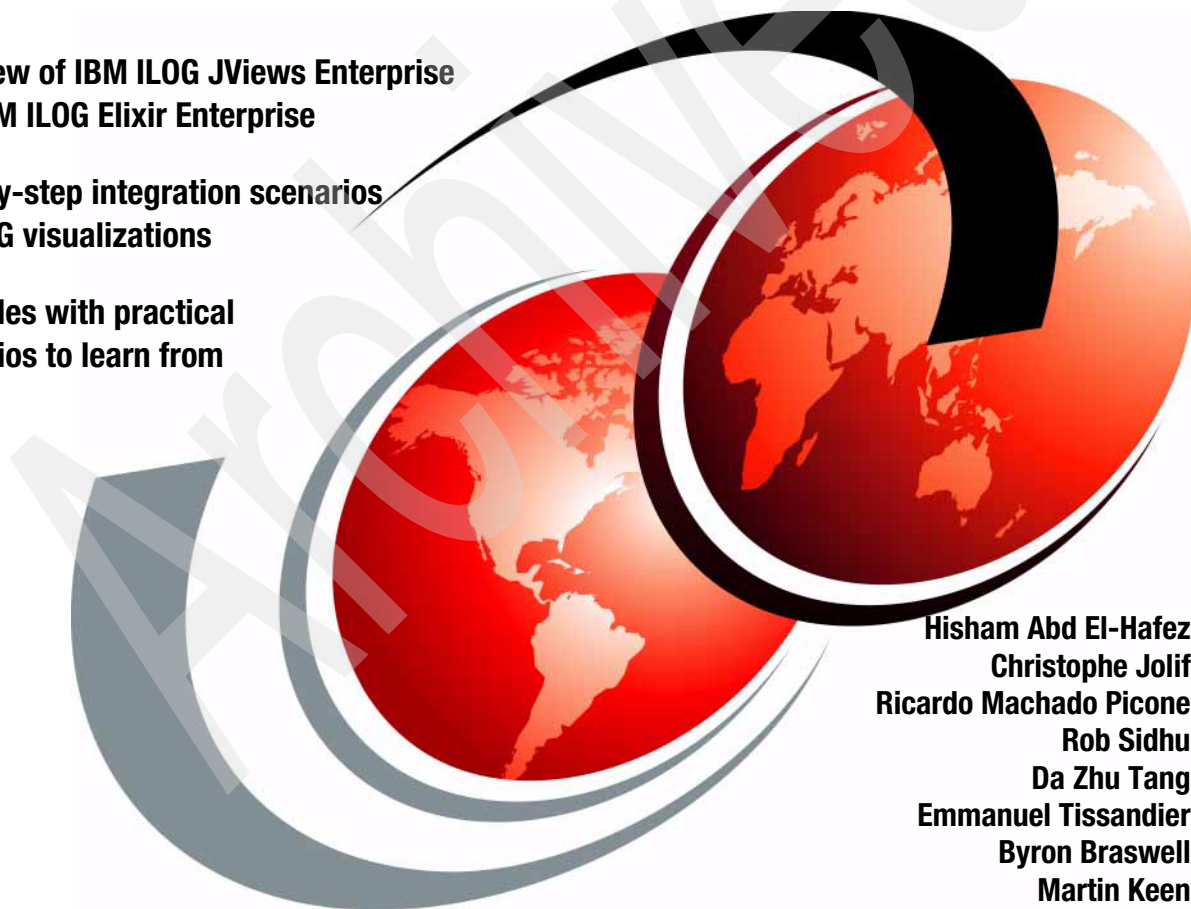


IBM ILOG Visualization Integration Scenarios

Overview of IBM ILOG JViews Enterprise
and IBM ILOG Elixir Enterprise

Step-by-step integration scenarios
for ILOG visualizations

Examples with practical
scenarios to learn from



Hisham Abd El-Hafez
Christophe Jolif
Ricardo Machado Picone
Rob Sidhu
Da Zhu Tang
Emmanuel Tissandier
Byron Braswell
Martin Keen



International Technical Support Organization

IBM ILOG Visualization Integration Scenarios

February 2011

Archived

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

First Edition (February 2011)

This edition applies to IBM ILOG JViews Enterprise 8.7 and IBM ILOG Elixir Enterprise 3.0.

