling down the file and adding the code to count the number of post requests received:

```
protected void doPost(HttpServletRequest request,
                          HttpServletResponse response)
        throws ServletException, IOException {
    // TODO Auto-generated method stub
    totalCount = totalCount + 1;
    System.out.println("The total number of requests is:" + totalCount);
}
```

- Save and close the file.
- Synchronize the project with the repository by right-clicking and selecting **Team** → **Synchronize with Repository**, and in the **Confirm Open Perspective** dialog, click **Yes** to open the **Team Synchronizing** perspective.
- Fully expand out the tree in the Synchronize view. The servlet should be the only change (Figure 27-28).



CVS (ITSOCVSGuide)
ITSOCVSGuide [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
src
itso.rad7.teamcvs.servlet
ServletA.java 1.2 - 1.2 (ASCII -kkv)

Figure 27-28 Changes in servlet from repository

- Right-click and select **Commit**, add the comment `User1 implementing the doPost method`, and click **Finish** to commit.

The developer cvsuser1 has now completed the task of adding code into the servlet. Changes can now be picked up by other developers in the team.

User cvsuser2 updates and commits changes

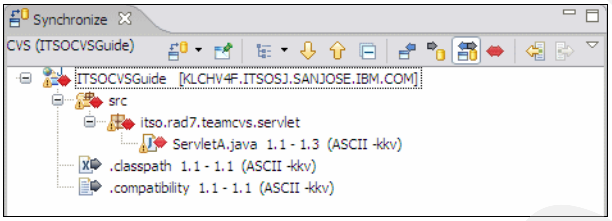
To complete the scenario, the second developer also makes changes to `doPost` method of `ServletA`. This is done in the workspace of cvsuser2 and assumes that the workspace was synchronized before this scenario was started.

To make the changes, do these steps:

- In the Project Explorer, expand **ITSOCVSGuide** → **Java Resources: src** → **itso.teamcvs.servlet** and open **ServletA**.
- Add a declaration for an instance variable for the view bean (`View1`). Navigate to the `doPost` method and add some logic. Save and close the file.

```
//add after the class declaration
private View1 myViewBean;

protected void doPost(HttpServletRequest request, HttpServletResponse
                                    response) throws ServletException, IOException {
   // TODO Auto-generated method stub
   myViewBean = new View1();
   myViewBean.setCount(totalCount);
   totalCount = totalCount + 1;
   System.out.println("The total number of requests is:"+totalCount);
   if (totalCount == 0){
      System.out.println("No hits on page");
   } else if (totalCount == 1){
      System.out.println("One hit on page");
   }
   if (totalCount > 1){
         System.out.println("Hits are greater than one");
   }
}
```

- Synchronize with the repository by right- clicking the **ITSOCVSGuide** project and selecting **Team** → **Synchronize with Repository**, and in the **Confirm Open Perspective** dialog, click **Yes** to open the **Team Synchronizing** perspective.
- Expand the tree in the Synchronize view to see the changes (Figure 27-29).



ITSOCVSGuide [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
src
itso.rad7.teamcvs.servlet
ServletA.java 1.1 - 1.3 (ASCII -kkv)
.classpath 1.1 - 1.1 (ASCII -kkv)
.compatibility 1.1 - 1.1 (ASCII -kkv)

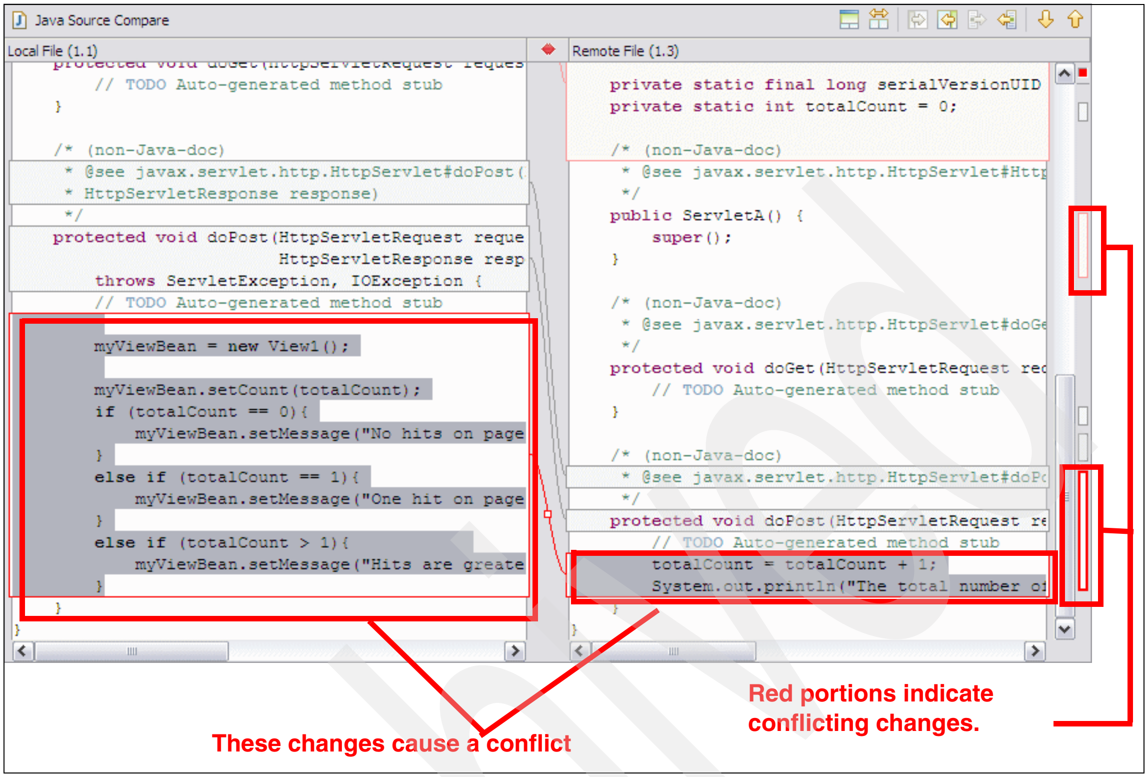
Figure 27-29 Synchronize view with conflicting changes

Note 1: The symbol indicates that the file has conflicting changes that requires merging.

Note 2: The `.classpath` and `.compatbility` files might still show as having changes, even though no direct actions have been taken to change these files. The files can be committed or ignored.

- Double-click the file **ServletA.java** to see the changes (Figure 27-30).
  - On the left side are the changes made by the current user cvsuser2. and on the right side is the code in the repository (checked in by user `cvsuser1`).
  - Use the arrow icons at the top to move from change to change.
  - Black lines between the panes indicate identical blocks.
  - Red lines indicate changes (inserts or conflicts).
  - Red bars on the right indicate conflicts.

  Merging in this case requires consolidation between the two developers as to the best solution. In our example, we assume that the changes in the repository (right side) have to be placed sequentially before changes performed by `cvsuser2` (left side).



Java Source Compare
Local File (1.1)
Remote File (1.3)
These changes cause a conflict
Red portions indicate conflicting changes.

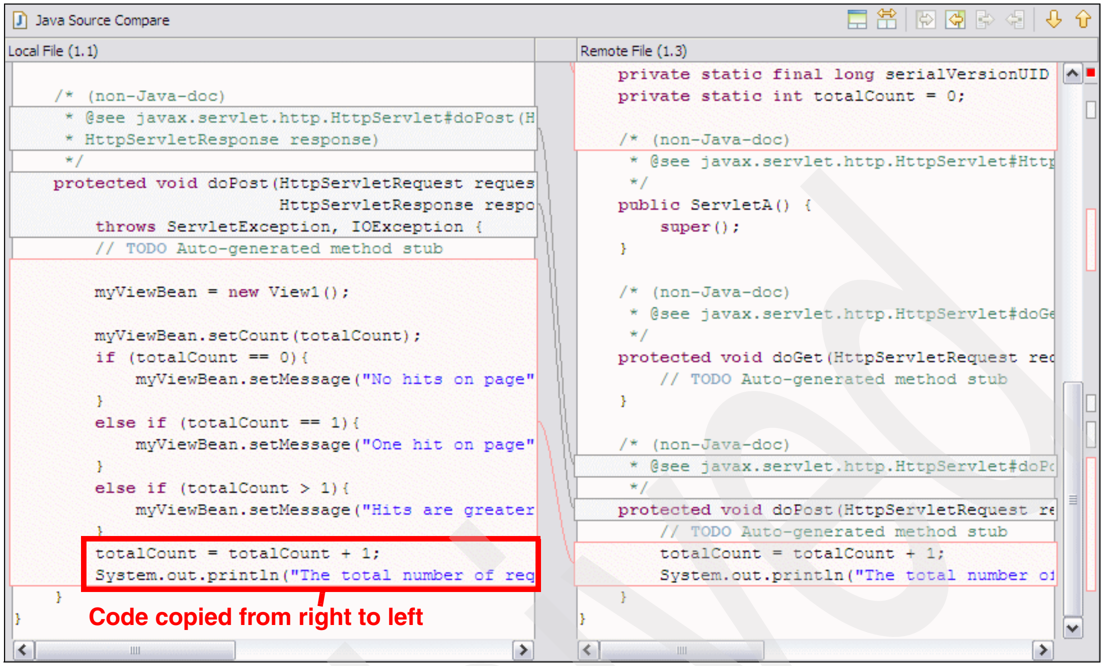
```
    protected void doGet(HttpServletRequest reques
        // TODO Auto-generated method stub
    }

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#doPost(
     * HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest reque
                          HttpServletResponse resp
        throws ServletException, IOException {
        // TODO Auto-generated method stub

        myViewBean = new View1();

        myViewBean.setCount(totalCount);
        if (totalCount == 0){
            myViewBean.setMessage("No hits on page
        }
        else if (totalCount == 1){
            myViewBean.setMessage("One hit on page
        }
        else if (totalCount > 1){
            myViewBean.setMessage("Hits are greate
        }
    }
}
```

```
    private static final long serialVersionUID
    private static int totalCount = 0;

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#Http
     */
    public ServletA() {
        super();
    }

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#doGe
     */
    protected void doGet(HttpServletRequest re
        // TODO Auto-generated method stub
    }

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#doPo
     */
    protected void doPost(HttpServletRequest re
        // TODO Auto-generated method stub
        totalCount = totalCount + 1;
        System.out.println("The total number of
    }
}
```

Figure 27-30 The changes between the local and remote repository

- Navigate to the change in the `doPost` method.
- Click the **Copy Current Change from Right to Left** icon ⇦.

  This places the change of the conflicting section on the right-hand panel to the bottom of the section in the left-hand panel (Figure 27-31).



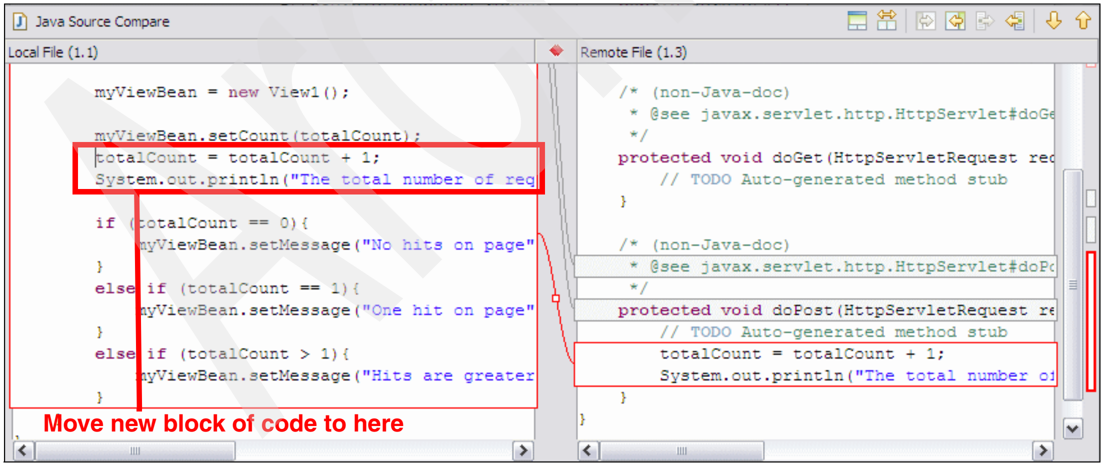
```
    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#doPost(H
     * HttpServletResponse response)
     */
    protected void doPost(HttpServletRequest reques
                          HttpServletResponse respo
        throws ServletException, IOException {
        // TODO Auto-generated method stub

        myViewBean = new View1();

        myViewBean.setCount(totalCount);
        if (totalCount == 0){
            myViewBean.setMessage("No hits on page"
        }
        else if (totalCount == 1){
            myViewBean.setMessage("One hit on page"
        }
        else if (totalCount > 1){
            myViewBean.setMessage("Hits are greater
        }
        totalCount = totalCount + 1;
        System.out.println("The total number of req
    }
}
```

```
    private static final long serialVersionUID
    private static int totalCount = 0;

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#Http
     */
    public ServletA() {
        super();
    }

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#doGe
     */
    protected void doGet(HttpServletRequest rec
        // TODO Auto-generated method stub
    }

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#doPo
     */
    protected void doPost(HttpServletRequest re
        // TODO Auto-generated method stub
        totalCount = totalCount + 1;
        System.out.println("The total number o
    }
}
```

Code copied from right to left

Figure 27-31 Merging changes from right to left

- In the left pane, highlight the two lines of code which were added and move them to the correct location in the method (Figure 27-32).



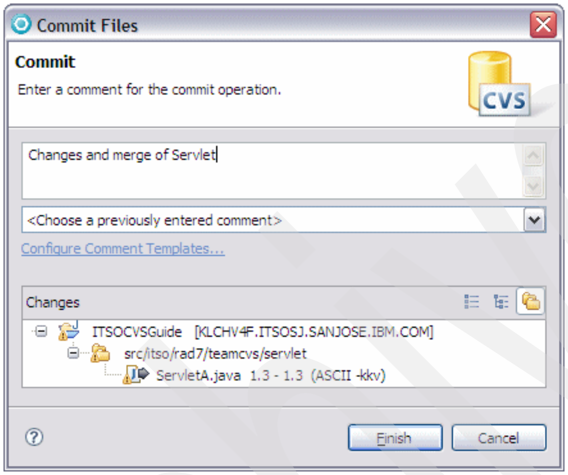
```
        myViewBean = new View1();

        myViewBean.setCount(totalCount);
        totalCount = totalCount + 1;
        System.out.println("The total number of req

        if (totalCount == 0){
            myViewBean.setMessage("No hits on page"
        }
        else if (totalCount == 1){
            myViewBean.setMessage("One hit on page"
        }
        else if (totalCount > 1){
            myViewBean.setMessage("Hits are greater
        }
```

```
    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#doGe
     */
    protected void doGet(HttpServletRequest rec
        // TODO Auto-generated method stub
    }

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#doPo
     */
    protected void doPost(HttpServletRequest re
        // TODO Auto-generated method stub
        totalCount = totalCount + 1;
        System.out.println("The total number o
    }
}
```

Move new block of code to here

Figure 27-32 Move the added code to the correct merge point

- Verify that the code is exactly as agreed by the developers, and save the new merged change by selecting **File** → **Save**.
- Re-synchronize the file using **Team** → **Synchronize With Repository**.
- In the Synchronize view, verify that the changes are correct, right-click `ServletA.java` and select **Mark as Merged**, then right-click again and select **Commit**.
- In the Commit dialog, enter the comment `Changes and merge of Servlet` (Figure 27-33).



Commit Files

Commit

Enter a comment for the commit operation.

Changes and merge of Servlet

<Choose a previously entered comment>

Configure Comment Templates...

Changes

ITSOCVSGuide [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
src/itso/rad7/teamcvs/servlet
ServletA.java 1.3 - 1.3 (ASCII -kkv)

Figure 27-33 Comment for merged changes

This operation creates a revision of the file, revision 1.4, which contains the merged changes from users cvsuser1 and cvsuser2. This is the case even though both developers originally checked out revision 1.2.

User cvsuser1 synchronizes

The workspace for cvsuser1 should also be synchronized with the repository at this stage to pick up the merged code of the servlet.

Creating a version (step 5 - cvsuser1)

Now that the changes for both users are committed and `cvsuser1` has synchronized with the repository, we want to create a version to milestone our work. Perform the following in the workspace of cvsuser1:

- Select the **ITSOCVSGuide** project in the Project Explorer and select **Team** → **Tag as Version**. The Tag Resources dialog opens (Figure 27-34).

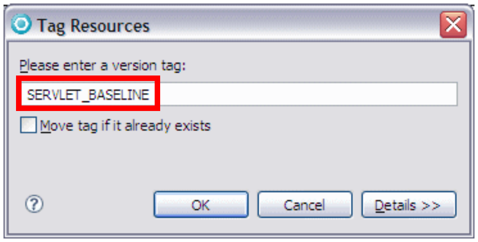


Figure 27-34 Tagging the project as a version

- Enter **SERVLET_BASELINE** as the version tag and click **OK**.
- Verify that the tag has been performed by switching to the CVS Repository Exploring perspective and expand the repository (Figure 27-35).

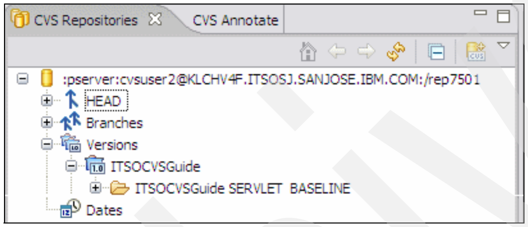

CVS Repositories
CVS Annotate
:pserver:cvsuser2@KLCHV4F.ITSOSJ.SANJOSE.IBM.COM:/rep7501
HEAD
Branches
Versions
ITSOCVSGuide
ITSOCVSGuide SERVLET_BASELINE
Dates

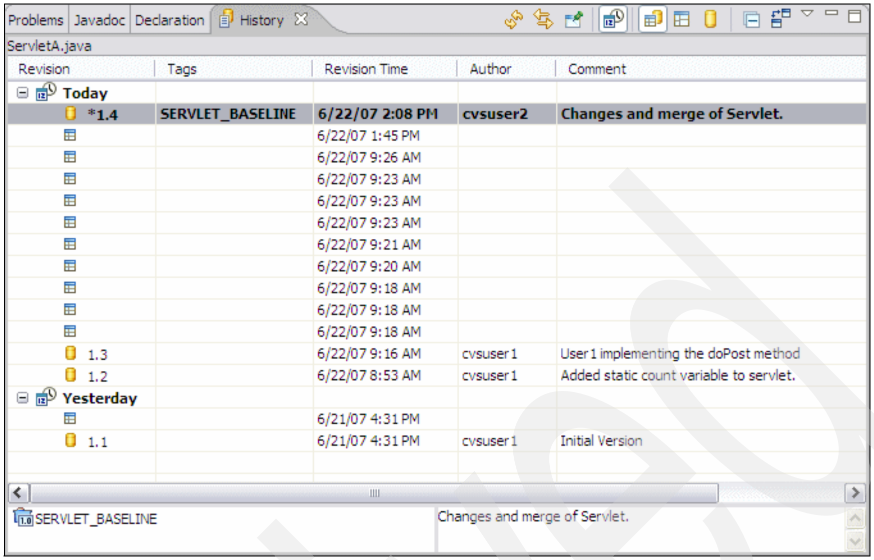
Figure 27-35 Repository view showing new project revision

CVS resource history

Within Application Developer, a developer can view the resource history of any file in a shared project. This is done in the CVS resource History view, which shows a list of all the revisions of a resource in the repository. From this view you can also compare two revisions, revert the existing workspace file to a previous revision, or open an editor to show the contents of a revision.

To demonstrate this feature, do these steps:

- In the Project Explorer, expand **ITSOCVSGuide** → **Java Resources: src** → **itso.rad7.teamcvs.servlet**, right-click `ServletA.java` and select **Team** → **Show History**, and the History view opens (Figure 27-36).



ServletA.java

Revision	Tags	Revision Time	Author	Comment
Today				
***1.4**	**SERVLET_BASELINE**	**6/22/07 2:08 PM**	**cvsuser2**	**Changes and merge of Servlet.**
		6/22/07 1:45 PM		
		6/22/07 9:26 AM		
		6/22/07 9:23 AM		
		6/22/07 9:23 AM		
		6/22/07 9:23 AM		
		6/22/07 9:21 AM		
		6/22/07 9:20 AM		
		6/22/07 9:18 AM		
		6/22/07 9:18 AM		
		6/22/07 9:18 AM		
1.3		6/22/07 9:16 AM	cvsuser1	User1 implementing the doPost method
1.2		6/22/07 8:53 AM	cvsuser1	Added static count variable to servlet.
Yesterday				
		6/21/07 4:31 PM		
1.1		6/21/07 4:31 PM	cvsuser1	Initial Version

SERVLET_BASELINE

Changes and merge of Servlet.

Figure 27-36 CVS History view for ServletA.java

The CVS resource history displays the columns described in Table 27-4.

Table 27-4 CVS resource history terminology

Column	Description
Revision	The revision number of each version of the file in the repository. An asterisk (*) indicates that this is the current version in the workspace.
Tags	Any tags which have been associated with the revision.
Revision Time	The date and time when the revision was created in the repository.
Author	The name of the used that created and checked in the revision into the repository.
Comments	The comment (if any) supplied for this revision at the time it was committed.

The following icons are available at the top of the History view:

- **Refresh**—Refreshes the history shown for the currently shown resource.
- **Link Editor with Selection**—A toggle switch which automatically shows the history of the resource currently being shown in the main editor.
- **Pin the History View**—Locks the history view into showing just the currently selected resource’s history. When this is toggled on then the **Link Editor with Selection** is automatically switched off.

- **Group Revisions by date**—This changes the view to show the revisions ordered by date rather than by logical revision number. It is also possible to order the items in the History View by clicking on the column headers.
- **Local Revisions, Local and Remote Revisions, Local Revisions**—These boxes provide three options, which type of revisions to show for the selected resource.
- **Compare Mode**—If this toggle is on then double-clicking on a line in the history view shows a comparison between the selected repository file and the file in the workspace. If this is switched off then double- clicking displays the contents of that file revision.
- **Filters**—This feature is available from the drop-down menu in the History view. It allows a user to filter the History view by author, date, or text within the check-in comments.

Comparisons in CVS

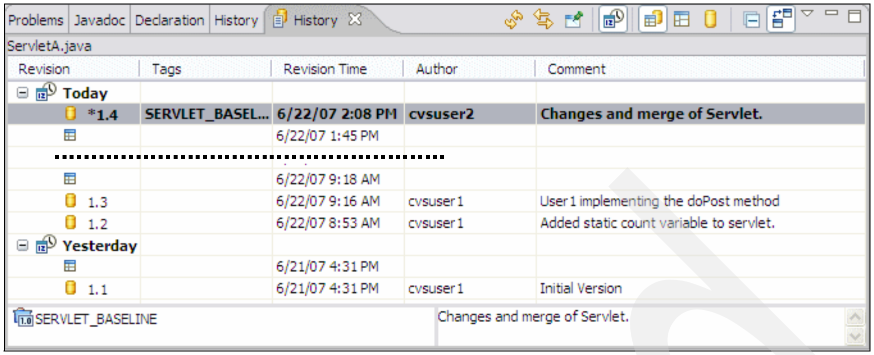
Often developers have to view what changes have been made to a file and in which revision. Application Developer provides a mechanism to graphically display two revisions of a file and their differences. Two types of comparison are possible, users can compare the version in their workspace with any version in the CVS repository, or any two files in the CVS repository can be compared with each other.

The Resource History view provides these mechanisms and the following scenario has an example of how to do this.

Comparing a workspace file with the repository

The user cvsuser1 has Version 1.4 of the ServletA file in the workspace and wants to compare the differences between their current version and 1.1. To do this, do these steps:

- In the Project Explorer, right-click `ServletA.java` and select **Compare with → History**, and a view similar to the History view opens (Figure 27-37).



ServletA.java

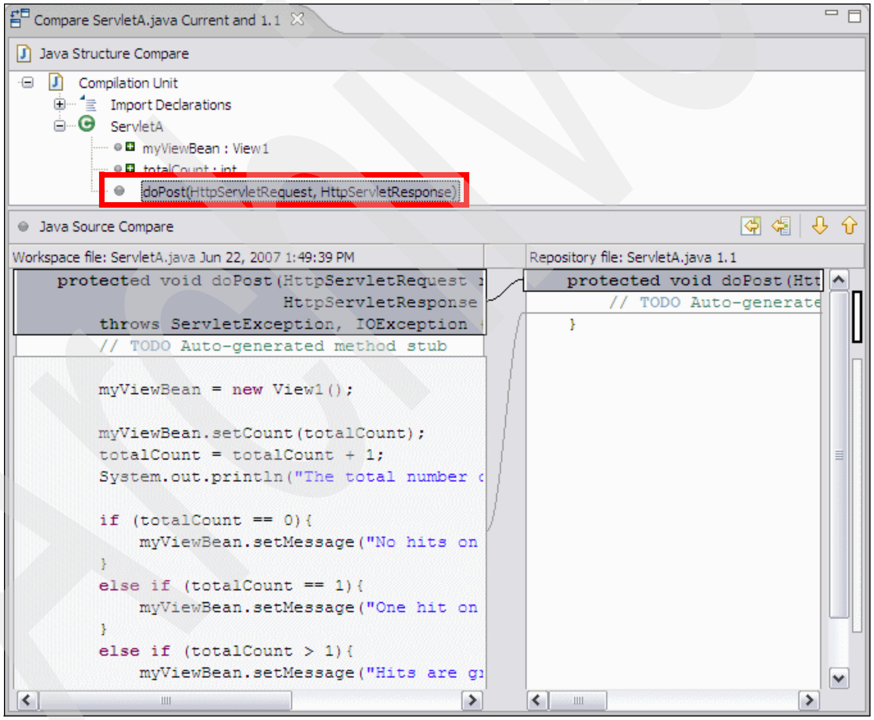
Revision	Tags	Revision Time	Author	Comment
Today				
*1.4	SERVLET_BASEL...	6/22/07 2:08 PM	cvsuser2	Changes and merge of Servlet.
		6/22/07 1:45 PM		
		6/22/07 9:18 AM		
1.3		6/22/07 9:16 AM	cvsuser1	User1 implementing the doPost method
1.2		6/22/07 8:53 AM	cvsuser1	Added static count variable to servlet.
Yesterday				
		6/21/07 4:31 PM		
1.1		6/21/07 4:31 PM	cvsuser1	Initial Version

SERVLET_BASELINE

Changes and merge of Servlet.

Figure 27-37   List of revisions for ServletA.java

- Double-click the revision 1.1 version and a comparison editor opens (Figure 27-38).



Workspace file: ServletA.java Jun 22, 2007 1:49:39 PM

```
    protected void doPost(HttpServletRequest r
                          HttpServletResponse
        throws ServletException, IOException {
        // TODO Auto-generated method stub

        myViewBean = new View1();

        myViewBean.setCount(totalCount);
        totalCount = totalCount + 1;
        System.out.println("The total number c

        if (totalCount == 0){
            myViewBean.setMessage("No hits on
        }
        else if (totalCount == 1){
            myViewBean.setMessage("One hit on
        }
        else if (totalCount > 1){
            myViewBean.setMessage("Hits are gi
```

Repository file: ServletA.java 1.1

```
    protected void doPost(Htt
        // TODO Auto-generate
    }
```

Figure 27-38   Comparison between ServletA.java and revision 1.1

- In the top half the outline view of the changes are shown. This includes attribute changes and which methods that have been changed.

- In the bottom two panes the actual code differences are highlighted. The left pane has the revision in the workspace and the right pane has the revision 1.1 from the repository.

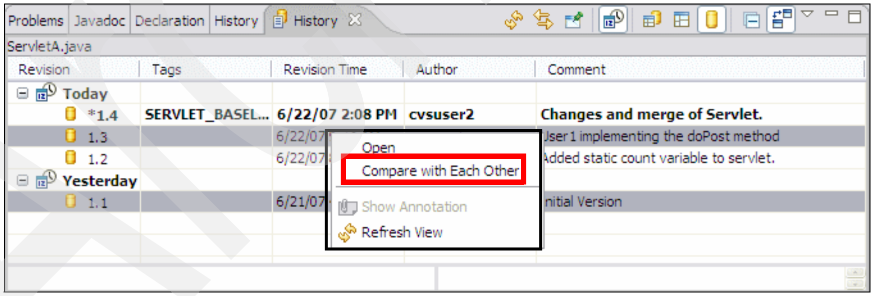
> **Note:** The bars in the bottom pane on the right-hand side indicate the parts of the file which are different. By clicking them Application Developer positions the panes to highlight the changes, This can assist in quickly moving around large files with many changes.

Comparing two revisions in repository

In this case, the developer wants to compare the differences between revision 1.1 and 1.3 in the repository of the ServletA file, but version 1.4 is in the workspace and the developer does not want to remove it.

The procedure to compare these two files is as follows:

- Open up the CVS Resource History using the procedure in "CVS resource history" on page 1254, which would display the view shown in Figure 27-39.
- First click the ▯ icon to only show remote revisions.
- Select the row of the first revision to compare, for example revision 1.1, and then, while pressing the Ctrl key, select the row of the second version, which is 1.3.
- Right-click, ensuring that the two revisions remain highlighted, and select **Compare With Each Other** (Figure 27-39).

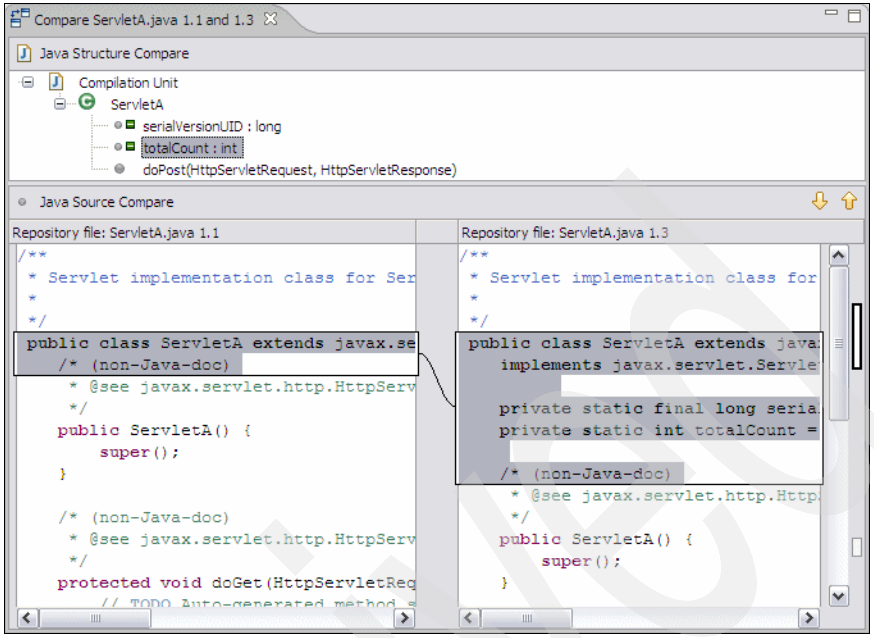


ServletA.java

Revision	Tags	Revision Time	Author	Comment
Today				
*1.4	SERVLET_BASEL...	6/22/07 2:08 PM	cvsuser2	Changes and merge of Servlet.
1.3		6/22/07		Jser1 implementing the doPost method
1.2		6/22/07		Added static count variable to servlet.
Yesterday				
1.1		6/21/07		nitial Version

Figure 27-39 Highlight the two versions to compare

- The result appears as in Figure 27-40. The higher version always appears in the left-hand pane and the lower version to the right.


Compare ServletA.java 1.1 and 1.3
Java Structure Compare
Compilation Unit
ServletA
serialVersionUID : long
totalCount : int
doPost(HttpServletRequest, HttpServletResponse)
Java Source Compare
Repository file: ServletA.java 1.1
Repository file: ServletA.java 1.3

```
/**
 * Servlet implementation class for Ser
 *
 */
public class ServletA extends javax.se
    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServ
     */
    public ServletA() {
        super();
    }

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServ
     */
    protected void doGet(HttpServletReq
        // TODO Auto-generated method s
```

```
/**
 * Servlet implementation class for
 *
 */
public class ServletA extends java
    implements javax.servlet.Servle

    private static final long seria
    private static int totalCount =

    /* (non-Java-doc)
     * @see javax.servlet.http.Http
     */
    public ServletA() {
        super();
    }
```

Figure 27-40 Comparisons of two revisions from the repository

Annotations in CVS

The Annotation view allows a user to view all the changes that have been performed on a particular file in a single combination of workspace views. It displays what lines were changed in particular revisions, the author responsible for the change, when the file was changed and the change description entered at the time. By showing this information across all revisions of a file and in the same set of connected views, developers can quickly determine the origin of changes and the explanation behind them.

To demonstrate annotations, we can go back to our example of looking at `ServletA` and see what the information the Annotations feature provides:

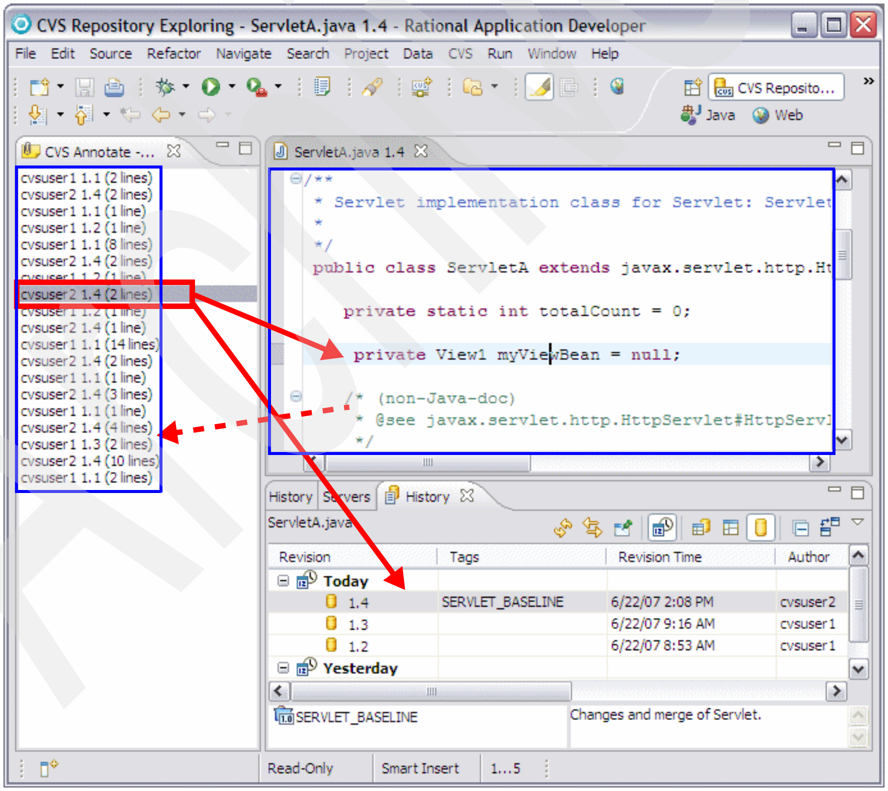
- In the Project Explorer, right-click `ServletA.java` and select **Team** → **Show Annotation**. This switches to the CVS Repository Exploring view and display the views shown in Figure 27-41.
- This prompts the user *Do you wish to view annotations using quick diff?*. This option allows the user to select between two possible versions of the Annotation view:
  - Answering **No** displays the full Annotation view.

- Answering **Yes** shows a more summarized version of the annotation details, which uses pop-ups and a color coded bar to show the changes.

Full Annotation view

Click **No** to displays the full Annotation view:

- A set of interrelated views are shown:
  - The CVS Annotation view is shown on the left-hand side. It displays a list of sections of the code and the CVS revision where that code segment was added or last edited. This is ordered on code sections from the top of the source code file to the bottom.
  - The source code is shown in the right-hand view.
  - The History view is shown at the bottom-right.
- Clicking on any line in the Annotation view highlights the associated lines in the source code and also the information from the revision in the History view. Click on a line of source code to show which CVS revision was changed.



```
cvsuser1 1.1 (2 lines)
cvsuser2 1.4 (2 lines)
cvsuser1 1.1 (1 line)
cvsuser1 1.2 (1 line)
cvsuser1 1.1 (8 lines)
cvsuser2 1.4 (2 lines)
cvsuser1 1.2 (1 line)
cvsuser2 1.4 (2 lines)
cvsuser1 1.2 (1 line)
cvsuser2 1.4 (1 line)
cvsuser1 1.1 (14 lines)
cvsuser2 1.4 (2 lines)
cvsuser1 1.1 (1 line)
cvsuser2 1.4 (3 lines)
cvsuser1 1.1 (1 line)
cvsuser2 1.4 (4 lines)
cvsuser1 1.3 (2 lines)
cvsuser2 1.4 (10 lines)
cvsuser1 1.1 (2 lines)
```

```
/**
 * Servlet implementation class for Servlet: Servlet
 *
 */
public class ServletA extends javax.servlet.http.Ht

   private static int totalCount = 0;

    private View1 myViewBean = null;

   /* (non-Java-doc)
    * @see javax.servlet.http.HttpServlet#HttpServl
    */
```

Revision	Tags	Revision Time	Author
Today			
1.4	SERVLET_BASELINE	6/22/07 2:08 PM	cvsuser2
1.3		6/22/07 9:16 AM	cvsuser1
1.2		6/22/07 8:53 AM	cvsuser1
Yesterday			

SERVLET_BASELINE

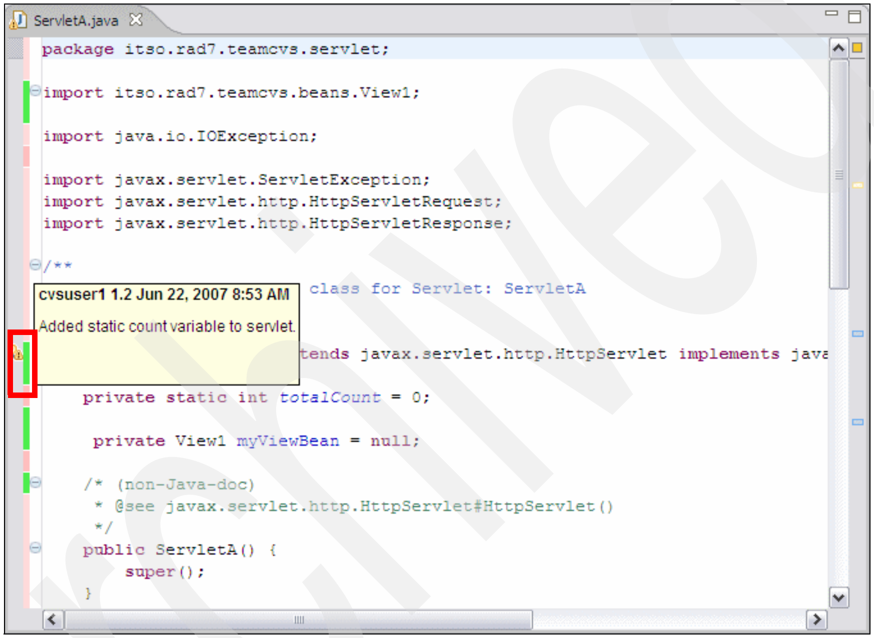
Changes and merge of Servlet.

Figure 27-41 Annotation view for ServletA.java

Summarized Annotation view

Repeat the first three steps and when prompted *Do you wish to view annotations using quick diff?*, answer **Yes**:

- This time Application Developer shows a different annotation view, where a colored line is displayed on the left bar of the source code. When you hover the mouse over the colored line, a a pop- up displays which CVS revision was last responsible for changing that line (Figure 27-42). Again, the CVS History view is shown in the bottom right-hand pane and shows the history for the revision associated with the line of source code selected.



```
package itso.rad7.teamcvs.servlet;

import itso.rad7.teamcvs.beans.View1;

import java.io.IOException;

import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

/**
                          class for Servlet: ServletA

                         tends javax.servlet.http.HttpServlet implements java
    private static int totalCount = 0;

     private View1 myViewBean = null;

    /* (non-Java-doc)
     * @see javax.servlet.http.HttpServlet#HttpServlet()
     */
    public ServletA() {
        super();
    }
```

cvsuser1 1.2 Jun 22, 2007 8:53 AM

Added static count variable to servlet.

Figure 27-42   View Annotations with Quick Diff

The annotation view allows a developer to identify changes that have occurred in a particular file and identify the root causes of issues that they might have.

Branches in CVS

Branches are a source control technique to allow development on more than one baseline in the repository.

In CVS, the HEAD branch always refers to the latest or current work that is being performed in a team environment. This is only sufficient for a development team that works on one release, which contains all the latest developments, including major enhancements and bug fixes. The real-world situation is usually that at least two streams are required. One main stream to manage the development, and a maintenance stream for the version that is in currently production. This allows new versions of the production build to be created without fear of being affected by the changes made to the main development stream. This scenario is when branches can be useful and where CVS baselines and parallel streams of work should be created.

At some point the development and maintenance streams have to be merged together to provide a new baseline to be a production version. This process ensures that any fixes or enhancements made in the maintenance stream make it into the development stream. This is known as a merge, and the CVS tools within Application Developer provide features to facilitate this process. A representation of this is shown in Figure 27-43.

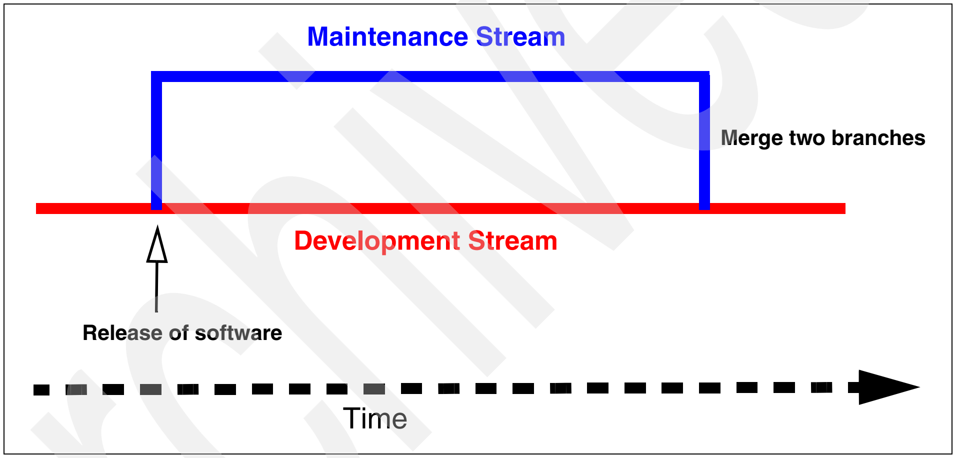


Figure 27-43 Branching with two streams

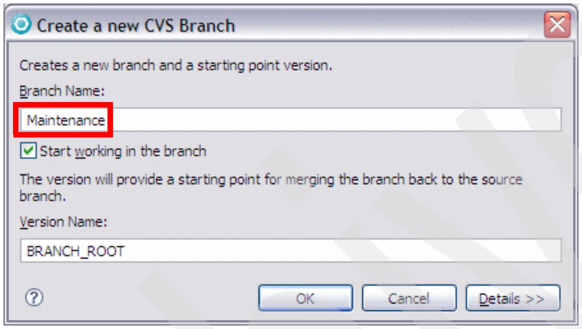
Branching

Creating a branch is useful when you want to maintain multiple streams of the software being developed or supported and when they are in different stages of delivery (usually development and production support).

The scenario demonstrated here is that a particular release has been deployed to the production environment, and a new project has started to enhance the application. In addition to this, the existing production release has to be maintained so that problems identified are fixed quickly. Branching provides the mechanism to achieve this and the following example outlines how this might be done.

Perform the following steps for the first workspace for cvsuser1:

- In the Project Explorer of the Web perspective, right-click **ITSOCVSGuide** and select **Team** → **Tag as Version**. Enter the tag name `BRANCH_ROOT`, and click **OK**.
- Right-click the project **ITSOCVSGuide**, right-click, and select **Team** → **Branch**.
- In the Create a new CVS Branch dialog (Figure 27-44), type the name of the branch (for example, `Maintenance`) and the branch to base it from (`BRANCH_ROOT`). Select **Start working on the branch** so that the workspace automatically sets itself up for development on the new branch. Click **OK**.



Create a new CVS Branch

Creates a new branch and a starting point version.

Branch Name:

Maintenance

☑ Start working in the branch

The version will provide a starting point for merging the branch back to the source branch.

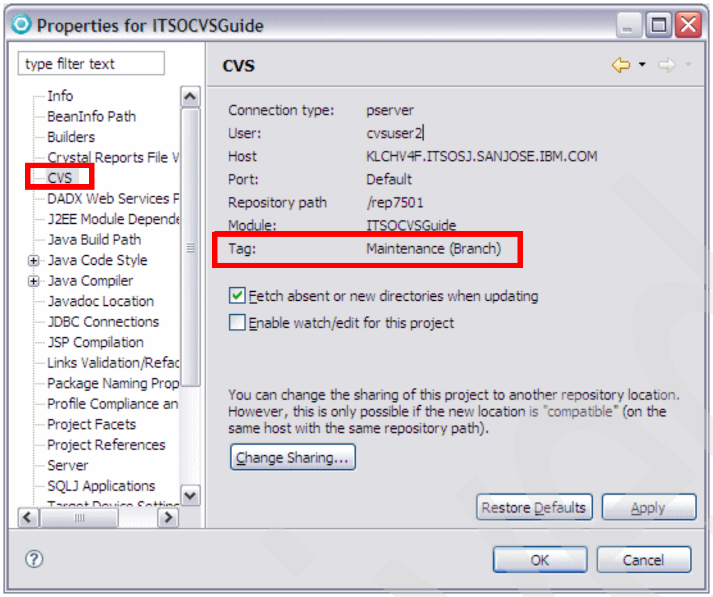
Version Name:

BRANCH_ROOT

Figure 27-44 Creating a new CVS branch

> **Attention:** Remember the version name entered here, as it is important. It identifies the point at which the branch was created and is required later when the branches are merged.