© Copyright International Business Machines Corporation 2011. All rights reserved.

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

Contents

Notices	vii
Trademarks	viii
 Preface	ix
The team who wrote this book	ix
Now you can become a published author, too!	xi
Comments welcome	xii
Stay connected to IBM Redbooks	xii
 Chapter 1. Introduction to IBM ILOG Visualization	1
1.1 Overview of IBM ILOG Visualization	2
1.1.1 What is IBM ILOG Visualization?	2
1.1.2 Principal benefits of using ILOG Visualization components	2
1.1.3 Supported platforms	3
1.2 What is IBM ILOG JViews Enterprise	7
1.2.1 Diagramming services	8
1.2.2 Gantt charting services	9
1.2.3 Mapping services	11
1.2.4 Charting services	13
1.2.5 Rapid Application Development tools	15
1.2.6 Customizable SDKs and MVC architecture	16
1.2.7 Deployment options available for IBM ILOG JViews Enterprise components	19
1.2.8 Required software for the creation of an IBM ILOG JViews Enterprise display	20
1.3 What is IBM ILOG Elixir Enterprise	21
1.3.1 Components available in IBM ILOG Elixir Enterprise	22
1.3.2 Required software for creating a Flex application using IBM ILOG Elixir Enterprise	34
1.3.3 Deployment options available for IBM ILOG Elixir components	35
1.4 Scenarios for integration with other IBM software	35
 Chapter 2. Using WebSphere REST Technology with ILOG Visualization products	37
2.1 Overview of REST	38
2.1.1 Basic concepts	38
2.1.2 Benefits of using REST	39
2.2 Designing a RESTful service	40
2.2.1 A RESTful service scenario	40

2.2.2	Defining the service resources	41
2.2.3	Defining the data format	43
2.3	Implementing a RESTful service	43
2.3.1	Software requirements	44
2.3.2	Implementing the service	44
2.3.3	Deploying and starting the service	56
2.4	Using the IBM ILOG Elixir Enterprise product to visualize data from a RESTful service	57
2.4.1	Overview	57
2.4.2	Software requirements	58
2.4.3	Integration scenario	58
2.4.4	Elixir diagram implementation	59
2.5	Using the IBM ILOG JViews Enterprise product to visualize data from a RESTful service	70
2.5.1	Overview	70
2.5.2	Software requirements	71
2.5.3	JViews diagram implementation	71
2.6	Conclusion	82
Chapter 3. Using IBM ILOG Visualization and IBM Cognos for web applications		85
3.1	Introduction to IBM Cognos Mashup Service	85
3.1.1	The CMS DataSet format	86
3.2	Creating the IBM Cognos sample report	87
3.2.1	Software requirements	88
3.2.2	The XML sample data source file	88
3.2.3	Creating the data source in IBM Cognos Framework Manager	90
3.2.4	Publishing a Framework Manager package for the IBM Human Resources sample report	93
3.2.5	Creating the listing report using the IBM Cognos Report Studio	96
3.3	Integrating ILOG JViews Gantt visualization with IBM Cognos report	101
3.3.1	Software requirements	101
3.3.2	Creating the IBM Human Resources JViews Gantt chart project	101
3.3.3	Components for Java and web deployment	105
3.3.4	Testing the application	110
3.4	Integrating the ILOG Elixir Gantt component with IBM Cognos report	111
3.4.1	Software requirements	111
3.4.2	Introduction to ILOG Elixir Gantt resource chart	112
3.4.3	Creating the IBM Human Resources Elixir Gantt chart project	113
3.4.4	Calling Cognos Mashup Services (CMS) from Flex	117
3.4.5	Displaying CMS Data in the Gantt component	122
3.4.6	Deploying the code into the IBM Cognos IBM Cognos server	130
3.5	Publishing the ILOG Visualizations to IBM Cognos Business Insight	131