- Right-click the project **ITSOCVSGuide**, select **Properties**, and select the **CVS** tab. A view is displayed with the tag name displayed as `Maintenance (Branch)`. This indicates that the project is now associated with the CVS Maintenance branch and any changes checked in go to that branch (Figure 27-45).


Properties for ITSOCVSGuide

CVS

Connection type: pserver
User: cvsuser2
Host KLCHV4F.ITSOSJ.SANJOSE.IBM.COM
Port: Default
Repository path /rep7501
Module: ITSOCVSGuide
Tag: Maintenance (Branch)

☑ Fetch absent or new directories when updating
☐ Enable watch/edit for this project

You can change the sharing of this project to another repository location. However, this is only possible if the new location is "compatible" (on the same host with the same repository path).

Figure 27-45 Branch information for a project in the local workspace

- Open the CVS Repository Explorer window by clicking **Window** → **Open Perspective** → **Other** → **CVS Repository Explorer**.
- Right-click the repository and click **Refresh View**.
- Expand the tree to verify that the branch has been created in the repository (Figure 27-46).

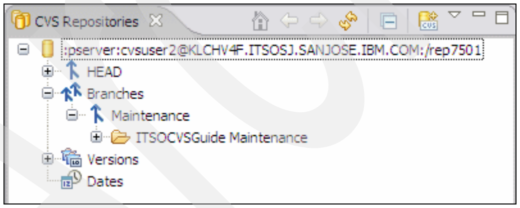


Figure 27-46 List of branches

Refreshing branching information

The CVS Repositories view does not automatically receive a list of all branches from the server. If a branch has been created on the CVS server, the user of a workspace must perform a refresh to receive the name of the new branch. In our sample case, `cvsuser2` should perform a refresh to receive information about the new maintenance branch.

From the cvsuser2 workspace, do these steps:

- To refresh the branches in a repository, open the CVS Repository Exploring perspective.
- Select the repository and expand the tree. Highlight the **Branches** node, right-click, and select **Refresh Branches**.
- In the Refresh Branches dialog, click **Select All** followed by **Finish**.
- The Maintenance branch is now shown under the Branches folder (Figure 27-47).

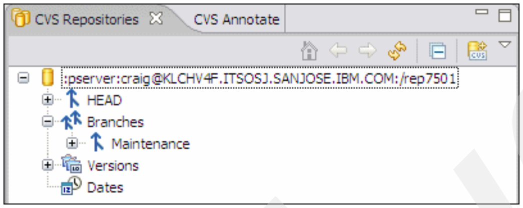


Figure 27-47 Refreshed Branch list

Updating branch code

Assume now that there are changes required to be made to ServletA and a new view bean (View2) must to be created. This scenario demonstrates the merge process with the changes being made in the maintenance branch and then moved into the main branch.

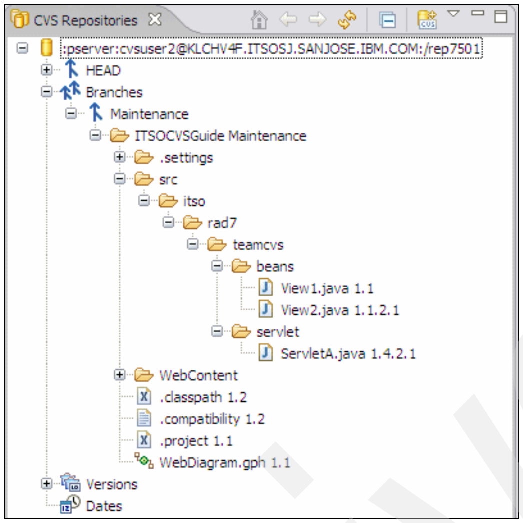
In the workspace of cvsuser1, do these steps:

- From the Project Explorer open **ServletA.java**.
- Navigate to the doPost method and at the top add the statement:

```
System.out.println("Added in some code to demonstrate branching");
```

- Save and close the file.
- Right-click the package **itso.rad7.teamcvs.beans** and select **New** → **Class.**
- Enter View2 for the name of the class and click **Finish**.
- Right-click the project **ITSOCVSGuide** and select **Team** → **Synchronize With Repository**. Click **Yes** to switch to the **Team Synchronizing** perspective.
- From the Synchronize view, select the **ITSOCVSGuide** project, right-click and select **Commit**.
- In the Commit dialog, enter Branching example as the revision comments, and click **Finish**.

- In the CVS Repository Explorer perspective, expand the tree below the Maintenance branch (Figure 27-48).



```
CVS Repositories
:pserver:cvsuser2@KLCHV4F.ITSOSJ.SANJOSE.IBM.COM:/rep7501
  HEAD
  Branches
    Maintenance
      ITSOCVSGuide Maintenance
        .settings
        src
          itso
            rad7
              teamcvs
                beans
                  View1.java 1.1
                  View2.java 1.1.2.1
                servlet
                  ServletA.java 1.4.2.1
        WebContent
        .classpath 1.2
        .compatibility 1.2
        .project 1.1
        WebDiagram.gph 1.1
  Versions
  Dates
```

Figure 27-48 Code checked into the branch

Note: Note that the logical revision for `View2.java` is 1.1.2.1 and for `ServletA.java` is 1.4.2.1. The extra two numbers in the logical revision are added by CVS when a branch is created. The first two numbers indicate the logical revision where the branch was created, the third indicates which branch the change from that logical revision (currently the second one if we count the HEAD branch) and the final number is the logical revision within this branch. In this case both files are the first revision in the Maintenance branch, and so the last digit is the number one.

The changes have now been committed into the Maintenance branch, which now has different content than the main branch. These changes are not seen by developers working on the HEAD branch, which is the development stream in our scenario.

Merging

Merging of branches occurs when it is decided that code from one branch should be incorporated into another branch. This might be required for several reasons, such as if a major integration release is about to be released for testing, or if bug fixes are required from the maintenance branch to resolve certain issues.

The scenario here is that development on the main CVS branch has completed and any production fixes made to the maintenance branch are required in the main branch as a new production build is planned.

To merge the two branches, the following information is required:

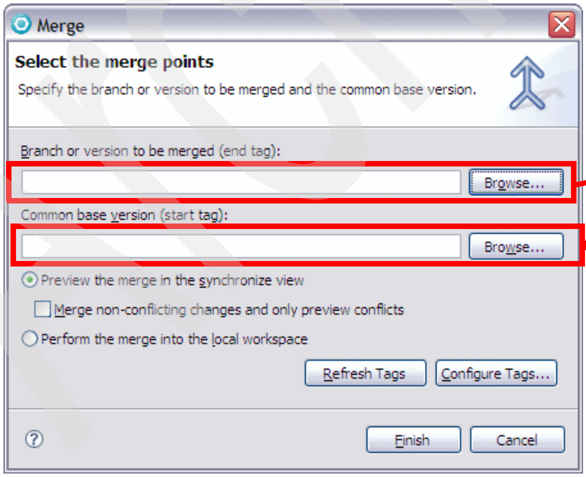
- The name of the branch or version that contains your changes.
- The version from which the branch was created. This is the version name that you supplied when branching.

In our case, the branch is called **Maintenance**, and the version from which we created the branch was called **Branch_Root**.

Merging requires that the target or destination branch be loaded into the workspace before merging in a branch. Because in our scenario the changes are merged to HEAD, the HEAD branch must be loaded in the workspace.

Perform the following on the cvsuser1 work space:

- In the Web perspective right-click **ITSOCVSGuide** and select **Replace With → Another Branch or Version**.
- In the dialog, select **HEAD** and click **OK** to load the latest (HEAD) version of the `ITSOCVSGuide` project into the workspace.
- Right-click the project **ITSOCVSGuide** and select **Team → Merge**. This displays the dialog shown in Figure 27-49, which prompts the user for the start and end points of the merge.



Merge

Select the merge points

Specify the branch or version to be merged and the common base version.

Branch or version to be merged (end tag):

Common base version (start tag):

● Preview the merge in the synchronize view

□ Merge non-conflicting changes and only preview conflicts

○ Perform the merge into the local workspace

A start and end branch or revision should be selected in these 2 boxes

Figure 27-49 Selection of the merge start point

- Click **Browse** for the Common base version (start tag) field and select **BRANCH_ROOT** and click **OK** (Figure 27-50).

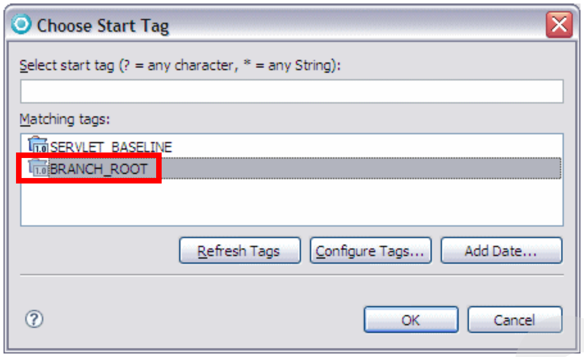


Figure 27-50 Select the start tag

- Click **Browse** for the Branch or version to be merged (end tag) and select **Maintenance** under Branches and click **OK** (Figure 27-51).

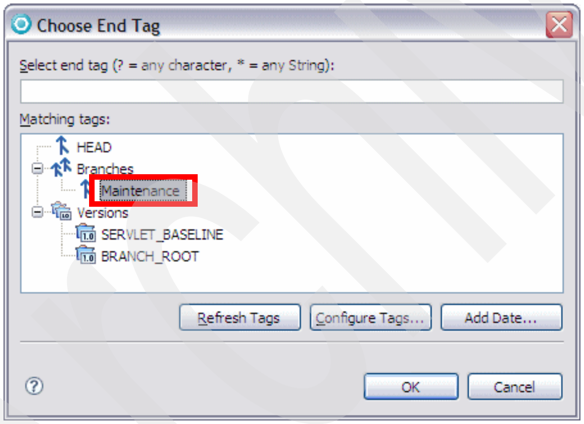
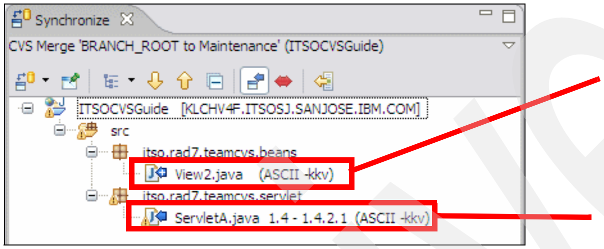


Figure 27-51 Select the end tag

- The Select the Merge Points dialog (Figure 27-49) now shows the start and end tags of the merge, which will be applied to the version in the work space.
- The options to **Preview Merge in the synchronize view** and **Perform the merge into the local workspace** provide the facility to select where to perform the merge:
  - Previewing it in the Synchronize view allows the user to review and make changes to each file as required.

- – Performing the merge into the local workspace applies the changes immediately into the workspace based on preferences selected in the workspace preferences.

- For the example, select **Preview Merge in the synchronize view** and clear **Merge non-conflicting changes and only preview conflicts**.
- Click **Finish** to start the merging, and click **Yes** when prompted whether to move to the **Team Synchronizing perspective.**
- Expand the tree in the Synchronize view to display changes. Verify that there are no conflicts. If there are, then the developer has to resolve these conflicts. In our case, the merge is simple and there are no conflicts (Figure 27-52).



Synchronize
CVS Merge 'BRANCH_ROOT to Maintenance' (ITSOCVSGuide)
ITSOCVSGuide [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
src
itso.rad7.teamcvs.beans
View2.java (ASCII -kkv)
itso.rad7.teamcvs.servlet
ServletA.java 1.4 - 1.4.2.1 (ASCII -kkv)

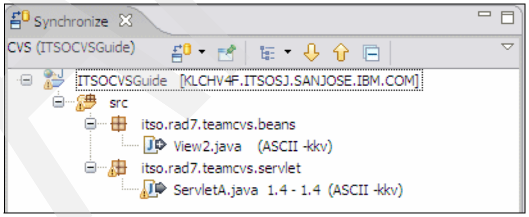
File to be added as part of the merge

File to be changed as part of the merge

Figure 27-52 Files required to be merged

- Right-click the project **ITSOCVSGuide** and select **Merge**.

  This attempts to bring the changes from the branch into the main stream, and because there are no conflicts this should complete successfully.
- In the Web perspective right-click **ITSOCVSGuide** and select **Team → Synchronize with Repository**.
- Click **Yes** to open the Synchronize view.
- Expand the Synchronize view to display the changed files `ServletA.java` and `View2.java` (Figure 27-53).



Synchronize
CVS (ITSOCVSGuide)
ITSOCVSGuide [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
src
itso.rad7.teamcvs.beans
View2.java (ASCII -kkv)
itso.rad7.teamcvs.servlet
ServletA.java 1.4 - 1.4 (ASCII -kkv)

Figure 27-53 CVS updates to HEAD from the merge

This view shows that the file `View2.java` is a new file to be added to the repository and that the file `Servlet1.java` has been changed. This is consistent with the changes that were made in the maintenance branch and now have to be added to the main branch.

- Right-click the project and select **Commit**. In the Commit dialog, add the comment `Merged changes from maintenance branch` and click **OK**.

The changes from the branch have now been merged into the main development branch.

This scenario, although a simple one, highlights the technique required by users to work with branches. In a real scenario there would be conflicts, and this would require resolution between developers. Be aware that branching and concurrent development is a complex process and requires communication and planning between the two teams.

Application Developer provides the tools to assist developers when merging; however, equally important are procedures for handling situations such as branching and merging of code, which should be established among the team early in a project life cycle.

Working with patches

Application Developer provides the facility for developers to be able to share work when they only have read access to a CVS repository. In this circumstance the developer that does not have full access to the repository can create a patch and forward it to another developer with write access, and the patch can be applied to the project and the changes committed.

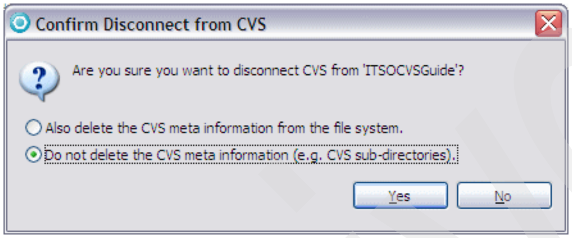
Such a configuration is useful when access to the source code repository has to be restricted to a small number of users to prevent uncoordinated changes corrupting the quality of the code. Any number of users can then contribute changes and fixes to the repository using patches, but only through designated code minders who can commit the work and who have the opportunity to review changes before applying them to the repository.

This is done through the **Team** → **Create Patch** and **Team** → **Apply patch** options available from the Project Explorer context menu. A patch can contain a single file, a project or any other combination of resources on the workspace. The Application Developer online help has a complete description of how to work with CVS patches.

Disconnecting a project

For many reasons (for example, to disassociate a project from one repository to allow it to be added to another repository) a developer might want to disconnect a project from the current CVS repository. To perform this task, complete the following steps:

- In the Web perspective right-click the **ITSOCVSGuide** project and select **Team** → **Disconnect**.
- A prompt opens, asking to confirm the disconnect from CVS and if the CVS control information should be deleted (Figure 27-54).



Confirm Disconnect from CVS

Are you sure you want to disconnect CVS from 'ITSOCVSGuide'?

○ Also delete the CVS meta information from the file system.

● Do not delete the CVS meta information (e.g. CVS sub-directories).

Figure 27-54 Disconnect confirmation

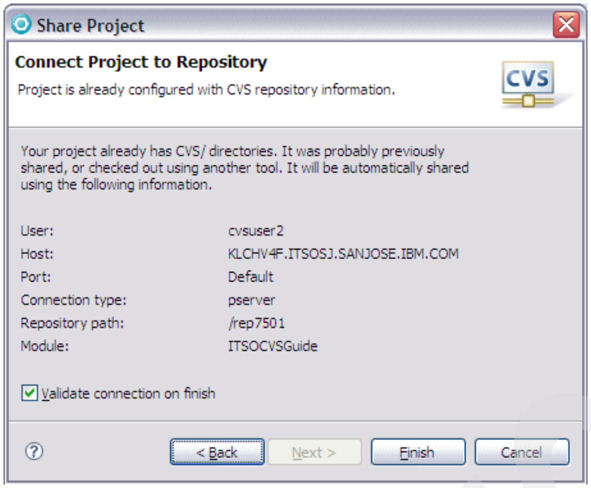
- Select **Do not delete the CVS meta information** and click **OK**.

By not deleting the CVS meta information, we can reconnect the project with the CVS repository later more easily. If the meta information is removed, CVS cannot determine which revision in the repository a particular file is associated with

Reconnect

You can reconnect a project to the repository by selecting **Team** → **Share Project**. Reconnecting is easier if the CVS meta information was not deleted:

- If the meta information was deleted, the user is prompted to synchronize the code with the an existing revision in the repository.
- If the meta information is still available, the user is shown the original CVS repository information and given the opportunity to complete the re-connect (Figure 27-55).


Share Project
Connect Project to Repository
Project is already configured with CVS repository information.
CVS

Your project already has CVS/ directories. It was probably previously shared, or checked out using another tool. It will be automatically shared using the following information.

User:	cvsuser2
Host:	KLCHV4F.ITSOSJ.SANJOSE.IBM.COM
Port:	Default
Connection type:	pserver
Repository path:	/rep7501
Module:	ITSOCVSGuide

☑ Validate connection on finish

?
< Back
Next >
Finish
Cancel

Figure 27-55 Reconnect to repository with original CVS meta information

Team Synchronizing perspective

The Team Synchronizing perspective in Application Developer has been used in the examples in this chapter but has not yet described in detail. The purpose of this perspective is to provide to the user with a tool to identify changes repository compared with what is on the local workspace, and assist in synchronizing the two together.

Features provided with the Team Synchronize perspective include:

- Provide a comparison of changes in the workspace (as described in “Comparisons in CVS” on page 1256).
- Committing the changes made to the repository (as described in the previous scenarios).
- Create custom synchronization of a subset of resources in the workspace.
- Schedule checkout synchronization.

Custom configuration of resource synchronization

The Synchronize view provides the ability to create custom synchronization sets for the purpose of synchronizing only identified resources that a developer might be working on. This allows the developer to focus on changes that are part of their scope of work and ensure they are aware of the changes that occur without worrying about changes to other areas.

The developer can make changes to the other areas or someone else might check-in changes to these parts, but only the resources in the defined set are synchronized.

This is handy if the changes being worked on are localized and other areas of the code are changing in ways not important for the work at hand. On the other hand, problems can occur with this mode of operation as well. Developers have to be careful that important changes to the non-synchronized parts are not ignored for long periods of time.

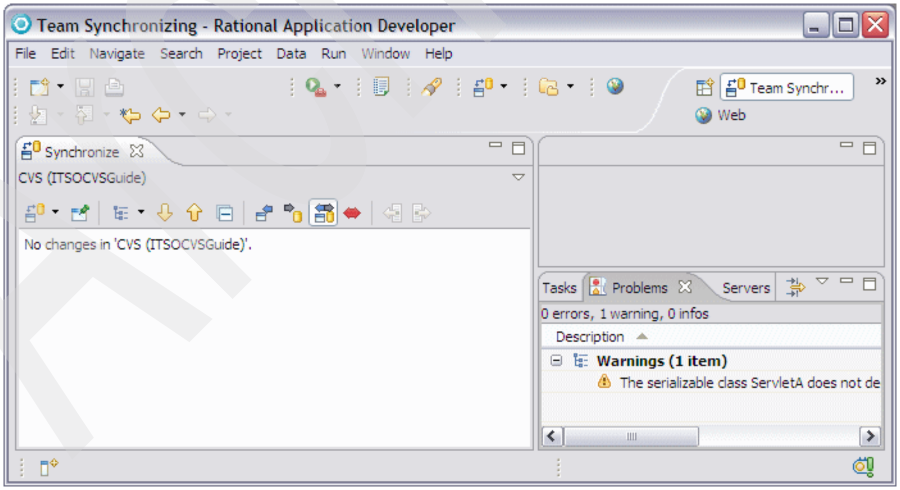
> **Important:** Custom synchronization is most effective when an application is designed with defined interfaces, where the partitioning of work is clear. However, even in this scenario, it should be used with caution because it can introduce additional work in the development cycle for final product integration. Procedures have to be documented and enforced to ensure that integration is incorporated as part of the work pattern for this scenario.

The example scenario (again using the `ITSOCVSGuide` project) demonstrates custom synchronization, through two procedures:

- Full synchronization of the project `ITSOCVSGuide`
- Partial synchronization of the servlet `ServletA.java`

To perform the example. complete the following for the cvsuser1 workspace:

- Open the **Team Synchronizing** perspective (Figure 27-56).


Team Synchronizing - Rational Application Developer
File Edit Navigate Search Project Data Run Window Help
Team Synchr...
Web
Synchronize
CVS (ITSOCVSGuide)
Tasks
Problems
Servers
Description

No changes in 'CVS (ITSOCVSGuide)'.

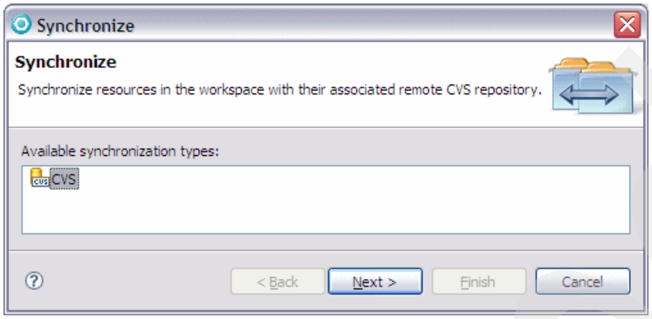
0 errors, 1 warning, 0 infos

Warnings (1 item)

The serializable class ServletA does not de

Figure 27-56 Team Synchronizing perspective

- Click the Synchronize button at the top the Synchronize view (left pane) and click **Synchronize** to add a new synchronization definition.
- In the Synchronize dialog, select **CVS** and click **Next** (Figure 27-57).



Synchronize

Synchronize

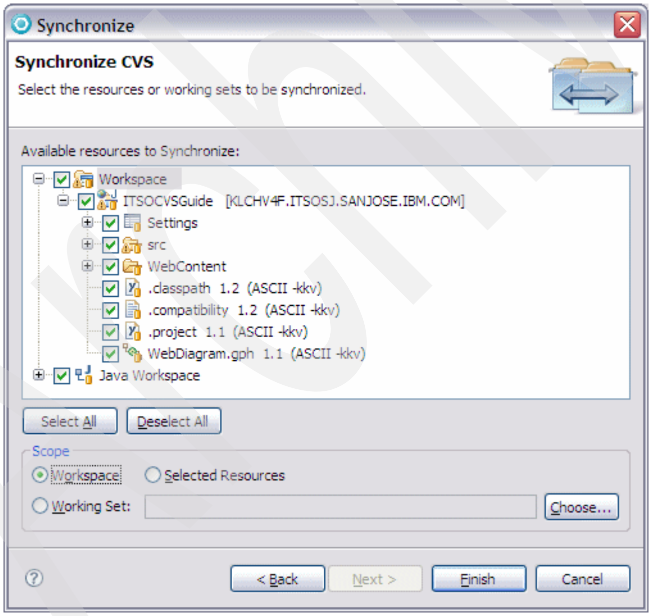
Synchronize resources in the workspace with their associated remote CVS repository.

Available synchronization types:

CVS

Figure 27-57 Synchronize dialog

- Expand the project tree to view the contents and note that all resources in the workspace are selected. Accept the defaults for the Synchronize CVS dialog, and click **Finish** (Figure 27-58).



Synchronize CVS

Select the resources or working sets to be synchronized.

Available resources to Synchronize:

```
⊟ ☑ Workspace
   ⊟ ☑ ITSOCVSGuide  [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
      ⊞ ☑ Settings
      ⊞ ☑ src
      ⊞ ☑ WebContent
        ☑ .classpath  1.2  (ASCII -kkv)
        ☑ .compatibility  1.2  (ASCII -kkv)
        ☑ .project  1.1  (ASCII -kkv)
        ☑ WebDiagram.gph  1.1  (ASCII -kkv)
⊞ ☑ Java Workspace
```

Scope

● Workspace ○ Selected Resources

○ Working Set:

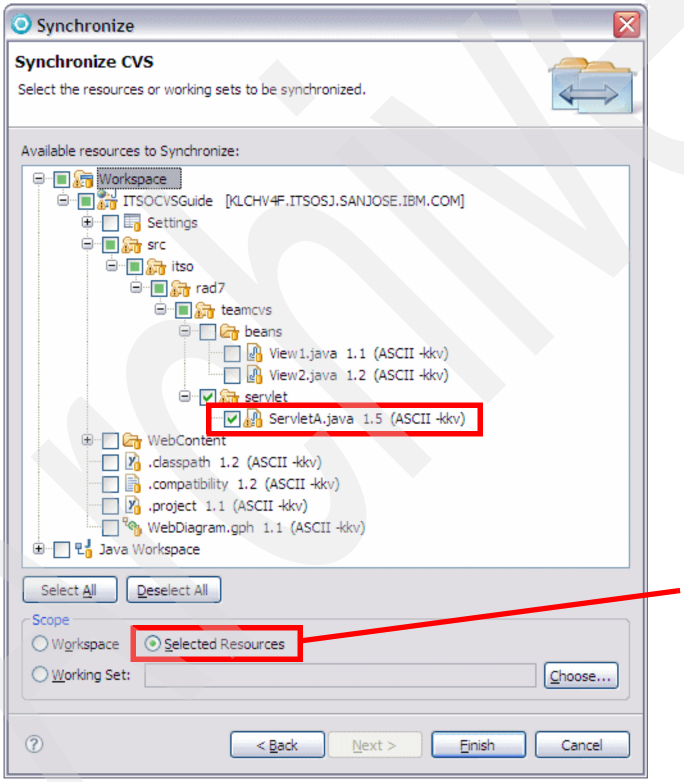
Figure 27-58 Default synchronization of the project ITSOCVSGuide

If there are no changes, then a dialog box opens, saying `Synchronizing: No changes found`, and in the Synchronize view a message of `No changes in 'CVS (Workspace)'`.

Note: When using Application Developer V7.0.0.2 the list of resources shows a **Workspace** and a **Java Workspace** folder. If a customized resource set is required here it can be selected from either folder with the same final result.

- To preserve this synchronization, click **Pin Current Synchronization** .
- Add a new synchronization by clicking the Synchronize icon at the top of the Synchronize view.
- In the Synchronize dialog, select **CVS** and click **Next** (Figure 27-57).
- Expand the project tree under JavaSource to view the contents, click **Deselect All** to deselect all the resources, and select only **ServletA.java**.

  The Synchronize dialog should appear as shown in Figure 27-59.



Synchronize

Synchronize CVS

Select the resources or working sets to be synchronized.

Available resources to Synchronize:

```
⊟ ■ Workspace
   ⊟ ■ ITSOCVSGuide   [KLCHV4F.ITSOSJ.SANJOSE.IBM.COM]
      ⊞ □ Settings
      ⊟ ■ src
         ⊟ ■ itso
            ⊟ ■ rad7
               ⊟ ■ teamcvs
                  ⊟ □ beans
                        □ View1.java  1.1  (ASCII -kkv)
                        □ View2.java  1.2  (ASCII -kkv)
                  ⊟ ☑ servlet
                        ☑ ServletA.java  1.5  (ASCII -kkv)
      ⊞ □ WebContent
        □ .classpath  1.2  (ASCII -kkv)
        □ .compatibility  1.2  (ASCII -kkv)
        □ .project  1.1  (ASCII -kkv)
        □ WebDiagram.gph  1.1  (ASCII -kkv)
⊞ □ Java Workspace
```

Scope

○ Workspace ● Selected Resources

○ Working Set:

Select this option to make sure only selected resources are part of the synchronization

Figure 27-59 Selecting ServletA.java for synchronization

- Select **Selected Resources** and click **Finish**.
- If there are no changes, then a dialog box opens, saying `Synchronizing: No changes found`, and in the Synchronize view a message of `No changes in 'CVS (Workspace)'`. To preserve this synchronization, click **Pin Current Synchronization** .
- In the list of synchronizations (click the icon at the top of the Synchronize view,) two synchronizations should appear (Figure 27-60).

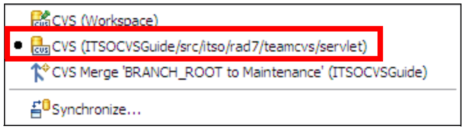

CVS (Workspace)
CVS (ITSOCVSGuide/src/itso/rad7/teamcvs/servlet)
CVS Merge 'BRANCH_ROOT to Maintenance' (ITSOCVSGuide)
Synchronize...

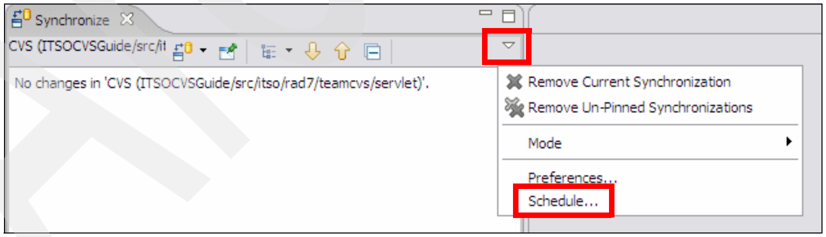
Figure 27-60 List of synchronizations created

Schedule synchronization

Application Developer allows the scheduling synchronization of the workspace. This feature follows on from “Custom configuration of resource synchronization” on page 1272, in which a user would like to schedule the synchronization that has been defined. Scheduling a synchronization can only be performed for synchronizations that have been pinned.

To demonstrate this feature, assume that the project `ITSOCVSGuide` is loaded in the workspace and a synchronization has been defined for this project and pinned. Scheduling of this project for synchronization is then performed using the following steps:

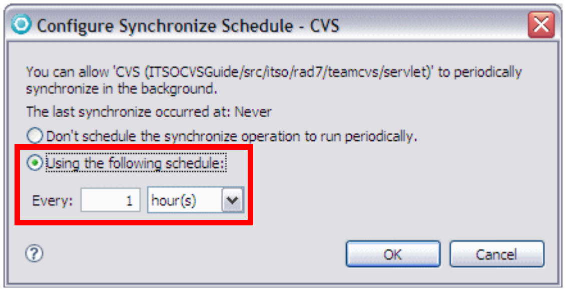
- In the Synchronize view select **Schedule** from the drop-down list (Figure 27-61).



No changes in 'CVS (ITSOCVSGuide/src/itso/rad7/teamcvs/servlet)'.

Figure 27-61 Drop-down selection for scheduling synchronization

- In the Configure Synchronize Schedule dialog, select **Using the following schedule:** and the time period that you want to synchronize (Figure 27-62). Click **OK**.



Configure Synchronize Schedule - CVS

You can allow 'CVS (ITSOCVSGuide/src/itso/rad7/teamcvs/servlet)' to periodically synchronize in the background.

The last synchronize occurred at: Never

○ Don't schedule the synchronize operation to run periodically.

● Using the following schedule:

Every: 1 hour(s)

Figure 27-62 Setting synchronization schedule

- The user might be prompted to pin the current CVS synchronization, if it is not already pinned. Click **Yes**.

Assuming that one hour is chosen, the project `ITSOCVSGuide` is synchronized every hour to ensure that the latest updates are available. This action performs the synchronize operation and shows any changes available in the synchronize view, where the user can accept or postpone integrating the changes as appropriate.

More information

The help feature provided with Application Developer has a large section on using the Team Synchronizing and the CVS Repository Exploring perspectives and describes all the features covered in this chapter.

In addition, the following URLs provide further information for the topics covered in this book:

- **CVS home page**—The main source of information for CVS:

  `http://www.nongnu.org/cvs/`

- **CVSNT home page**—This is the main source of information for CVSNT, which is the CVS server implementation for Windows machines and the CVS Server software used in this chapter. See the following URL:

  `http://cvsnt.org/wiki`

- **Eclipse CVS information**—The CVS features available in Application Developer V7.0 come from Eclipse 3.2. The following link is the main information page for this project. It contains documentation, downloads, and even the source code:

  `http://wiki.eclipse.org/index.php/CVS_Howto`

- **Tortoise CVS home page**—This is another CVS client that lets users perform CVS operations from Windows Explorer. It provides most of the features of Application Developer and is available under the GNU public license. This can come in handy when CVS operations are required outside Application Developer. See the following URL:

  `http://www.tortoisecvs.org`

- **CVS command line reference**—Some operations on the CVS server (for example changing the password of a user) is best done from the command line. The following URL provides a quick reference for the CVS command line interface:

  `http://refcards.com/docs/forda/cvs/cvs-refcard-a4.pdf`

Part 6

Appendixes

A

Product installation

The objective of this appendix is to highlight the key installation considerations and options, identify components installed while writing this book, and provide a general awareness regarding the use of IBM Installation Manager to install IBM Rational Application Developer V7.0.0.3.

The appendix is organized into the following sections:

- IBM Installation Manager
- Installing IBM Rational Application Developer
- Installing the WebSphere Portal V6.0 test environment
- Installing IBM Rational ClearCase LT

IBM Installation Manager

IBM Installation Manager is used to install Rational Application Developer. IBM Installation Manager is a program that helps you install the Rational desktop product packages on your workstation. It also helps you update, modify, and uninstall this and other packages that you install. A package can be a product, a group of components, or a single component that is designed to be installed by Installation Manager.

There are six wizards in the Installation Manager that make it easy to maintain your package through its lifecycle, as seen in Figure A-1.

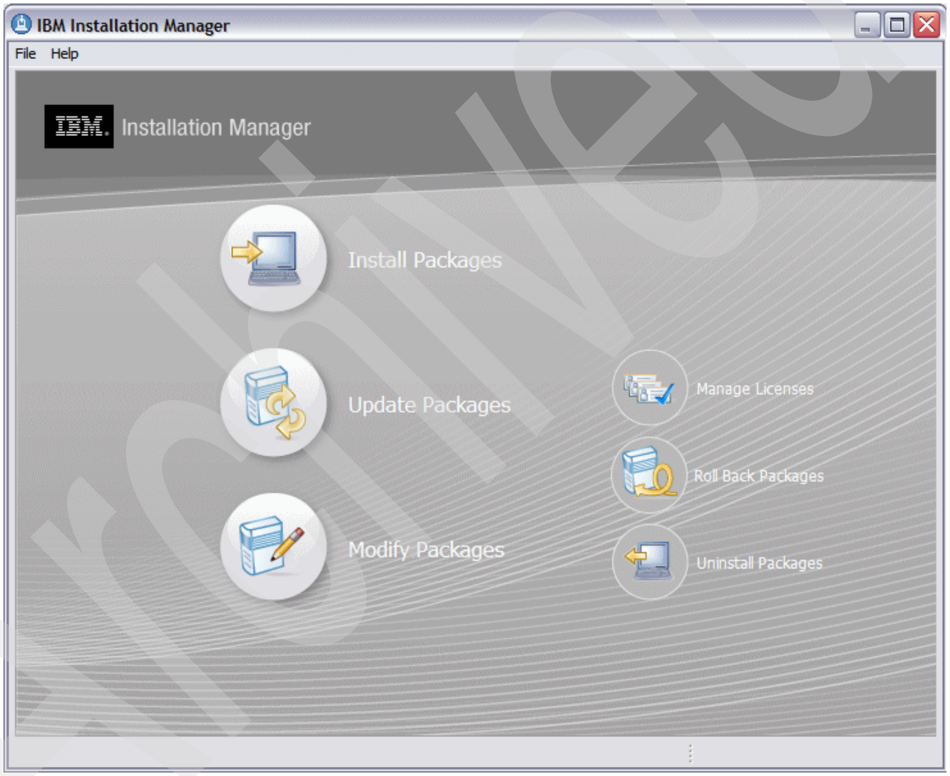


Figure A-1 Installation Manager

These are the six wizards:

- The **Install Packages** wizard walks you through the installation process.
- The **Update Packages** wizard searches for available updates to packages you have installed.

- With the **Modify Packages** wizard, you can modify certain elements of a package you have already installed.
- The **Manage Licenses** wizard helps you set up the licenses for your packages.
- Using the **Roll Back Packages** wizard, you can revert back to a previous version of a package.
- The **Uninstall Packages** wizard removes a package from your computer.

Installing IBM Rational Application Developer

Note: For detailed information on the IBM Rational Application Developer V7.0 installation, refer to the product guides found in the `disk1\documentation` directory:

- Installation Guide, IBM Rational Application Developer v7.0. Open `install.html` in a Web browser, or `intall.pdf` using Adobe Reader®.
- Release Notes, IBM Rational Application Developer v7.0. Open `readme.html` in a Web browser.

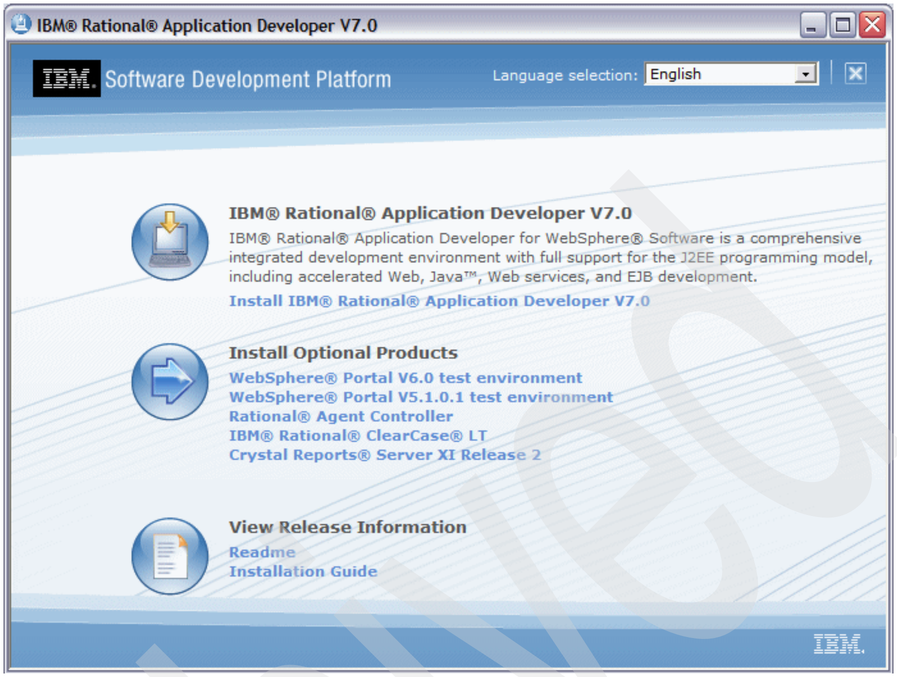
There are a number of scenarios that you can follow when installing Rational Application Developer:

- Installing from the CDs
- Installing from a downloaded electronic image on your workstation
- Installing from an electronic image on a shared drive
- Installing from a repository on an HTTP or HTTPS server

Note that in the latter three scenarios, you can choose to run the Installation Manager program in silent mode to install Rational Application Developer. See the installation guide for details on each scenario.

While writing this Redbooks publication, we installed IBM Rational Application Developer V7.0 from a downloaded electronic image on the workstation. The steps are listed as follows:

- Start the Rational Application Developer Installer by running **launchpad.exe** from the **disk1** folder.
- The IBM Rational Application Developer V7.0 components have separate installations from the main Launchpad Base page, as seen in Figure A-2.


IBM® Rational® Application Developer V7.0
IBM. Software Development Platform
Language selection: English
IBM.

IBM® Rational® Application Developer V7.0

IBM® Rational® Application Developer for WebSphere® Software is a comprehensive integrated development environment with full support for the J2EE programming model, including accelerated Web, Java™, Web services, and EJB development.

Install IBM® Rational® Application Developer V7.0

Install Optional Products

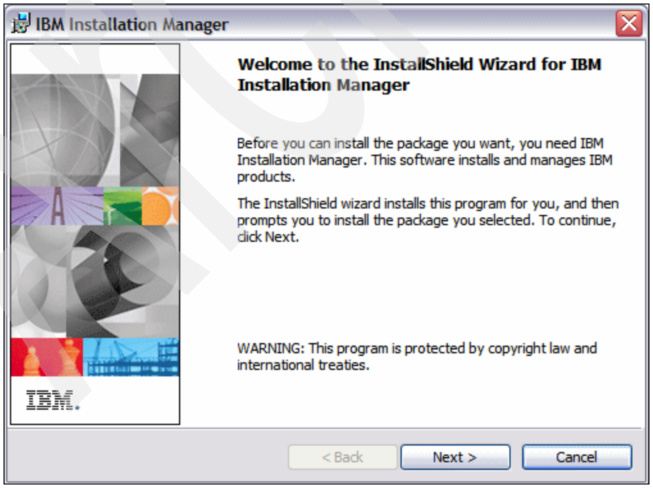
WebSphere® Portal V6.0 test environment
WebSphere® Portal V5.1.0.1 test environment
Rational® Agent Controller
IBM® Rational® ClearCase® LT
Crystal Reports® Server XI Release 2

View Release Information

Readme
Installation Guide

Figure A-2 Launchpad base page

- Select **Install IBM Rational Application Developer v7.0**. You are directed to install IBM Installation Manager because it is not installed yet, as seen in Figure A-3. Click **Next**.



Welcome to the InstallShield Wizard for IBM Installation Manager

Before you can install the package you want, you need IBM Installation Manager. This software installs and manages IBM products.

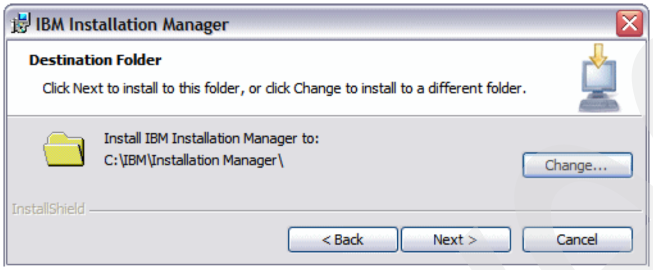
The InstallShield wizard installs this program for you, and then prompts you to install the package you selected. To continue, click Next.

WARNING: This program is protected by copyright law and international treaties.

Figure A-3 Install IBM Installation Manager

- In the License Agreement window, review the terms in the license agreement, select **I accept the terms of the license agreement**, and click **Next**.
- You are directed to the destination Folder of IBM installation manager. The default folder is very long. Having a long installation path might cause path or classpath problems when you run applications in Application Developer. We recommend that you use a short installation path (Figure A-4). Click **Next**.

```
C:\IBM\Installation Manager\
```


IBM Installation Manager

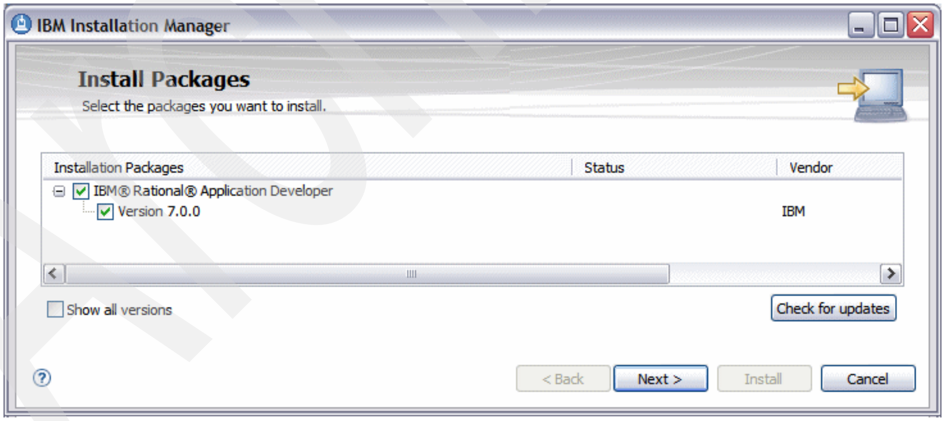
Destination Folder

Click Next to install to this folder, or click Change to install to a different folder.

Install IBM Installation Manager to:
C:\IBM\Installation Manager\

Figure A-4 Destination folder for IBM Installation Manager

- Click **Install** to begin the installation of IBM Installation Manager.
- After the installation of IBM Installation Manager, click **Finish**.
- The installation manager starts up. In the Install Packages page, make sure you click **Check for Updates**, because we have to install the latest version of Application Developer (Figure A-5).


IBM Installation Manager

Install Packages

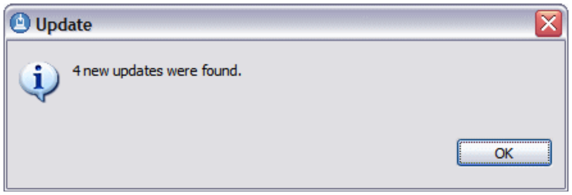
Select the packages you want to install.

Installation Packages	Status	Vendor
⊟ ☑ IBM® Rational® Application Developer		
☑ Version 7.0.0		IBM

□ Show all versions

Figure A-5 Check for updates

- A dialog box pops up (Figure A-6). Click **OK**.