Chapter 4. Creating IBM ILOG Elixir iWidgets for Mashup Center	135
4.1 Creating an iWidget wrapper for an IBM ILOG Elixir component	136
4.1.1 Software requirements	138
4.1.2 Creating an iWidget in Rational Application Developer	138
4.1.3 Creating the Adobe Flex application for the iWidget	144
4.1.4 Bringing the Flex application into the iWidget	147
4.1.5 Sending iWidget data to the Flex application	150
4.1.6 Testing the iWidget in Rational Application Developer	152
4.1.7 Adding more events to the iWidget	154
4.1.8 Creating an edit mode for the iWidget	158
4.2 Deploying the IBM ILOG Elixir iWidget in IBM Mashup Center	164
4.2.1 Software requirements	165
4.2.2 Deploying the iWidget to the catalog	165
4.2.3 Prepare your data with the feed generator	167
4.2.4 Using the iWidget in mashup builder	168
 Chapter 5. Business Monitoring with ILOG Elixir and WebSphere Business Space	 177
5.1 Introduction to IBM Business Space powered by WebSphere	178
5.2 Business monitoring using IBM ILOG Elixir components	179
5.2.1 Business monitoring	179
5.2.2 The integration scenario	181
5.2.3 Software requirements	182
5.3 WebSphere Business Monitor REST Services	183
5.4 Creating the Elixir dashboard	183
5.4.1 Creating the dashboard in MXML	184
5.4.2 Creating a new KPI in WebSphere Business Monitor	187
5.4.3 Calling the WebSphere Business Monitor REST Service	189
5.4.4 Displaying the KPI in Elixir components	195
5.5 Deploying the Elixir application using the Web Site widget	205
5.5.1 Deploying the Elixir Dashboard in the Business Space server	205
5.5.2 Creating the Web Site widget	209
5.6 Creating an iWidget to deploy in Business Space	211
5.6.1 Creating the iWidget definition	212
5.6.2 Adding the Elixir dashboard into the widget	218
5.6.3 Implementation of the widget	219
5.6.4 Testing the widget	222
5.6.5 Creating an edit mode for the iWidget	224
5.6.6 Registering the iWidget in Business Space	229
5.6.7 Using the widget in Business Space	235
 Chapter 6. Integrating ILOG JViews JSF components in WebSphere Dashboard Framework	 239
6.1 Introduction to WebSphere Dashboard Framework	240

6.2	WebSphere Dashboard Framework installation	240
6.2.1	Software requirements	240
6.2.2	Installing WebSphere Dashboard Framework	240
6.3	How WebSphere Dashboard Framework integrates with ILOG JViews in the current release	241
6.4	How to develop a customized JViews Gantt JSF application	242
6.4.1	Software requirements for this section	243
6.4.2	Integration scenario	243
6.4.3	Create WebSphere Portlet Factory project	244
6.4.4	Create JViews Gantt Java project	248
6.4.5	Configure WebSphere Portlet Factory Project to allow JViews Gantt support	257
6.4.6	Add builders in WebSphere Portlet Factory	260
6.4.7	Debug the project in WebSphere Portlet Factory Designer	268
6.5	How to deploy the application	272
6.5.1	Run the application on WebSphere Application Server	273
6.5.2	Run the application on WebSphere Portal Server	277
6.5.3	Conclusion	286
	Appendix A. Additional material	287
	Locating the web material	287
	Using the web material	287
	Downloading and extracting the web material	289
	Abbreviations and acronyms	291
	Related publications	293
	IBM Redbooks	293
	Help from IBM	293

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

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

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

Cognos®
DB2®
IBM®
ILOG Elixir®

ILOG®
InfoSphere™
Lotus®
Rational®

Redbooks®
Redpaper™
Redbooks (logo) ®
WebSphere®

The following terms are trademarks of other companies:

Adobe, the Adobe logo, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

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

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® ILOG® Visualization products allow you to create the most advanced graphical user interfaces for line-of-business applications, help users understand their data better, and react to a changing market faster and smarter.

This IBM Redbooks® publication describes two IBM Visualization products: IBM ILOG JViews Enterprise and IBM ILOG Elixir® Enterprise. It provides detailed samples and scenarios covering how these products can be integrated with other IBM software such as IBM WebSphere® REST Technology, IBM Cognos®, IBM Mashup Center, IBM WebSphere Business Monitor and Business Space, and IBM WebSphere Dashboard Framework to provide Web 2.0 and Ajax visualization solutions.