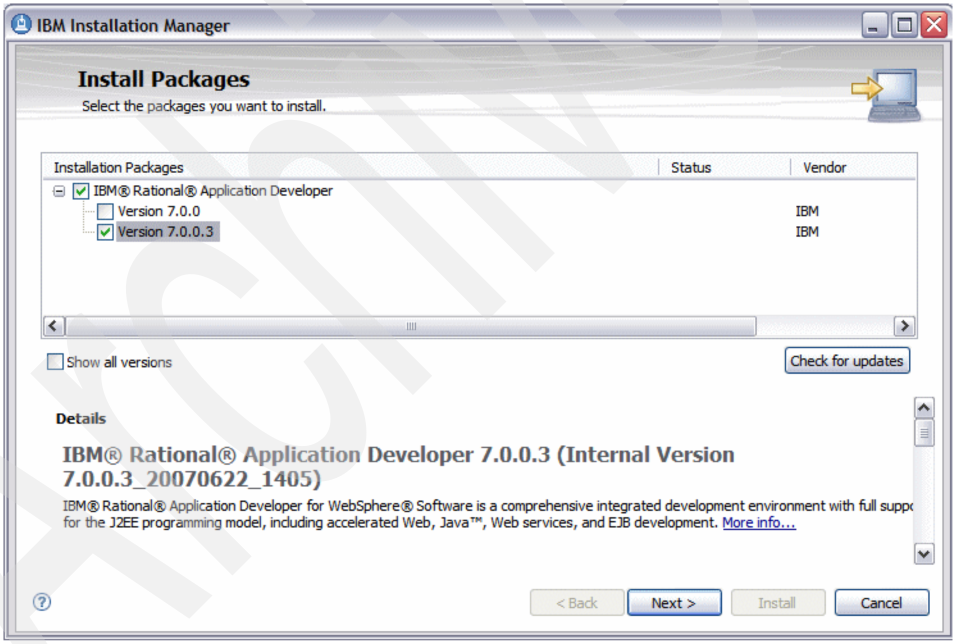


Update

4 new updates were found.

Figure A-6 Updates found

- At the time of the writing, the latest version of Application Developer was 7.0.0.3. Note that IBM releases a fix pack approximately every three months. You might see a newer version of Application Developer at the time you install Application Developer. Each new version of Application Developer contains a large quantity of fixes. We recommend that the customer install the latest version to avoid encountering problems that have already been fixed. Now Version 7.0.0.3 updates appear in the installation packages list. Select **Version 7.0.0.3** (Figure A-7) and click **Next**.



IBM Installation Manager

Install Packages

Select the packages you want to install.

Installation Packages	Status	Vendor
☑ IBM® Rational® Application Developer		
□ Version 7.0.0		IBM
☑ Version 7.0.0.3		IBM

□ Show all versions

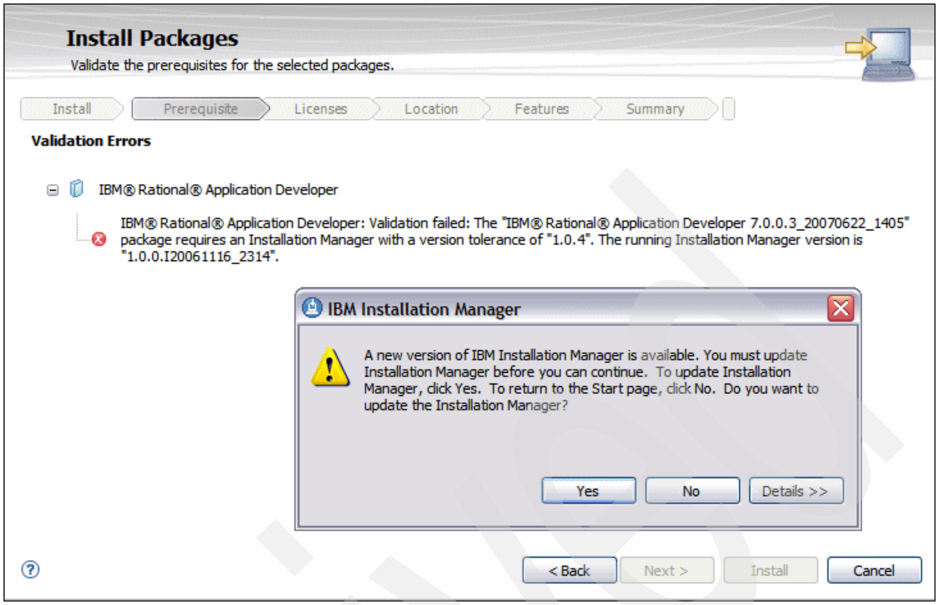
Details

IBM® Rational® Application Developer 7.0.0.3 (Internal Version 7.0.0.3_20070622_1405)

IBM® Rational® Application Developer for WebSphere® Software is a comprehensive integrated development environment with full suppo
for the J2EE programming model, including accelerated Web, Java™, Web services, and EJB development. More info...

Figure A-7 7.0.0.3 updates

- You are prompted to update the IBM Installation Manager, as seen in Figure A-8. Click **Yes**.



Validation Errors

IBM® Rational® Application Developer

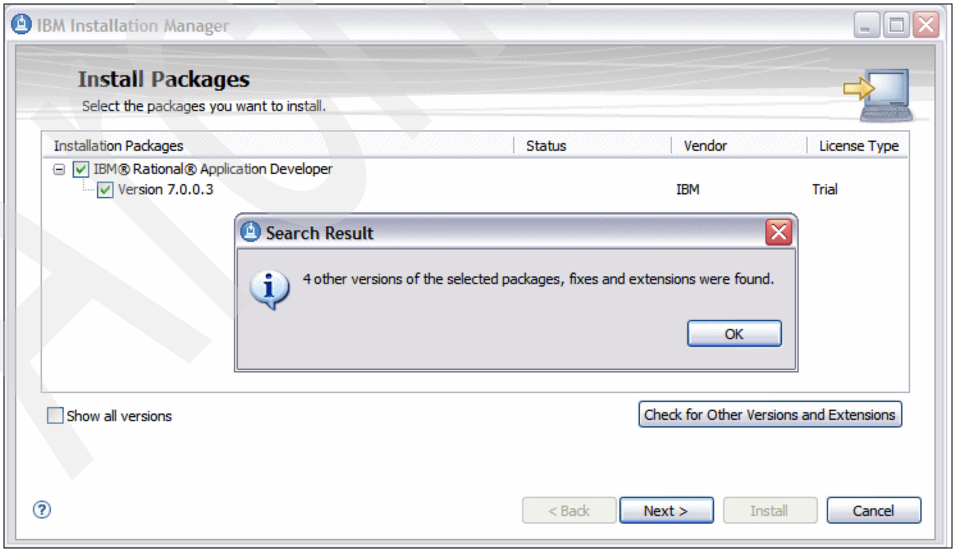
IBM® Rational® Application Developer: Validation failed: The "IBM® Rational® Application Developer 7.0.0.3_20070622_1405" package requires an Installation Manager with a version tolerance of "1.0.4". The running Installation Manager version is "1.0.0.I20061116_2314".

IBM Installation Manager

A new version of IBM Installation Manager is available. You must update Installation Manager before you can continue. To update Installation Manager, click Yes. To return to the Start page, click No. Do you want to update the Installation Manager?

Figure A-8 Update IBM Installation Manager

- After the Installation Manager is updated and started, click **Check for Other Versions and Extensions**. The updated Installation Manager finds the 7003 updates again (Figure A-9).



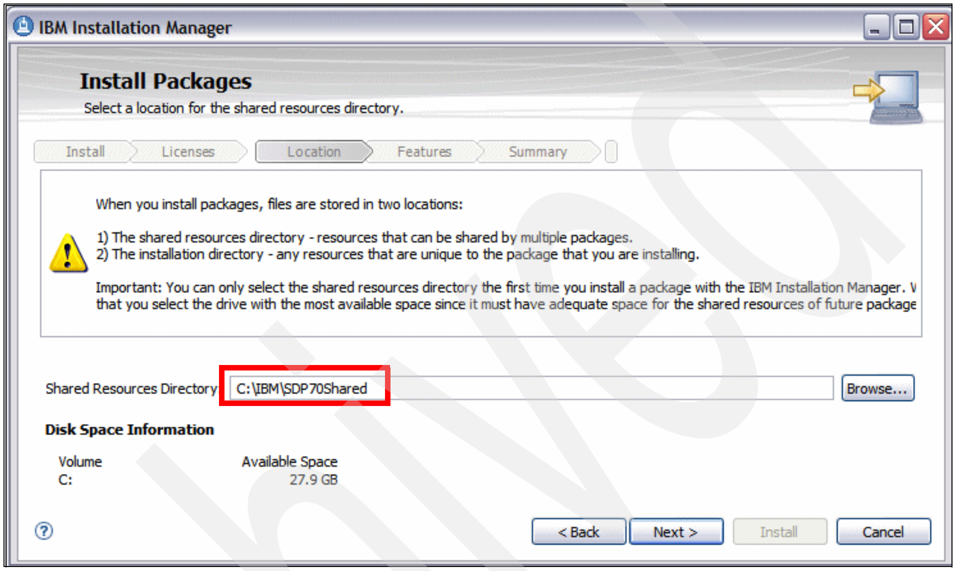
Installation Packages	Status	Vendor	License Type
☑ IBM® Rational® Application Developer			
☑ Version 7.0.0.3		IBM	Trial

Search Result

4 other versions of the selected packages, fixes and extensions were found.

Figure A-9 Updated Installation Manager finds the 7.0.0.3 updates

- Click **OK** and then click **Next**.
- Select **I accept the terms in the license agreements** and then click **Next**.
- In the Select a location for the shared resources directory page, change the Shared Resource Directory from `C:\Program Files\IBM\SDP70Shared` to **`C:\IBM\SDP70Shared`** (Figure A-10). Click **Next**.



Select a location for the shared resources directory.

When you install packages, files are stored in two locations:

1) The shared resources directory - resources that can be shared by multiple packages.
2) The installation directory - any resources that are unique to the package that you are installing.

Important: You can only select the shared resources directory the first time you install a package with the IBM Installation Manager. V
that you select the drive with the most available space since it must have adequate space for the shared resources of future package

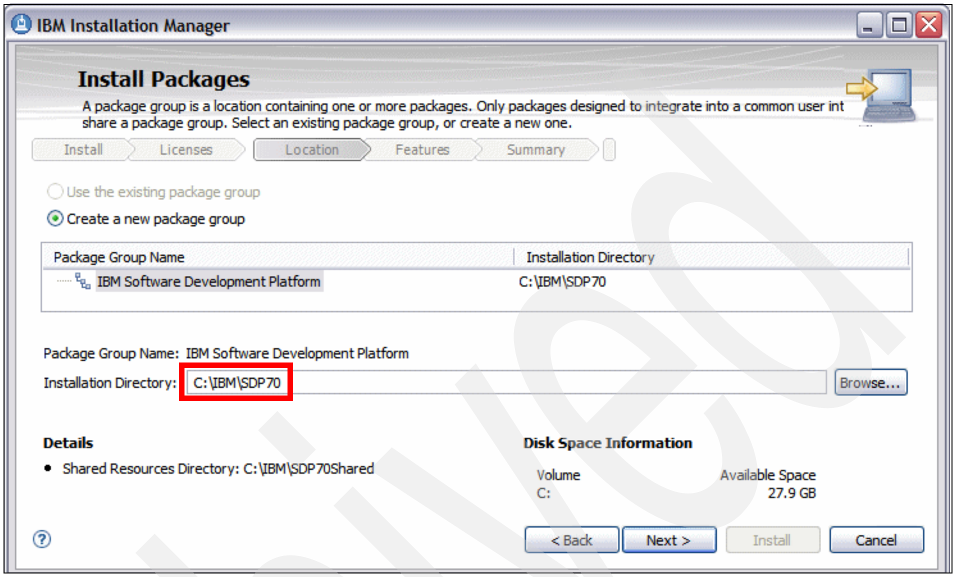
Shared Resources Directory C:\IBM\SDP70Shared

Disk Space Information

Volume	Available Space
C:	27.9 GB

Figure A-10 Shared Resource Directory

- In the Package Group page, select **Create a new package group**, and set the installation directory for the package group to `C:\IBM\SDP70` (Figure A-11), and click **Next**.



IBM Installation Manager

Install Packages

A package group is a location containing one or more packages. Only packages designed to integrate into a common user int share a package group. Select an existing package group, or create a new one.

○ Use the existing package group

● Create a new package group

Package Group Name	Installation Directory
IBM Software Development Platform	C:\IBM\SDP70

Package Group Name: IBM Software Development Platform

Installation Directory: C:\IBM\SDP70

Details

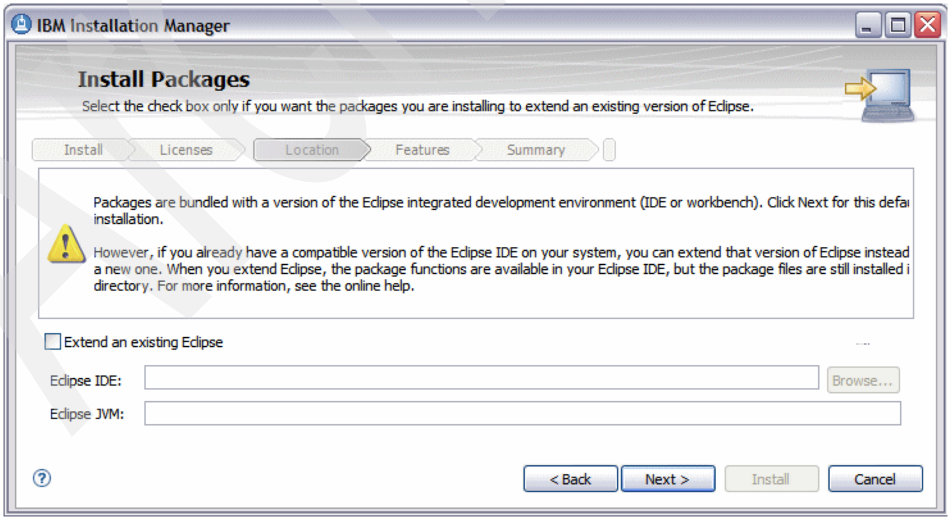
- Shared Resources Directory: C:\IBM\SDP70Shared

Disk Space Information

Volume	Available Space
C:	27.9 GB

Figure A-11 Installation location for package group

- In the Extend an existing Eclipse page, we do not want to extend an existing Eclipse. Leave this page as default and click **Next** (Figure A-12).



IBM Installation Manager

Install Packages

Select the check box only if you want the packages you are installing to extend an existing version of Eclipse.

Packages are bundled with a version of the Eclipse integrated development environment (IDE or workbench). Click Next for this defa installation.

However, if you already have a compatible version of the Eclipse IDE on your system, you can extend that version of Eclipse instead a new one. When you extend Eclipse, the package functions are available in your Eclipse IDE, but the package files are still installed i directory. For more information, see the online help.

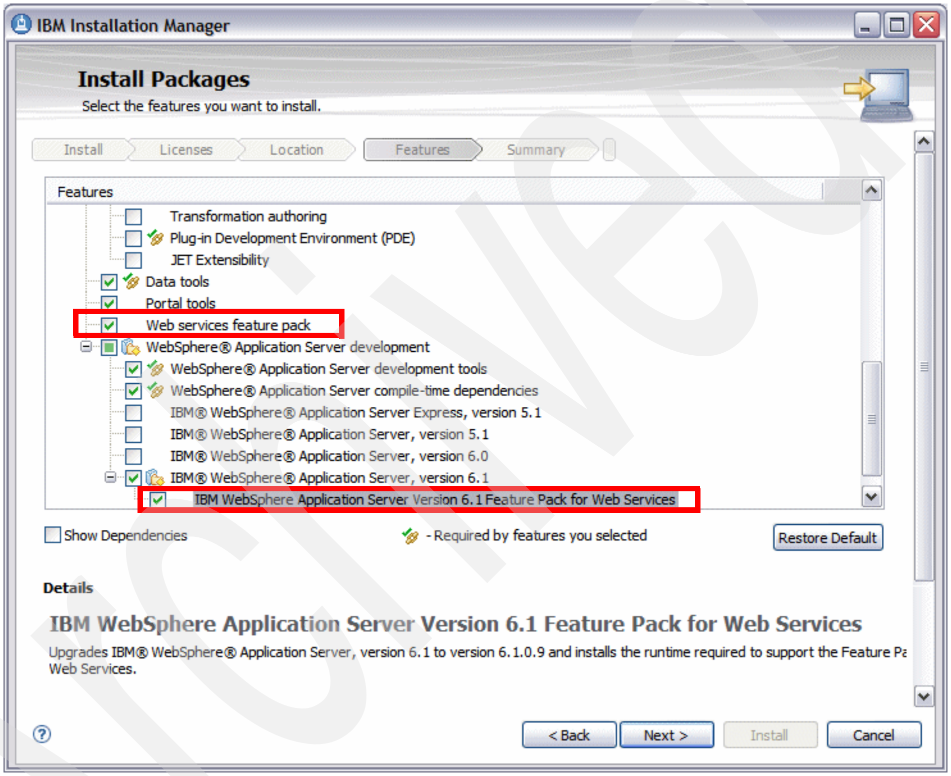
□ Extend an existing Eclipse

Eclipse IDE:

Eclipse JVM:

Figure A-12 Extending and existing Eclipse

- In the Select the languages you want to install page, choose your language and click **Next**.
- In the Select the features you want to install page, select the package features that you want to install. If you want to complete all the chapters for this Redbooks publication, you have to select **Struts tools**, **Data tools**, **Portal tools**, **Web services feature pack**, **IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services**, in addition to the default selections. Click **Next** (Figure A-13).



Install Packages

Select the features you want to install.

Features

☐ Transformation authoring
☐ Plug-in Development Environment (PDE)
☐ JET Extensibility
☑ Data tools
☑ Portal tools
☑ Web services feature pack
WebSphere® Application Server development
☑ WebSphere® Application Server development tools
☑ WebSphere® Application Server compile-time dependencies
☐ IBM® WebSphere® Application Server Express, version 5.1
☐ IBM® WebSphere® Application Server, version 5.1
☐ IBM® WebSphere® Application Server, version 6.0
☑ IBM® WebSphere® Application Server, version 6.1
☑ IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services

☐ Show Dependencies

- Required by features you selected

Details

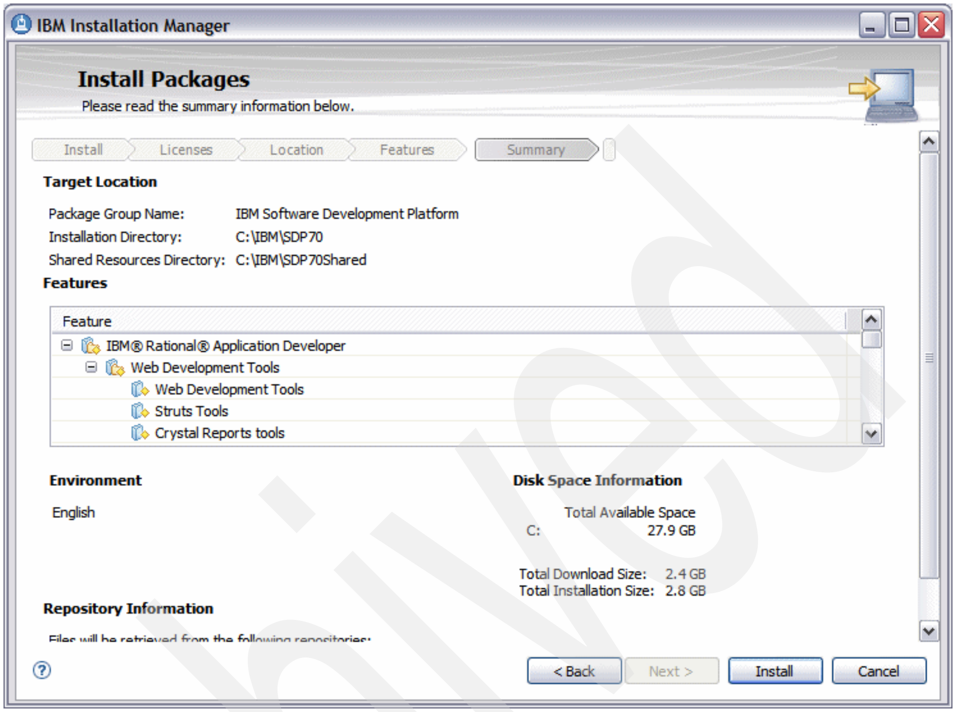
IBM WebSphere Application Server Version 6.1 Feature Pack for Web Services

Upgrades IBM® WebSphere® Application Server, version 6.1 to version 6.1.0.9 and installs the runtime required to support the Feature Pa
Web Services.

Figure A-13 Select the features you want to install

Note: You only have to install the Web Services Feature Pack if you want to explore the new Web services functions of the feature pack.

- On the Summary page, review your choices, and click **Install** (Figure A-14).



Install Packages

Please read the summary information below.

Target Location

Package Group Name: IBM Software Development Platform
Installation Directory: C:\IBM\SDP70
Shared Resources Directory: C:\IBM\SDP70Shared

Features

Feature
⊟ IBM® Rational® Application Developer
⊟ Web Development Tools
Web Development Tools
Struts Tools
Crystal Reports tools

Environment

English

Disk Space Information

	Total Available Space
C:	27.9 GB

Total Download Size: 2.4 GB
Total Installation Size: 2.8 GB

Repository Information

Files will be retrieved from the following repositories:

Figure A-14 Install summary

- When the installation process is completed, a message confirms the success of the process. Click **View log file** to open the installation log file for the current session in a new window. Close the Installation Log window to continue.
- In the Install Package wizard, do not start IBM Rational Application Developer when you exit. Click **Finish** to close the Install Package wizard, and you are returned to the Start page of Installation Manager.

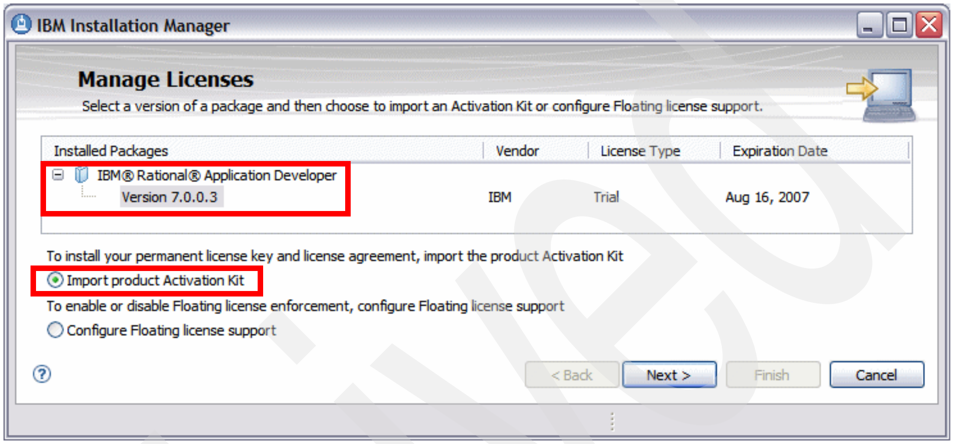
Installing the license for Rational Application Developer

You have two options on how to enable licensing for Rational Application Developer:

- Importing a product activation kit
- Enabling Rational Common Licensing to obtain access to floating license keys

In this section, we show you how to import a product activation kit:

- ► Start IBM Installation Manager.
- ► On the main page, click **Manage Licenses**.
- ► The Manage Licenses dialog opens (Figure A-15). Select **Version 7.0.0.3** and **Import product enablement kit**. Click **Next**.



Manage Licenses

Select a version of a package and then choose to import an Activation Kit or configure Floating license support.

Installed Packages	Vendor	License Type	Expiration Date
IBM® Rational® Application Developer			
Version 7.0.0.3	IBM	Trial	Aug 16, 2007

To install your permanent license key and license agreement, import the product Activation Kit

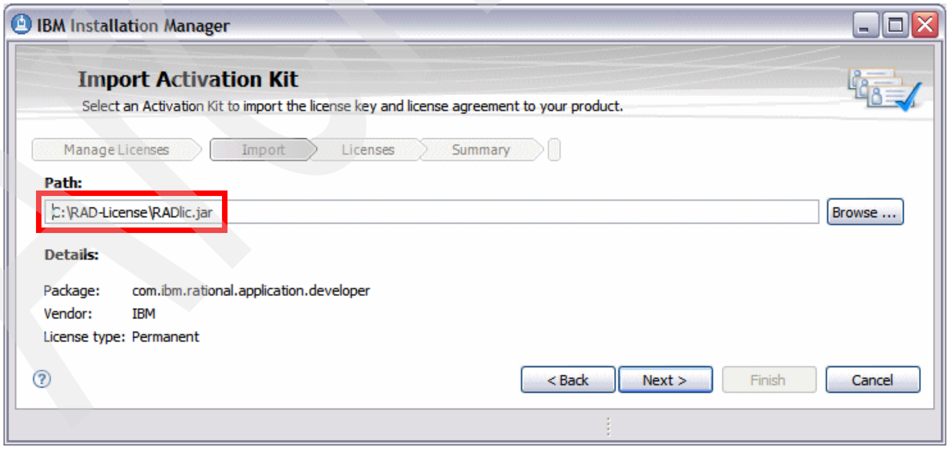
● Import product Activation Kit

To enable or disable Floating license enforcement, configure Floating license support

○ Configure Floating license support

Figure A-15 Manage Licenses

- ► In the Import Activation Kit page, browse to the path of the download location for the kit, then select the appropriate Java archive (JAR) file and click **Open**. Click **Next**.



Import Activation Kit

Select an Activation Kit to import the license key and license agreement to your product.

Path: C:\RAD-License\RADlic.jar

Details:

Package: com.ibm.rational.application.developer

Vendor: IBM

License type: Permanent

Figure A-16 Import Activation Kit

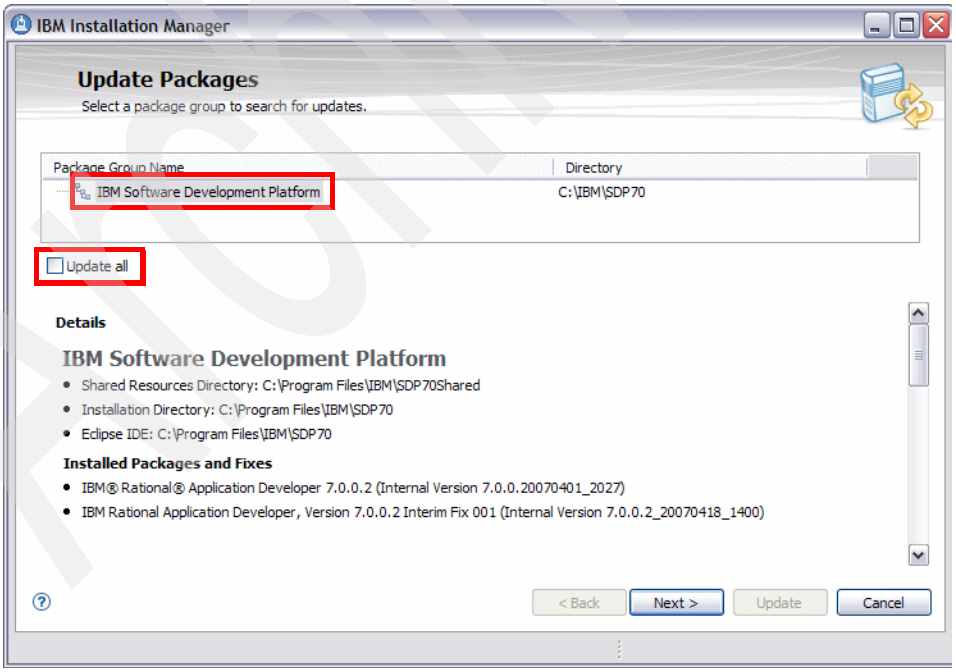
- In the Licenses page, select **I accept the terms in the license agreements**. Click **Next**.
- In the summary page, click **Finish**.
- The product activation kit with its permanent license key is imported to Application Developer. The Manage Licenses wizard indicates whether the import is successful.

Updating Rational Application Developer

Once Application Developer has been installed, the Installation Manager provides an interface to update the product. This replaces the Rational Product Updater provided in Application Developer V6.0.

In the Installation Manager overview, select **Update Packages** (see Figure A-1 on page 1282).

Selecting Update Packages takes you to the dialogue shown in Figure A-17. You can select **Update all** to update all products installed, or you can be explicit about a particular product. Clicking **Next** searches for the updates to the products selected.



IBM Software Development Platform

- Shared Resources Directory: C:\Program Files\IBM\SDP70Shared
- Installation Directory: C:\Program Files\IBM\SDP70
- Eclipse IDE: C:\Program Files\IBM\SDP70

Installed Packages and Fixes

- IBM® Rational® Application Developer 7.0.0.2 (Internal Version 7.0.0.20070401_2027)
- IBM Rational Application Developer, Version 7.0.0.2 Interim Fix 001 (Internal Version 7.0.0.2_20070418_1400)

Figure A-17 Update Packages

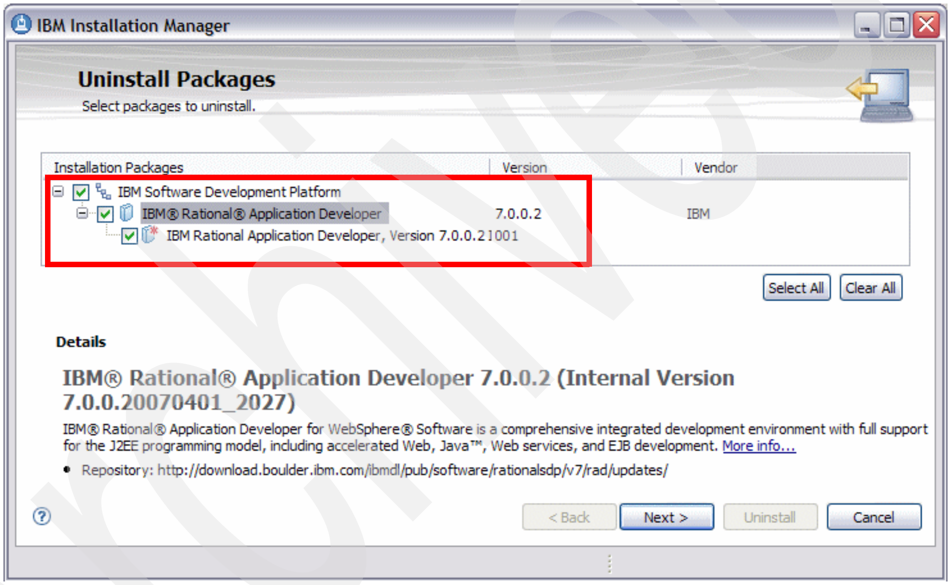
Uninstalling Rational Application Developer

Application Developer V7.0 can be uninstalled interactively through the IBM Installation Manager.

Before uninstallation of any products, ensure to terminate the programs that you installed using Installation Manager.

In the Installation Manager overview select **Uninstall Packages** (see Figure A-1 on page 1282).

In the Uninstall Packages page select the Rational Application Developer product package that you want to uninstall (Figure A-18). Click **Next**.



☑ IBM Software Development Platform
☑ IBM® Rational® Application Developer 7.0.0.2 IBM
☑ IBM Rational Application Developer, Version 7.0.0.2 1001

IBM® Rational® Application Developer 7.0.0.2 (Internal Version 7.0.0.20070401_2027)

IBM® Rational® Application Developer for WebSphere® Software is a comprehensive integrated development environment with full support for the J2EE programming model, including accelerated Web, Java™, Web services, and EJB development. More info...

- Repository: http://download.boulder.ibm.com/ibmdl/pub/software/rationalsdp/v7/rad/updates/

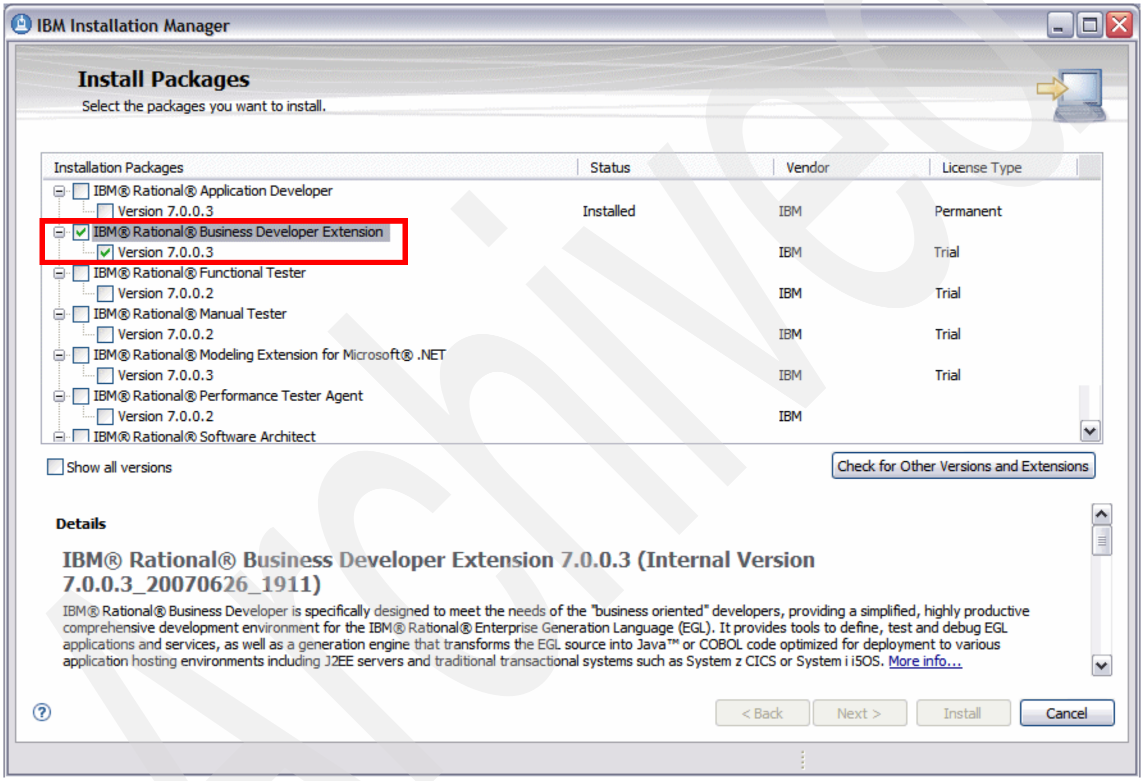
Figure A-18 Uninstall Application Developer

In the Summary page review the list of packages that will be uninstalled and then click **Uninstall**. The Complete page is displayed after the uninstallation finishes. Click **Finish** to exit the wizard.

Installing Enterprise Generation Language

Enterprise Generation Language (EGL) is available as a package called IBM Rational Business Developer Extension that is installed using the IBM Installation Manager:

- Start IBM Installation Manager.
- Click **Install Packages**.
- Select **IBM Rational Business Developer Extension** (Figure A-19).



Install Packages

Select the packages you want to install.

Installation Packages	Status	Vendor	License Type
⊟ □ IBM® Rational® Application Developer			
□ Version 7.0.0.3	Installed	IBM	Permanent
⊟ ☑ IBM® Rational® Business Developer Extension			
☑ Version 7.0.0.3		IBM	Trial
⊟ □ IBM® Rational® Functional Tester			
□ Version 7.0.0.2		IBM	Trial
⊟ □ IBM® Rational® Manual Tester			
□ Version 7.0.0.2		IBM	Trial
⊟ □ IBM® Rational® Modeling Extension for Microsoft® .NET			
□ Version 7.0.0.3		IBM	Trial
⊟ □ IBM® Rational® Performance Tester Agent			
□ Version 7.0.0.2		IBM	
⊟ □ IBM® Rational® Software Architect			

□ Show all versions

Details

IBM® Rational® Business Developer Extension 7.0.0.3 (Internal Version 7.0.0.3_20070626_1911)

IBM® Rational® Business Developer is specifically designed to meet the needs of the "business oriented" developers, providing a simplified, highly productive comprehensive development environment for the IBM® Rational® Enterprise Generation Language (EGL). It provides tools to define, test and debug EGL applications and services, as well as a generation engine that transforms the EGL source into Java™ or COBOL code optimized for deployment to various application hosting environments including J2EE servers and traditional transactional systems such as System z CICS or System i i5OS. More info...

Figure A-19   Installing IBM Rational Business Developer Extension

- Click **Next** and follow the instructions to install the package (similar to install Application Developer).
- The package is installed into the same Eclipse shell as Application Developer and its functions will be available inside Application Developer.

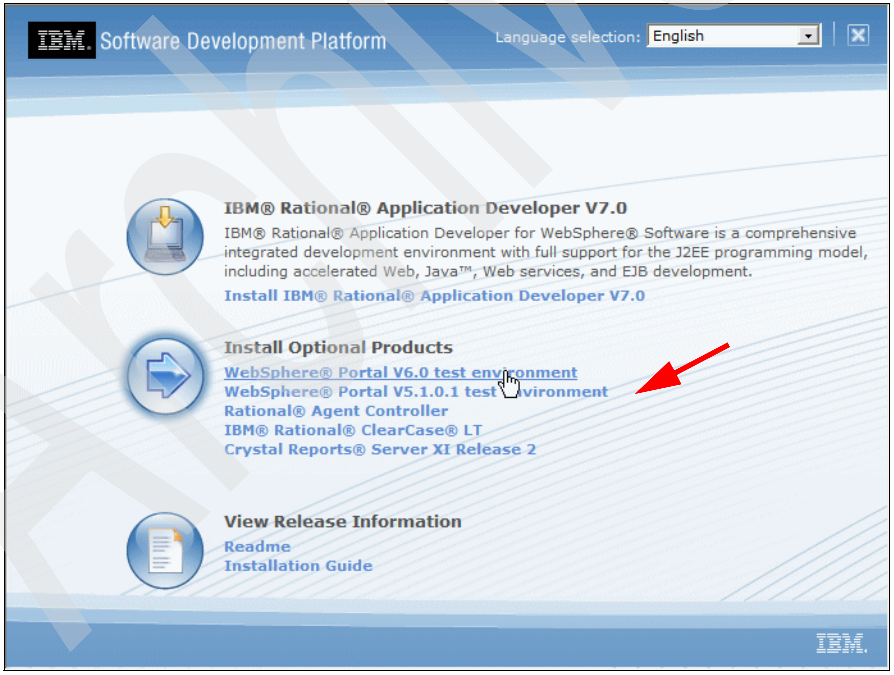
Installing the WebSphere Portal V6.0 test environment

In this section we describe how to install WebSphere Portal V6, add it to Application Developer V7, and configure the portal test environment for performance.

Installing WebSphere Portal V6.0

The installation of WebSphere Portal V6.0 as a test environment is a more simpler and straight forward exercise in Application Developer V7.0. If you have already installed Portal Server V6.0 on the same machine where you are about to install Rational Application Developer, the installation wizard of Application Developer automatically integrates the installed Portal Server as a target runtime. This is the same for WebSphere Application Server V6.0 and V6.1.

Most of you are installing the Portal Test Environment after you have installed Application Developer. The best way to do this is to bring up the Launchpad for Rational Application Developer V7 (Figure A-20).



IBM. Software Development Platform

Language selection: English

IBM® Rational® Application Developer V7.0

IBM® Rational® Application Developer for WebSphere® Software is a comprehensive integrated development environment with full support for the J2EE programming model, including accelerated Web, Java™, Web services, and EJB development.

Install IBM® Rational® Application Developer V7.0

Install Optional Products

WebSphere® Portal V6.0 test environment
WebSphere® Portal V5.1.0.1 test environment
Rational® Agent Controller
IBM® Rational® ClearCase® LT
Crystal Reports® Server XI Release 2

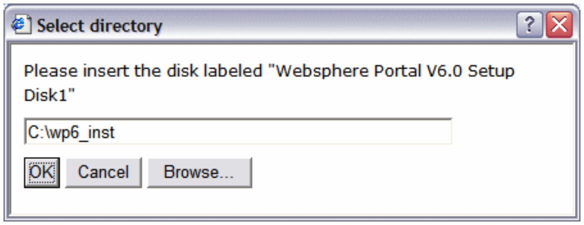
View Release Information

Readme
Installation Guide

Figure A-20 Product installation launchpad

- In the Launchpad dialog, click **WebSphere Portal V6.0 test environment**.

- A dialog prompts the user to insert the Disk 1, which is the setup disk for WebSphere Portal V6.0. You can either do this or extract all the downloaded zip files for Application Developer V7.0 into a temporary directory and specify the path of that folder in this dialog (Figure A-21).

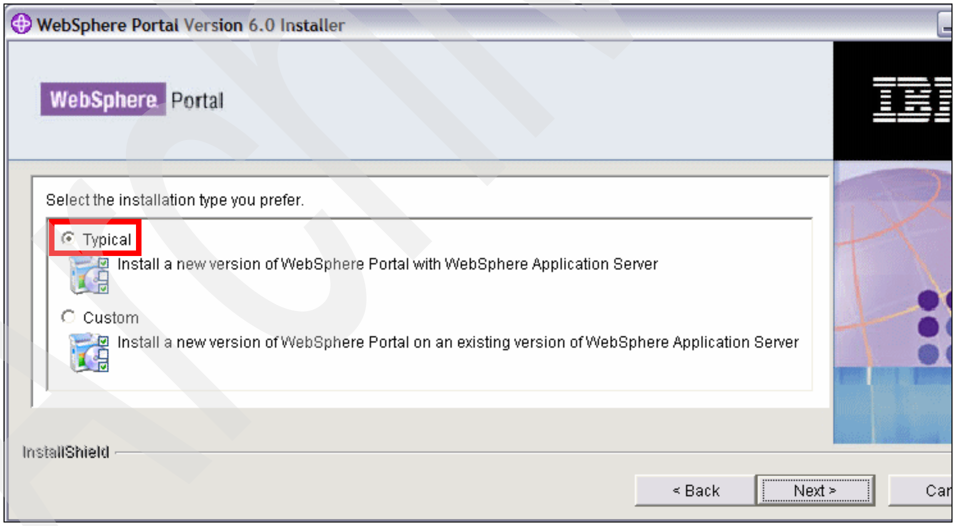


Please insert the disk labeled "Websphere Portal V6.0 Setup Disk1"

C:\wp6_inst

Figure A-21 Location of the install image of WebSphere Portal Setup Disk 1

- Click **OK** to initiate the InstallShield wizard for WebSphere Portal V6.0. Select the language when prompted. Click **Next** on the Welcome dialog to continue.
- Click **Next** on the Software license agreement (if you agree).
- In the next page, select the **Typical** as the Installation type and click **Next (**Figure A-22). The **Custom** option is chosen when the user wants to install the Portal Server on an existing version of the Application Server.



Select the installation type you prefer.

● Typical

Install a new version of WebSphere Portal with WebSphere Application Server

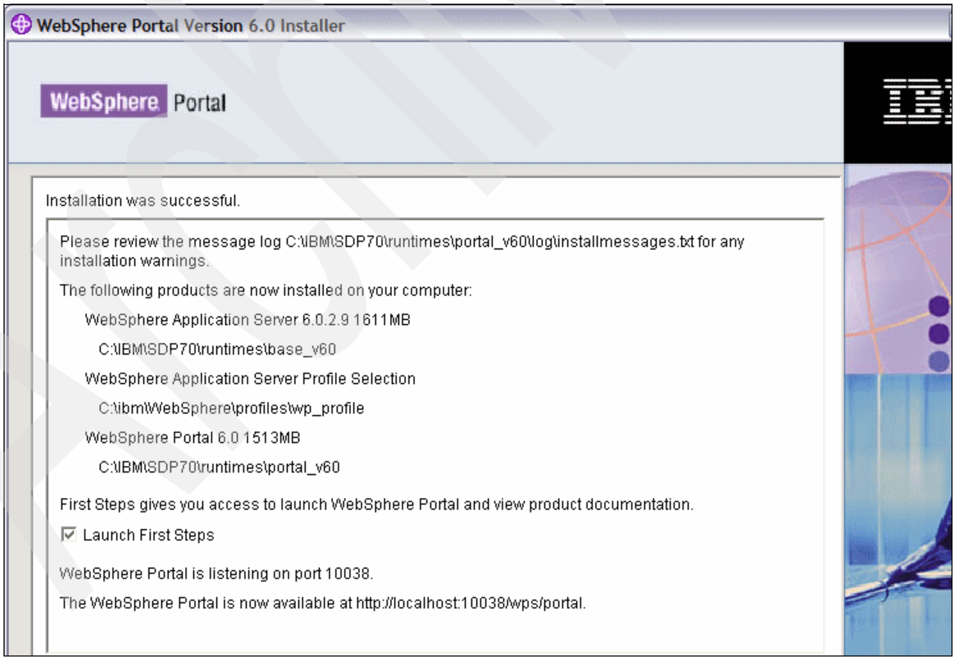
○ Custom

Install a new version of WebSphere Portal on an existing version of WebSphere Application Server

Figure A-22 Installing the WebSphere Portal V6.0 Test Environment

- In the next page of the installation wizard (Specify the install location of the Application Server), accept the default, and click **Next**.
- In the next page (Specify the Cellname, Nodename, and the fully qualified hostname of your machine), accept the default, and click **Next**.