This book is targeted to application interface developers and programmers who develop highly advanced graphical user interfaces using IBM ILOG Visualization products with IBM Cognos, IBM Mashup Center, IBM WebSphere Business Monitor and Business Space, and IBM WebSphere Dashboard Framework.

The team who wrote this book

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

Hisham Abd El-Hafez is an IT Specialist at the Cairo Technology Development Center (CTDC) of IBM Egypt. He works on a business intelligence application that provides business reporting for IBM Support personnel and IBM managers worldwide. He has more than five years of experience in application development. His area of expertise includes J2EE application development, IBM DB2® and Web 2.0. He holds a bachelor's degree in Computer Engineering from Cairo University, Egypt.

Christophe Jolif is an Advisory Software Engineer at IBM France. He leads the IBM ILOG Elixir development team and contributes to the product. He also writes about ILOG Elixir on the team blog and in technical articles. Before joining IBM, Christophe worked for ILOG during more than a decade developing advanced visualization components in Java™, Ajax and Adobe® Flex. He holds an engineer's degree in Computer Science from the Ecole Centrale Marseille, France.

Ricardo Machado Picone is an IBM IT Specialist in Brazil. He has six years of experience in business intelligence applications. He leads a team that provides IBM software support metrics to IBM managers/planners worldwide. His areas of expertise include IBM Cognos and IBM DB2. Ricardo is an IBM Certified Business Intelligence Solutions professional and has been working on helping the Brazilian Business Intelligence Center of Competence (CoC-BI) in areas such as IBM Cognos infrastructure and education. He holds a bachelor's degree in Business Administration from the Universidade Paulista, Brazil.

Rob Sidhu is a Technical Support Engineer at the Bay Area Lab in Foster City, California. He has seven years of experience in the software industry. He holds a Bachelor of Science degree in Computer Science from San Jose State University. His areas of expertise include the IBM ILOG visualization software products, including the Java, Flex, .NET, and C++ products.

Da Zhu Tang is an Advisory Software Engineer at IBM China Development Lab. He has eight years of experience in software architecture and development. Before joining IBM China, Da Zhu worked for three years for ILOG developing advanced visualization components in Java and Ajax. He holds a master's degree in Software Engineering from the ZhongShan University, China.

Emmanuel Tissandier is a Senior Technical Staff Member at IBM France. He has 20 years of experience in software architecture and development. Before joining IBM France, he worked for 15 years at ILOG. He was the initiator and team leader for the ILOG JViews product and is the architect of the .NET ILOG visualization product line. His areas of expertise include advanced visualization components development on the Java and .NET platform, and rich internet applications (RIA) developments in Ajax, Microsoft® Silverlight, and Adobe Flex. He holds an engineer's degree in Computer Science and Applied Mathematics from the École nationale supérieure d'électrotechnique, d'électronique, d'informatique, d'hydraulique et des télécommunications, France.

Byron Braswell is a Networking Professional at the ITSO, Raleigh Center. He writes extensively in the areas of networking, application integration middleware, and personal computer software. Before joining the ITSO, Byron worked in IBM Learning Services Development in networking education development. He received a bachelor's degree in Physics and a master's degree in Computer Sciences from Texas A&M University.

Martin Keen is a Consulting IT Specialist at the ITSO, Raleigh Center. He writes extensively about WebSphere products and service-oriented architecture (SOA). He also teaches IBM classes worldwide about WebSphere, SOA, and ESB. Before joining the ITSO, Martin worked in the EMEA WebSphere Lab Services team in Hursley, United Kingdom. Martin holds a bachelor's degree in Computer Studies from the Southampton Institute of Higher Education.

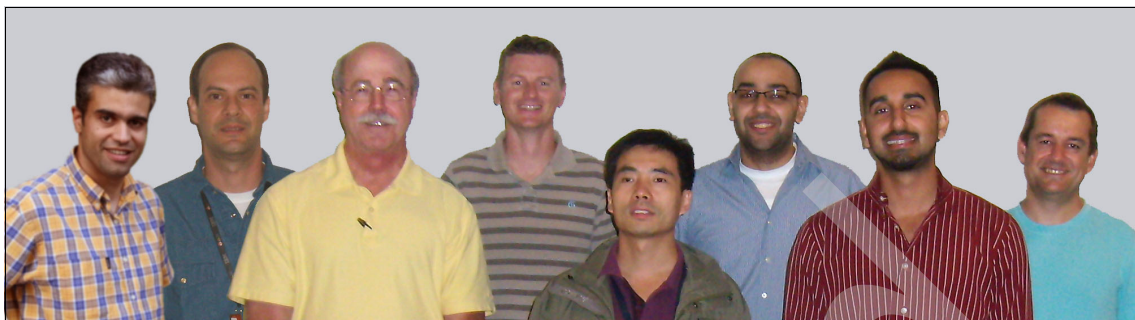


Figure 1 Team (left-to-right): Christophe, Ricardo, Byron, Martin, Da Zhu, Hisham, Rob, and Emmanuel

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

Tamikia Barrow
Linda Robinson
Margaret Ticknor
International Technical Support Organization, Raleigh Center

Damien Mandrioli
Patrick Megard
Julian Payne
IBM France

Richard Johnson
Eric Wayne
IBM USA

Now you can become a published author, too!

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

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

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

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

- ▶ Use the online **Contact us** review Redbooks form found at:
ibm.com/redbooks
- ▶ Send your comments in an email to:
redbooks@us.ibm.com
- ▶ Mail your comments to:
IBM Corporation, International Technical Support Organization
Dept. HYTD Mail Station P099
2455 South Road
Poughkeepsie, NY 12601-5400

Stay connected to IBM Redbooks

- ▶ Find us on Facebook:
<http://www.facebook.com/IBMRedbooks>
- ▶ Follow us on Twitter:
<http://twitter.com/ibmredbooks>
- ▶ Look for us on LinkedIn:
<http://www.linkedin.com/groups?home=&gid=2130806>
- ▶ Explore new Redbooks publications, residencies, and workshops with the IBM Redbooks weekly newsletter:
<https://www.redbooks.ibm.com/Redbooks.nsf/subscribe?OpenForm>
- ▶ Stay current on recent Redbooks publications with RSS Feeds:
<http://www.redbooks.ibm.com/rss.html>

Introduction to IBM ILOG Visualization

This chapter provides a brief overview of the IBM ILOG Visualization offering and describes in more detail the two products discussed throughout the rest of the book: IBM ILOG JViews Enterprise 8.7 and IBM ILOG Elixir Enterprise 3.0.

This chapter also discusses the purpose of the book: How the IBM ILOG Visualization products can be integrated with other IBM software to provide Web 2.0 and Ajax visualization solutions.

1.1 Overview of IBM ILOG Visualization

This section provides an overview of IBM ILOG Visualization software.

1.1.1 What is IBM ILOG Visualization?

With the amount of data being made available to businesses, there is an increasing need to make sense of it all. Visualization software from IBM provides the industry's most comprehensive set of graphics products for creating highly graphical, interactive displays, allowing end users to have a visual representation and a better understanding of the information.

IBM ILOG Visualization can be used to create the following kinds of displays:

- ▶ Diagrams
- ▶ Business dashboards
- ▶ Process control screens
- ▶ Maps with objects
- ▶ Charts (2D or 3D)
- ▶ Project schedules
- ▶ Resource usage displays
- ▶ Calendars
- ▶ Heat maps
- ▶ Tree maps
- ▶ Organizational charts

The visualization software is delivered as Software Development Kits (SDKs) that help user interface developers create custom graphical user interfaces for business applications.

Note: IBM ILOG Visualization software is designed to be used by developers, to create applications for their users. It is not designed to be used by users.

User interface developers reduce development time and risk. Being able to visualize the underlying data, end users understand their information better, and can react faster.

1.1.2 Principal benefits of using ILOG Visualization components

IBM ILOG Visualization components provide, among others, the following benefits:

- ▶ A wide spectrum of advanced user interface display types: diagrams, dashboards, charts, schedule displays, and maps
- ▶ Substantial cost savings versus in-house user interface development effort for difficult display types
- ▶ Improved reliability and quality, with years of successful installations in a wide variety of demanding application domains
- ▶ Offerings available on multiple development platforms
- ▶ Increased customer satisfaction because the user interface is customized to meet user requirements

1.1.3 Supported platforms

IBM ILOG Visualization software is provided on the following development platforms:

- ▶ IBM ILOG Visualization for Java
- ▶ IBM ILOG Visualization for Adobe Flex
- ▶ IBM ILOG Visualization for .NET
- ▶ IBM ILOG Visualization for C++

IBM ILOG Visualization for Java

IBM ILOG Visualization for Java components provide custom, graphical components for desktop, Ajax, and Eclipse displays. These components, delivered as point-and-click editing tools and full-featured software development kits (SDKs), allow Java developers to add intuitive, graphical displays to their mission-critical applications.