- In the next page, enter the userid and password for the Application Server administrator, and click **Next**.
- In the next page, select whether you want Business Process support in your Portal tooling. We select **No** because this is not in the scope of this book. Click **Next**.
- In the next page (Specify the install location of the Portal Server), accept the default and click **Next**.
- In the next page, enter the userid and password for the Portal Server administrator user, and click **Next**.
- In the next page, clear the option to configure two Windows services to start and stop the Application and Portal Server. Click **Next**.
- In the next page, the wizard displays the summary screen. Read and verify, and if everything is as you desired, click **Next** to start the install process. This can take a long time, and the amount of time the installation process will take depends on your machine and its resources. It took us about 2 hours on a Pentium 4™ desktop with 2 GB of memory.
- When the wizard has completed installing the product, it displays the final summary dialog (Figure A-23). Click **Finish**.



Installation was successful.

Please review the message log C:\IBM\SDP70\runtimes\portal_v60\log\installmessages.txt for any installation warnings.

The following products are now installed on your computer:

WebSphere Application Server 6.0.2.9 1611MB

C:\IBM\SDP70\runtimes\base_v60

WebSphere Application Server Profile Selection

C:\ibm\WebSphere\profiles\wp_profile

WebSphere Portal 6.0 1513MB

C:\IBM\SDP70\runtimes\portal_v60

First Steps gives you access to launch WebSphere Portal and view product documentation.

☑ Launch First Steps

WebSphere Portal is listening on port 10038.

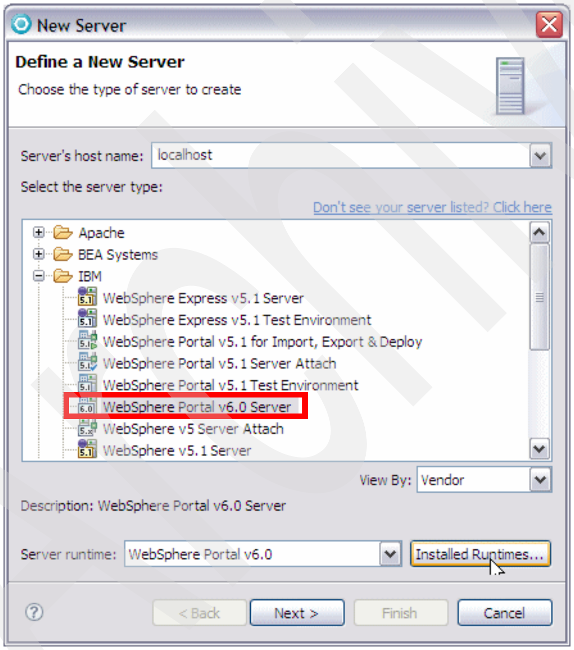
The WebSphere Portal is now available at http://localhost:10038/wps/portal.

Figure A-23 WebSphere Portal v6.0 installation successful

Adding WebSphere Portal V6.0 to Application Developer

To execute the portal and portlet applications, we have to create a new server in Application Developer for the installed Portal V6.0 Server as the target runtime.

- Start Application Developer.
- Before you define a new Portal V6.0 Server in Application Developer, there are couple of things you might want to do:
  - Start with a brand new workspace.
  - Delete the WebSphere Application Server V6.1 from the Servers view and restart the Workbench. The presence of Application Server V6.1 and Portal V6.0 Server causes an issue where the Application Developer cannot sense the proper state of Portal Server V6.x.
- In the Servers view, right- click and select **New** → **Server** to start the New Server dialog (Figure A-24).



Define a New Server

Choose the type of server to create

Server's host name: localhost

Select the server type:

Don't see your server listed? Click here
Apache
BEA Systems
IBM
WebSphere Express v5.1 Server
WebSphere Express v5.1 Test Environment
WebSphere Portal v5.1 for Import, Export & Deploy
WebSphere Portal v5.1 Server Attach
WebSphere Portal v5.1 Test Environment
WebSphere Portal v6.0 Server
WebSphere v5 Server Attach
WebSphere v5.1 Server

View By: Vendor

Description: WebSphere Portal v6.0 Server

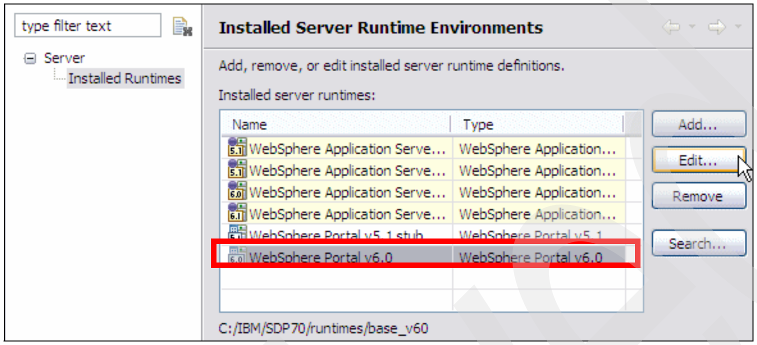
Server runtime: WebSphere Portal v6.0

Installed Runtimes...
< Back
Next >
Finish
Cancel

Figure A-24 Define a New Portal V6.0 Server

- In the New Server dialog, select **WebSphere Portal V6.0 Server** as the server type and click **Installed Runtimes** to verify if the install wizard has configured the newly installed Portal Server correctly.

- The next page displays the list of installed server runtime environments (Figure A-25). You should see an entry for WebSphere Portal v6.0. You can also bring up this dialog by clicking **Window** → **Preferences**, expand **Server**, and select **Installed Runtimes**.
- Select WebSphere Portal V6.0 and click **Edit**.



type filter text

Server
Installed Runtimes

Installed Server Runtime Environments

Add, remove, or edit installed server runtime definitions.

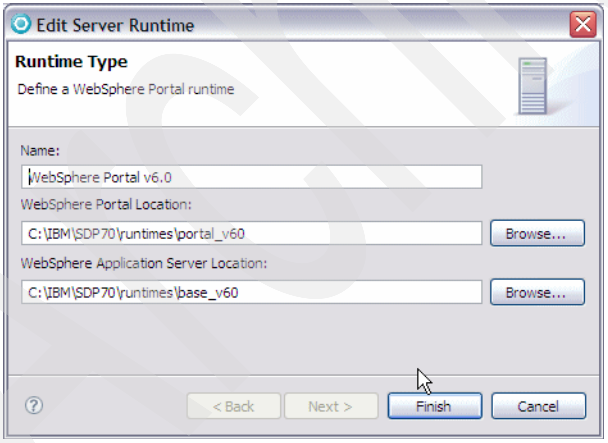
Installed server runtimes:

Name	Type
WebSphere Application Serve...	WebSphere Application...
WebSphere Application Serve...	WebSphere Application...
WebSphere Application Serve...	WebSphere Application...
WebSphere Application Serve...	WebSphere Application...
WebSphere Portal v5.1 stub	WebSphere Portal v5.1
WebSphere Portal v6.0	WebSphere Portal v6.0

C:/IBM/SDP70/runtimes/base_v60

Figure A-25 List of Installed Server Runtime Environments

- In the Edit Server Runtime dialog (Figure A-26), verify that the install path locations of Portal and Application Servers are correct. Click **Finish** to close this dialog, and also close the Installed Server Runtime Environments dialog.



Edit Server Runtime

Runtime Type

Define a WebSphere Portal runtime

Name: WebSphere Portal v6.0

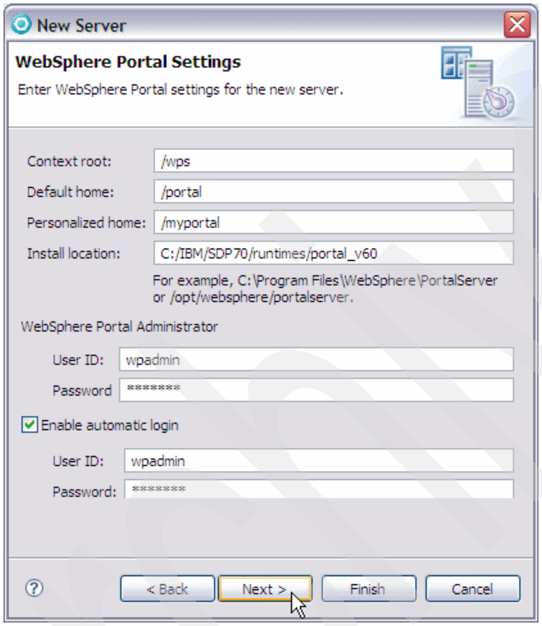
WebSphere Portal Location: C:\IBM\SDP70\runtimes\portal_v60

WebSphere Application Server Location: C:\IBM\SDP70\runtimes\base_v60

Figure A-26 Portal and Application Server locations

- In the New Server dialog, click **Next**.
- In the WebSphere Settings dialog, verify that **Soap** is selected as the server connection type (and not RMI), and also verify that **admin port** for this connection is the correct Soap port for the Application server. The default value for this port is 10033.

- In the same dialog, enter the userid and password for the Application Server administrator. In our installation, the user ID is `wpsbind` and the password is `wpsbind`. Click **Next**.
- In the WebSphere Portal Settings dialog, verify the portal settings, such as context root, default, and personalized home, and the install location of Portal Server. In the same dialog, enter the userid and password for Portal Server administrator. In our installation the user ID is `wpadmin` and the password is `wpadmin`. Decide if you want to enable the automatic login of a particular user when the Portal test environment starts. Click **Next** (Figure A-27).


New Server

WebSphere Portal Settings

Enter WebSphere Portal settings for the new server.

Context root: /wps

Default home: /portal

Personalized home: /myportal

Install location: C:/IBM/SDP70/runtimes/portal_v60

For example, C:\Program Files\WebSphere\PortalServer or /opt/websphere/portalserver.

WebSphere Portal Administrator

User ID: wpadmin

Password ********

☑ Enable automatic login

User ID: wpadmin

Password: ********

Figure A-27 WebSphere Portal Settings

- In the Properties Publishing Properties dialog, select the default value of **Local Copy** for the transfer method. Click **Next**.
- In the Add and Remove Projects dialog, click **Next** (you do not have any portal or portlet projects to add to this server).
- In the Tasks dialog, click **Finish**. This should install a new test server. You can verify this in the Servers view that displays a new entry for WebSphere Portal V6.0 Server.
- You can now go ahead and right-click the Portal Server and select **Publish**, which starts the server, and change its status to Started and its state to Synchronized. Note that it takes a while to start the Portal Server.

- To verify that the server is running, right-click on the portal server entry in the Server view, and select **Run administrative console**. This opens a browser session with the WebSphere Application Server console. The important thing to note here is that we do not have to start the underlying application server to be able to use the Application Server console.

Upgrading the Portal Server runtime to Version 6.0.1

The upgrade of Portal Server V6.0 to version 6.0.1 (or 6.0.1.1) is a multi-step process and involves upgrading the Application Server and upgrading the Portal Server:

- **Documentation**—The latest upgrade for WebSphere Portal Version 6.0.1.1 is documented in the InfoCenter at:

  `http://publib.boulder.ibm.com/infocenter/wpdoc/v6r0/index.jsp?topic=/com.ibm.wp.ent.doc/wpf/pui_intro6011.html`

- **Preparation**—Follow the detailed instructions to prepare your environment and download the Fixpack from:

  `http://www-1.ibm.com/support/docview.wss?rs=688&uid=swg24016161`

- **Installation**—Install the Fix Pack by running the **`updatePortalWizard`** command (follow the instructions in the GUI):

  ```
  <Portal_Home>\update\updatePortalWizard
  ```

- **Upgrade Portal Server to V6.0.1.1**—Some of the updates of the user interface JSPs are placed into `<Portal_Home>/fixes` and must be manually merged into the `wps.ear` file. This is done because you might have tailored those pages.

The upgrade process is quite complex and you have to follow the instructions very carefully.

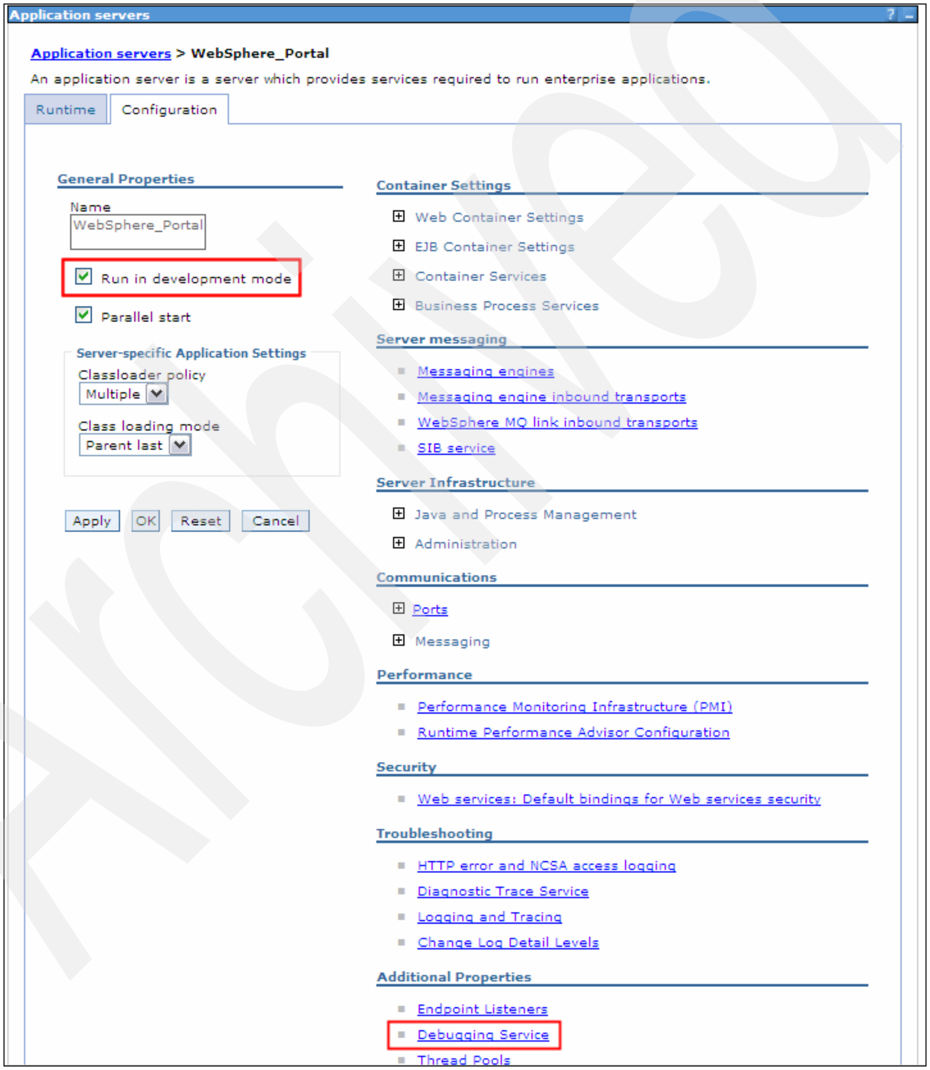
Optimizing the Portal Server for development

Starting with WebSphere Portal V6.0 and Application Developer V7.0, the Portal Server runtime is a complete Portal Server and not just an integrated test environment or some sort of plug-in for the Application Developer as it was with the previous versions of Portal Server and Application Developer. Though this is good news, there are certain optional things that you might want to do to optimize your Portal Server so it performs more effectively in the development mode when you are using it for testing and debugging.

Enable development mode

Do these steps:

- Start the Portal server.
- Open the administrative console. Enter user ID and password (`wpsbind`).
- Navigate to **Server** → **Application Servers** → **WebSphere_Portal** and click **Run in development mode** (Figure A-28). Click **Apply**.
- Click **Save** to apply changes to the master configuration.



Application servers

Application servers > WebSphere_Portal

An application server is a server which provides services required to run enterprise applications.

Runtime | Configuration

General Properties

Name
WebSphere_Portal

☑ Run in development mode

☑ Parallel start

Server-specific Application Settings

Classloader policy
Multiple

Class loading mode
Parent last

Container Settings

⊞ Web Container Settings
⊞ EJB Container Settings
⊞ Container Services
⊞ Business Process Services

Server messaging

- Messaging engines
- Messaging engine inbound transports
- WebSphere MQ link inbound transports
- SIB service

Server Infrastructure

⊞ Java and Process Management
⊞ Administration

Communications

⊞ Ports
⊞ Messaging

Performance

- Performance Monitoring Infrastructure (PMI)
- Runtime Performance Advisor Configuration

Security

- Web services: Default bindings for Web services security

Troubleshooting

- HTTP error and NCSA access logging
- Diagnostic Trace Service
- Logging and Tracing
- Change Log Detail Levels

Additional Properties

- Endpoint Listeners
- Debugging Service
- Thread Pools

Figure A-28 Enable Development mode for Portal Server

Enable debugging service

If you are using a remote Portal Server without using the Rational tools and you intend to use this Portal Server for debugging purposes, then you might want to enable the debugging service for this server during its startup process.

This can be done by selecting **Debugging Service** (Figure A-28), then:

- Click **Enable service at server startup** and click **Apply**.
- Click **Save** to apply changes to the master configuration**.**

Besides these basic changes, there are more tips that you can implement to increase the performance of the Portal Server during the development mode or to reduce its startup time. Refer to “More information” on page 960.

Installing IBM Rational ClearCase LT

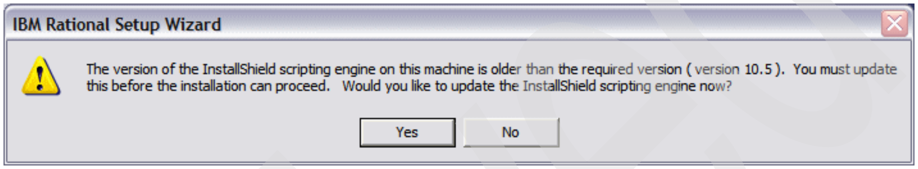
Application Developer entitles you to a free license of Rational ClearCase LT. Installing the ClearCase LT Server component also installs the ClearCase LT Client component. We recommend that you install the ClearCase LT Server component before installing ClearCase LT Client on any additional machines.

> **Tips:** When installing the ClearCase LT Server, you can be logged on either locally on your Windows machine or logged on to a Windows domain. If installing while logged on locally, you will only be able to connect to the server from your local machine. Other people in your development team will not be able to connect to your machine and use your ClearCase LT Server. The user account used when installing must be a member of the local Administrators group.
>
> To use ClearCase LT in a team environment and let other team members use your ClearCase LT Server, you must be logged on to a Windows domain with a user account having Domain Administrator privileges while installing the ClearCase LT Server. The domain must also have a group for the ClearCase users, and all members of your development team must be members of this group. This group should also be the Primary Group for these users. You can use the Domain Users group for this.
>
> We highly recommend that you use the Windows domain approach. Local setup can be useful for testing and demonstration purposes.

The installation instructions in this section are intended to help you install the client and server code for Rational ClearCase LT. For more detailed installation instructions, refer to the *Rational ClearCase LT Installation Guide*.

- Run **setup.exe** from the root directory of the downloaded ClearCase LT installation image.
- When the LaunchPad comes up, select **Install IBM Rational ClearCase LT**.

> **Note:** If the IBM Rational Setup Wizard warning dialog opens, as shown in Figure A-29, just click **Yes**.



IBM Rational Setup Wizard

The version of the InstallShield scripting engine on this machine is older than the required version (version 10.5). You must update this before the installation can proceed. Would you like to update the InstallShield scripting engine now?

Figure A-29 IBM Rational Setup Wizard warning dialog

- In the IBM Rational Setup Wizard, click **Next**.
- In the Deployment Method dialog, select **Desktop installation from CD image** and click **Next**.
- In the Client/Server dialog, select **Install server and client software** [default] and click **Next***.*
- In the Welcome dialog, click **Next***.*
- A warning message *STOP! Before proceeding with this install, please close all applications and disable anti-virus software* is displayed. Follow the instructions and click **Next**.
- In the Software License Agreement window, click **Accept**.
- In the Destination Folder dialog, verify the installation path (we use the default of `C:\Program Files\Rational\`). Click **Next***.*
- In the Custom Setup dialog, you can unselect the Web Interface, because it is not used in the samples of this Redbooks publication. Click **Next**.
- In the Configure IBM Rational ClearCase LT dialog, click **Done** to exit.
- In the Ready to Install the Program dialog, click **Install**.
- In the Setup Complete dialog, clear the two **Take me to** selections and click **Finish**.
- In the IBM Rational ClearCase LT Installer Information warning, click **Yes** to restart your system.

Note: After you have installed the Rational ClearCase LT product, we recommend that you review the Rational ClearCase Support page on the IBM Software Support site and make sure that the latest fixes have been applied. The Web site can be found at:

```
http://www.ibm.com/software/awdtools/clearcase/support
```

ClearCase LT uses a feature called Unified Change Management (UCM) by default out of the box. This configuration requires a process VOB first be created. Before this, we have to create a storage location for the VOBs and views.

To create storage locations for VOBs and views and a process VOB, do these steps:

- Create the following directories on the machine you have installed ClearCase LT server:

```
c:\clearstorage\vobs
c:\clearstorage\views
```

- Share the directory `c:\clearstorage` and give permission **Full Control** to **Everyone**.
- Open the Windows Command Prompt and enter the following commands:

```
cd \
cleartool mkstgloc -vob vobs \\<servername>\clearstorage\vobs\
cleartool mkstgloc -view views \\<servername>\clearstorage\views\
```

  – The *vobs* and *views* are the names of the VOB and view storage location and how ClearCase knows how to identify them. The pathname to the directory must be given using a UNC pathname.

  – So `\\server\clearstorage\` would be the server hostname followed by the shared directory that we created.

  Once this storage location exists, you can create the process VOB:

```
cleartool mkvob -tag \process -ucmproject -stgloc vobs
```

  – You will then be prompted for comments, you can type your creation comments and then enter a period (.) on an empty line to proceed when the comments are finished.

B

Additional material

The additional material is a Web download of the sample code for this book. This appendix describes how to download, unpack, describe the contents, and import the project interchange file. In some cases the chapters also require database setup; however, if needed, the instructions will be provided in the chapter in which they are needed.

The appendix is organized into the following sections:

- Locating the Web material
- Unpacking the sample code
- Description of the sample code
- Setting up the ITSOBANK database
- Configuring the data source in WebSphere Application Server
- Importing sample code from a project interchange file

Locating the Web material

The Web material associated with this book is available in softcopy on the Internet from the IBM Redbooks Web server. Enter the following URL in a Web browser, and then download the two ZIP files from:

`ftp://www.redbooks.ibm.com/redbooks/SG247501`

Alternatively, you can go to the IBM Redbooks Web site at:

`http://www.ibm.com/redbooks`

Select the **Additional materials** and open the directory that corresponds with the Redbooks publication form number, SG24-7501.

The additional Web material that accompanies this Redbooks publication includes the following files:

File name	*Description*
7501code.zip	Zip file containing sample code
7501codesolution.zip	Zip file containing solution interchange files

System requirements for downloading the Web material

The following system configuration is recommended:

Hard disk space:	20 GB minimum
Operating System:	Windows or Linux
Processor:	2 GHz
Memory:	2 GB

Using the sample code

In this section we provide a description of the sample code and how to use it.

Unpacking the sample code

After you have downloaded the two ZIP files, unpack the files to your local file system using WinZip, PKZip, or similar software. For example, we unpacked the `7501code.zip` and `7501codesolution.zip` files to the `c:\7501code` directory. Throughout the samples, we reference the sample code as if you have already unpacked the files to the C drive.

Description of the sample code

Table B-1 describes the contents of the sample code after unpacking. The `7501code` folder has two major sections:

- Code sample to follow the instructions in a chapter
- Interchange files with the solution of each chapter

Table B-1 Sample code description

Directory	Sample code for which chapter
c:\7501code	Root directory after unpacking the sample code
..\java	Chapter 7, “Develop Java applications” on page 227
..\patterns	Chapter 8, “Accelerate development using patterns” on page 325
..\database	Chapter 9, “Develop database applications” on page 355 Also includes the code to setup the `ITSOBANK` database in either Derby or DB2
..\gui	Chapter 10, “Develop GUI applications” on page 403
..\xml	Chapter 11, “Develop XML applications” on page 429
..\webapps	Chapter 12, “Develop Web applications using JSPs and servlets” on page 465
..\struts	Chapter 13, “Develop Web applications using Struts” on page 541
..\jsfsdo	Chapter 14, “Develop Web applications using JSF and SDO” on page 587
..\egl	Chapter 15, “Develop applications using EGL” on page 665
..\ejb	Chapter 16, “Develop Web applications using EJBs” on page 719
..\j2eeclient	Chapter 17, “Develop J2EE application clients” on page 793
..\webservices	Chapter 18, “Develop Web services applications” on page 811
..\portal	Chapter 19, “Develop portal applications” on page 897
..\junit	Chapter 21, “Test using JUnit” on page 1001
..\debug	Chapter 22, “Debug local and remote applications” on page 1041
..\ant	Chapter 23, “Build applications with Ant” on page 1075
..\jython	Chapter 24, “Deploy enterprise applications” on page 1103
..\zInterchangeFiles	Directory with subdirectories for each chapter with an interchange file containig the finished code

Interchange files with solutions

The directory C:\7501code\zInterchangeFiles contains the finished applications for most chapters:

```
..\java\RAD7Javaa.zip RAD7JavaImport.zip
..\patterns\RAD7Patterns.zip RAD7FacadePatternJava.zip
..\gui\RAD7Gui.zip
..\xml\RAD7XML.zip
..\webapps\RAD7BankBasic.zip
..\struts\RAD7Struts.zip
..\jsfsdo\RAD7JSF.zip
..\egl\RAD7EGL.zip
..\ejb\RAD7EJB.zip RAD7EJBonly.zip RAD7EJBWeb.zip RAD7EJBdb2.zip
..\j2eeclient\RAD7AppClient.zip
..\webservices\RAD7WebService.zip RAD7WSFP.zip
..\portal\FacesPortletSDOExample.zip WebServicesPortlets.zip
..\junit\RAD7JUnit.zip RAD7JUnitWebTest.zip
..\ant\RAD7Ant.zip
..\jython\RAD7Jython.zip
```

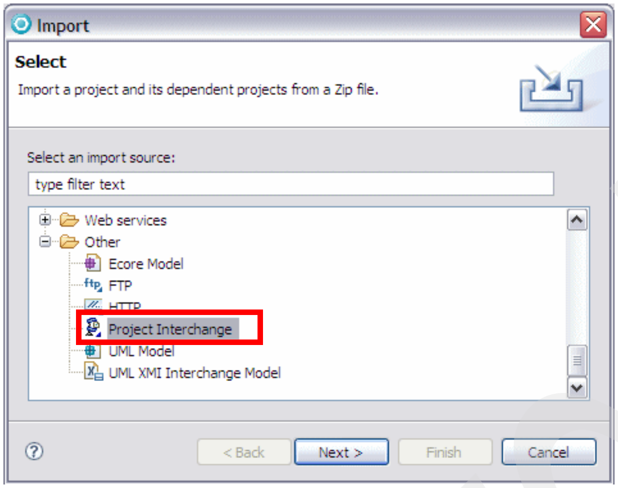
Importing sample code from a project interchange file

This section describes how to import the Redbooks publication sample code project interchange zip files into Application Developer. This section applies for each of the chapters containing sample code that have been packaged as a project interchange zip file.

Tip: After importing an interchange file for the RAD7EJB project, you have to deploy the code again. Right-click the **RAD7EJB** project and select **Prepare for Deployment**.

To import a project interchange file, do these steps:

- From the Workbench, select **File** → **Import**.
- From the Import dialog, select **Project Interchange**, and then click **Next (**Figure B-1).


Import

Select

Import a project and its dependent projects from a Zip file.

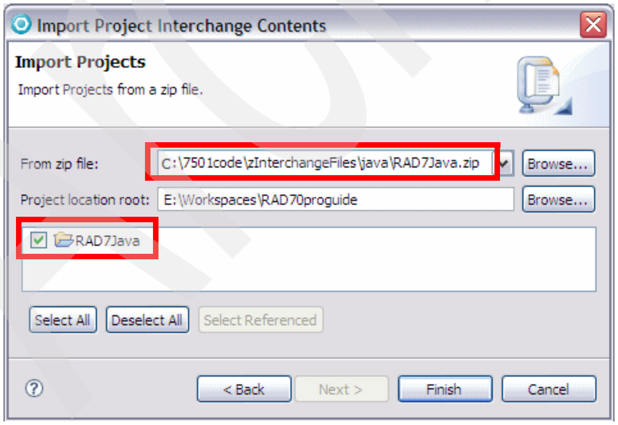
Select an import source:

type filter text

Figure B-1 Import a project interchange file

- When prompted for the path and file name, enter the following:
  - From zip file: Click **Browse** and locate the path and the zip file (for example, `C:\7501code\zInterchangeFiles\java\RAD7Java.zip`).
  - Project location root: Leave the default workspace location.
- After locating the zip file, the projects contained in the interchange file are listed. Select the project(s) that you want to import, and click **Finish**.

  For example, we select **RAD7Java** and clicked **Finish**, (Figure B-2).


Import Project Interchange Contents

Import Projects

Import Projects from a zip file.

From zip file: C:\7501code\zInterchangeFiles\java\RAD7Java.zip

Project location root: E:\Workspaces\RAD70proguide

☑ RAD7Java

Figure B-2 Import projects for an interchange file

Setting up the ITSOBANK database

We provide two implementations of the `ITSOBANK` database, Derby and DB2. You can choose to implement either or both databases and then set up the enterprise applications to use one of the databases. The Derby database is shipped with WebSphere Application Server v6.1. We tested the application on DB2 Version 8.2, but it should work on DB2 v7.2, v8.1, and v9.1 as well.

Derby

Command files to define and load the `ITSOBANK` database in Derby are provided in the `C:\7501code\database\derby` folder:

- **DerbyCreate.bat**, **DerbyLoad.bat** and **DerbyList.bat** assume that you installed Application Developer in **C:\IBM\SDP70\** folder. You have to edit these files to point to your Application Developer installation directory if you installed the product in a different folder.
- In the **C:\7501code\database** directory:
  - Execute the **DerbyCreate.bat** file to create the database and table.
  - Execute the **DerbyLoad.bat** file to delete the existing data and add records.
  - Execute the **DerbyList.bat** file to list the contents of the database.

These command files use the SQL statements and helper files provided in:

- `itsobank.ddl`—Database and table definition
- `itsobank.sql`—SQL statements to load sample data
- `itsobanklist.sql`—SQL statement to list the sample data
- `tables.bat`—Command file to execute `itsobank.ddl` statements
- `load.bat`—Command file to execute `itsobank.sql` statements
- `list.bat`—Command file to execute `itsobanklist.sql` statements

The Derby itsobank database is created under:

```
C:\7501code\database\derby\ITSOBANK
```

DB2

DB2 command files to define and load the `ITSOBANK` database are provided in **C:\7501code\database\db2** folder:

- Execute the **createbank.bat file** to define the database and table.

- Execute the **loadbank.bat** file to delete the existing data and add records.
- Execute the **listbank.bat** file to list the contents of the database.

These command files use the SQL statements provided in:

- `itsobank.ddl`—Database and table definition
- `itsobank.sql`—SQL statements to load sample data
- `itsobanklist.sql`—SQL statement to list the sample data

Configuring the data source in WebSphere Application Server

This section shows how to configure the data source in the WebSphere Administrative Console. We configure the data source against the WebSphere Application Server v6.1 test environment shipped with Application Developer. The integrated WebSphere Application Server v6.1 test environment install is a very thin layer on top of a true stand-alone WebSphere Application Server install. In other words, it can be considered as a stand-alone WebSphere Application Server v6.1.

Here are the high-level configuration steps to configure the data source within WebSphere Application Server for the ITSO Bank application sample:

- Starting the WebSphere Application Server
- Configuring the environment variables
- Configuring J2C authentication data
- Configuring the JDBC provider
- Creating the data source

Starting the WebSphere Application Server

If you are using a stand-alone WebSphere Application Server v6.1, enter the following commands in a command window:

```
cd \Program Files\IBM\WebSphere\AppServer\profiles\AppSrv01\bin
startServer.bat server1
```

If you are using the WebSphere Application Server v6.1 test environment shipped with Application Developer, in the Servers view, right-click **WebSphere Application Server v6.1** and select **Start**.

Configuring the environment variables

Prior to configuring the data source, ensure that the environment variables are defined for the desired database server type. This step does not apply to Derby because we are using the embedded Derby, which already has the variables defined. For example, if you choose to use DB2 Universal Database, you must verify the path of the driver for DB2 Universal Database.

- Launch the WebSphere Administrative Console:
  - Enter the following URL in a Web browser:

    `http://<hostname>:9060/ibm/console`

    You may use a different port other than 9060. The administrative console port number was chosen during the installation of the server profile.
  - If you are using the WebSphere Application Server v6.1 test environment shipped with Application Developer, you can simply right-click **WebSphere Application Server v6.1** and select **Run administrative console**.
- Click **Log in**. The user ID can be anything at this point, since WebSphere security is not enabled.
- Expand **Environment** → **WebSphere Variables**.
- Scroll down the page and click the desired variable and update the path accordingly for your installation.

  For Derby:

  - `DERBY_JDBC_DRIVER_PATH`—By default this variable is already configured because Derby is installed with WebSphere Application Server.

  For DB2:

  - `DB2UNIVERSAL_JDBC_DRIVER_PATH`
  - `UNIVERSAL_JDBC_DRIVER_PATH`
  - `DB2UNIVERSAL_JDBC_DRIVER_NATIVEPATH`

    For example, if you decide to use DB2 Universal Database, select each of the above three variables and enter the path value, for example, `C:\Program Files\IBM\SQLLIB\java`. Click **OK**.
- Click **Save**.

Configuring J2C authentication data

This section describes how to configure the J2C authentication data (database login and password) for WebSphere Application Server from the WebSphere Administrative Console. This step is required for DB2 UDB and optional for Derby.

If using DB2 UDB, configure the J2C authentication data (database login and password) for WebSphere Application Server from the Administrative Console:

- Select **Security** → **Secure administration, applications, and infrastructure**.
- Under the Authentication properties, expand **Java Authentication and Authorization Service** → select **J2C Authentication data**.
- Click **New**.
- Enter the Alias, User ID, and Password in the JAAS J2C Authentication data page if you are using DB2 UDB and then click **OK**. For example, create an alias called **db2user** and enter the user ID and password used when installing DB2.
- Click **Save** and then click **Save**.

Configuring the JDBC provider

This section describes how to configure the JDBC provider for the selected database type. The following procedure demonstrates how to configure the JDBC provider for Derby, with notes on how to do the equivalent for DB2 UDB.

To configure the JDBC provider from the WebSphere Administrative Console, do these steps:

- Select **Resources** → **JDBC** → **JDBC Providers**.
- Select the scope settings.
  - Select the server scope from the drop-down menu. In our case, we select **Node=henrycuiNode02, Server=server1**.
- Click **New**.
- From the New JDBC Providers page, do these steps and then click **Next**:
  - Select the Database Type: Select **Derby**.

    Note: For DB2 UDB, select **DB2**.
  - Select the JDBC Provider: Select **Derby JDBC Provider**.

    Note: For DB2 UDB, select **DB2 Universal JDBC Driver Provider**.
  - Select the Implementation type: Select **XA data source**.

    Note: For DB2 UDB, select **XA data source**.
- Click **Next**.
- Click **Finish**.

Creating the data source

To create the data source from the WebSphere Administrative Console, do these steps:

- Select **Resources** → **JDBC** → **JDBC Providers** → **Data sources**.
- Select the scope settings.
  - Select the server scope from the drop-down menu, for example, **Node=<userid>Node02, Server=server1**.
- Click **New**.
- From the Enter basic data source information page, do these steps and then click **Next**:
  - Data source name: **RAD7DS**
  - JNDI name: **jdbc/itsobank**

  > **Note:** For DB2 UDB, in the **Component-managed authentication alias and XA recovery authentication alias** section, select your DB2 alias (for example, **db2user**) from the drop-down menu.

- Click **Next**.
- In the select JDBC provider page, select **Derby JDBC Provider (XA)**. Click **Next**.

  Note: For DB2 UDB, select **DB2 Universal JDBC Provider (XA)**.
- Enter **C:\7501code\database\derby\ITSOBANK** as the Database name, click **Next**.

  Note: For DB2 UDB, enter **ITSOBANK** as the Database name and **localhost** as the Server name.
- Click **Finish** and then click **Save**.
- Verify the connection by selecting **RAD7DS** (the check box) and then click **Test connection**. You should get the message:

  The test connection operation for data source RAD7DS on server server1 at node xxxxxNode02 was successful

Abbreviations and acronyms

ACL	access control list
AJAX	Asynchronous JavaScript and XML
API	Application Programming Interface
AST	Application Server Toolkit
ATM	automatic teller machine
AWT	Abstract Window Toolkit
BCI	byte-code instrumentation
BIRT	Business Intelligence and Reporting Tools
BMP	bean-managed persistence
BPEL	Business Process Execution Language
BSF	Bean Scripting Framework
BVT	build verification test
CBE	Common Base Event
CDT	C/C++ Development Tooling
CMP	container managed persistence
CMR	container managed relationship
CORBA	Common Object Request Broker Architecture
CRUD	create, retrieve, update and delete
CSS	cascading style sheet
CSV	comma separated values
CVS	Concurrent Versions System
DADX	document access definition extension
DB	database
DDL	data definition language
DMS	database management system; data mediator services
DOM	domain object model
DTD	document type definition
DTO	data transfer object
EAR	Enterprise Application Archive
EGL	Enterprise Generation Language
EJB	Enterprise JavaBean
EJS	Enterprise Java Server
EL	expression language
EMF	Eclipse Modeling Framework
FK	foreign key
FTP	File Transfer Protocol
FVT	function verification test
GEF	Graphical Editing Framework
GIF	Graphic Interchange File
GMF	Graphical Modeling Framework
GUI	graphical user interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
IBM	International Business Machines
IDE	integrated development environment
ITSO	International Technical Support Organization
J2EE	Java 2 Platform Enterprise Edition
J2SE	Java 2 Platform Standard Edition

JAAS	Java Authentication and Authorization Service
JAF	Java Activation Framework
JAR	Java archive
JAX-B	Java API for XML Binding
JAX-RPC	Java API for XML RPC
JAX-WS	Java API for XML Based Web Services
JAXP	Java API for XML Processing
JAXR	Java API for XML Registries
JCA	J2EE Connector Architecture
JCP	Java Community Process
JDBC	Java DataBase Connectivity
JDK	Java Development Kit
JDT	Java development tools
JET	Java Emitter Templates
JMS	Java Message Service
JMX	Java Management Extensions
JNDI	Java Naming and Directory Interface
JNI	Java Native Interface
JRE	Java Runtime Environment
JSF	JavaServer Faces
JSP	JavaServer Pages
JSR	Java Specification Request
JSTL	JSP Standard Tag Library
JVM	Java Virtual Machine
JVMPI	Java Virtual Machine Profiler Interface
JWL	JSF Widget Library
LDAP	Lightweight Directory Access Protocol
MDB	message-driven bean
MIME	Multipurpose Internet Mail Extensions
MTOM	Message Transmission Optimization Mechanism
MVC	model-view-controller
OASIS	Organization for the Advancement of Structured Information Standards
ODBC	Open DataBase Connectivity
OMG	Object Management Group
OO	object-oriented
ORB	Object Request Broker
PDA	personal digital assistant
PDE	Plug-in Development Environment
PTP	point-to-point
QL	query language
RAD	rapid application development
RAMP	Reliable Asynchronous Messaging Profile
RAR	resource adapter archive
RDB	relational database
RMI	remote method invocation
RUP	Rational Unified Process
SCM	software configuration management
SDK	Software Developer Kit
SDO	Service Data Objects
SDP	Software Delivery Platform
SGML	Standard Generalized Markup Language
SOA	service oriented architecture
SOAP	Simple Object Access Protocol
SPF	Struts Portal Framework
SQL	Structured Query Language
SQLJ	Structured Query Language for Java
SSI	server side include
SSL	secure socket layer
SSN	social security number
SSO	single sign-on

SVG	scalable vector graphics
SVT	system verification test
SWT	Standard Widget Toolkit
TLA	term license agreement
TLD	tag library descriptor
TPTP	Test & Performance Tools Platform
UCM	Unified Change Management
UDB	Universal Database
UDDI	Universal Description, Discovery, and Integration
UI	user interface
UML	Unified Modeling Language
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
UTC	Universal Test Client
VM	virtual machine
VOB	versioned object base
VOIP	voice over IP
WAR	Web Application Archive
WML	Wireless Markup Language
WS-I	Web Services Interoperability Organization
WSDL	Web Service Description Language
WSRP	Web Services for Remote Portlet
WSS	Web services security
WTP	Web Tools Platform
WYSIWYG	what-you-see-is-what-you-get
XML	eXtensible Markup Language
XOP	XML-binary Optimized packaging
XSD	XML Schema Definition
XSL	eXtensible Stylesheet Language
XSLT	XSL transformations

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 publications

For information about ordering these publications, see “How to get Redbooks” on page 1323. Note that some of the documents referenced here may be available in softcopy only.

- *Experience J2EE! Using WebSphere Application Server V6.1*, SG24-7297
- *Building SOA Solutions Using the Rational SDP*, SG24-7356
- *Web Services Handbook for WebSphere Application Server 6.1*, SG24-7257
- *WebSphere Application Server V6.1: Planning and Design*, SG24-7305
- *IBM WebSphere Application Server V6.1 Security Handbook*, SG24-6316
- *WebSphere Application Server V6.1: Systems Management and Configuration*, SG24-7304
- *WebSphere Studio 5.1.2 JavaServer Faces and Service Data Objects*, SG24-6361
- *EJB 2.0 Development with WebSphere Studio Application Developer*, SG24-6819
- *Software Configuration Management: A Clear Case for IBM Rational ClearCase and ClearQuest UCM*, SG24-6399
- *Rational Application Developer V6 Programming Guide*, SG24-6449

Other publications

These publications are also relevant as further information sources:

- *D’Anjou, et al., The Java Developer’s Guide to Eclipse-Second Edition,* Addison Wesley, 2004, ISBN 0-321-30502-7
- Eric Gamma, et al., *Design Patterns, Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995, ISBN 0-201-63361-2

Online resources

These Web sites are also relevant as further information sources:

- IBM Software and IBM Rational:

  http://www.ibm.com/software
  http://www.ibm.com/software/rational/

- IBM Rational Application Developer for WebSphere Software:

  http://www.ibm.com/software/awdtools/developer/application/index.html
  http://www.ibm.com/software/awdtools/developer/application/support/index.html

- IBM developerWorks:

  http://www.ibm.com/developerworks/
  http://www.ibm.com/developerworks/rational/
  http://www.ibm.com/developerworks/rational/products/rad/
  http://www.ibm.com/developerworks/rational/products/rbde/
  http://www.ibm.com/developerworks/rational/products/egl/
  http://www.ibm.com/developerworks/websphere

- The Eclipse Project site, with information on the underlying platform of Rational Application Developer:

http://www.eclipse.org/	
http://www.eclipse.org/projects/	Projects
http://www.eclipse.org/webtools/	Web Tools Platform
http://www.eclipse.org/tptp/	Test & Performance Tools Platform
http://www.eclipse.org/swt/	Standard Widget Toolkit

- The WorldWide Web Consortium:

  http://www.w3c.org/
  http://www.w3c.org/XML/

- Sun Microsystem's Java site, with specifications, tutorials, and best practices:

  http://java.sun.com/
  http://java.sun.com/j2se/
  http://java.sun.com/j2se/1.4.2/docs/guide/jdbc/
  http://java.sun.com/j2se/1.4.2/docs/guide/awt/
  http://java.sun.com/j2se/1.4.2/docs/guide/swing/

- Apache projects:

  http://apache.org/
  http://jakarta.apache.org/
  http://struts.apache.org/
  http://ant.apache.org/
  http://tomcat.apache.org/

- The Java Community Process site, for Java specifications:

  http://www.jcp.org/

- OASIS:

  http://www.oasis-open.org/

- Web Services Interoperability Organization:

  http://www.ws-i.org/

- TheServerSide.com is an enterprise Java site with articles, books, news, and discussions:

  http://www.theserverside.com/

- CVS:

  http://www.nongnu.org/cvs/
  http://www.cvsnt.org
  http://www.march-hare.com/cvspro/

- SourceForge repository for Open Source:

  http://sourceforge.net/
  http://junit.sourceforge.net/

How to get Redbooks

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

IBM

Redbooks

Rational Application Developer V7 Programming Guide

(2.5” spine)
2.5”<->nnn.n”
1315<-> nnnn pages

Rational Application Developer V7 Programming Guide

Develop Java, Web, XML, database, EJB, Struts, JSF, SDO, EGL, Web services, and portal applications

Test, debug, and profile with built-in and remote servers

Deploy applications to WebSphere Application Server and WebSphere Portal

IBM Rational Application Developer for WebSphere Software V7.0 (for short, Rational Application Developer) is the full function Eclipse 3.2 based development platform for developing Java 2 Platform Standard Edition (J2SE) and Java 2 Platform Enterprise Edition (J2EE) applications with a focus on applications to be deployed to IBM WebSphere Application Server and IBM WebSphere Portal. Rational Application Developer provides integrated development tools for all development roles, including Web developers, Java developers, business analysts, architects, and enterprise programmers.

Rational Application Developer is part of the IBM Rational Software Delivery Platform (SDP), which contains products in four life cycle categories: Architecture management (includes integrated development environments and Application Developer), change and release management, process and portfolio management, and quality management.

This IBM Redbooks publication is a programming guide that highlights the features and tooling included with Rational Application Developer V7.0. Many of the chapters provide working examples that demonstrate how to use the tooling to develop applications, as well as how to achieve the benefits of visual and rapid application development.

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

SG24-7501-00 ISBN 0738486671