IBM ILOG JViews Enterprise is the main product in the Java offering for IBM ILOG Visualization software and is discussed throughout this book.

The following are other IBM ILOG Java Visualization products that are not discussed in this book:

- ▶ IBM ILOG JViews Telecom Graphic Objects

Simplify development for telecommunications systems as shown in Figure 1-1 on page 4.

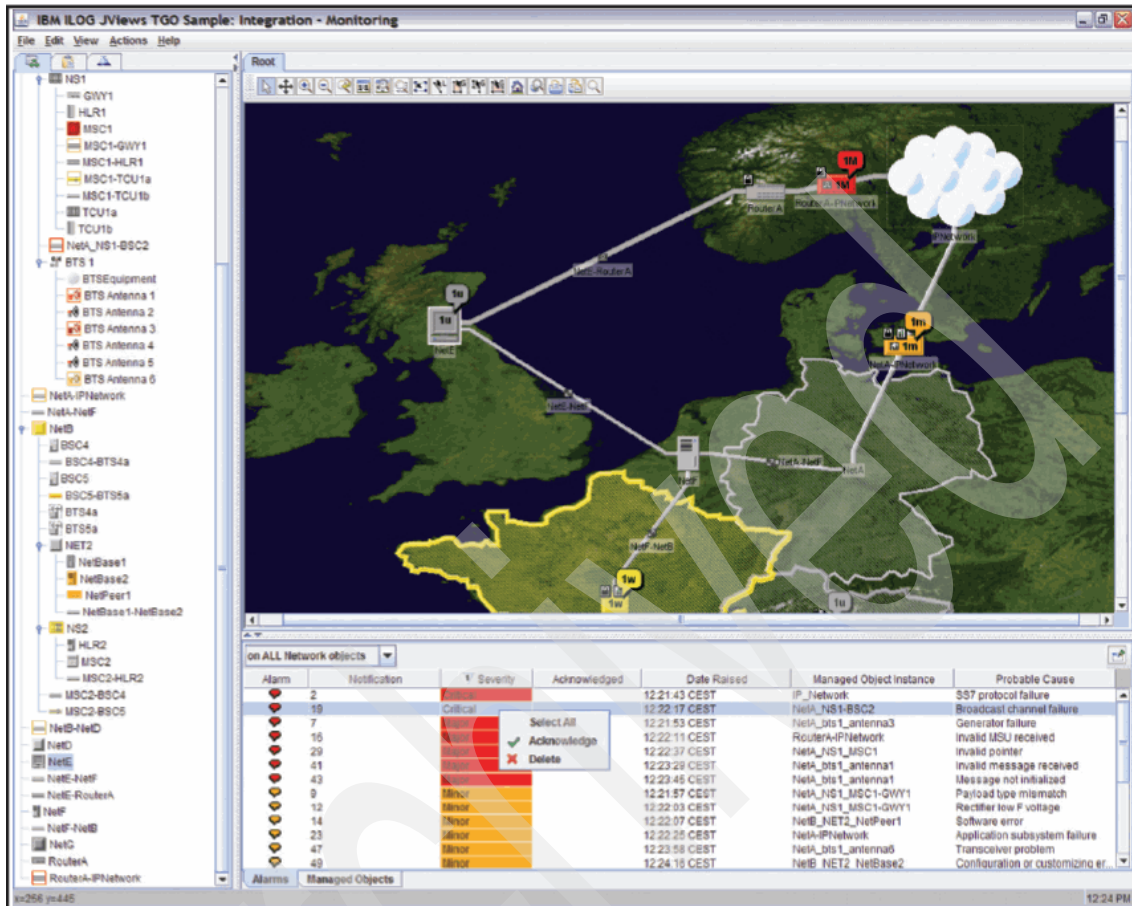
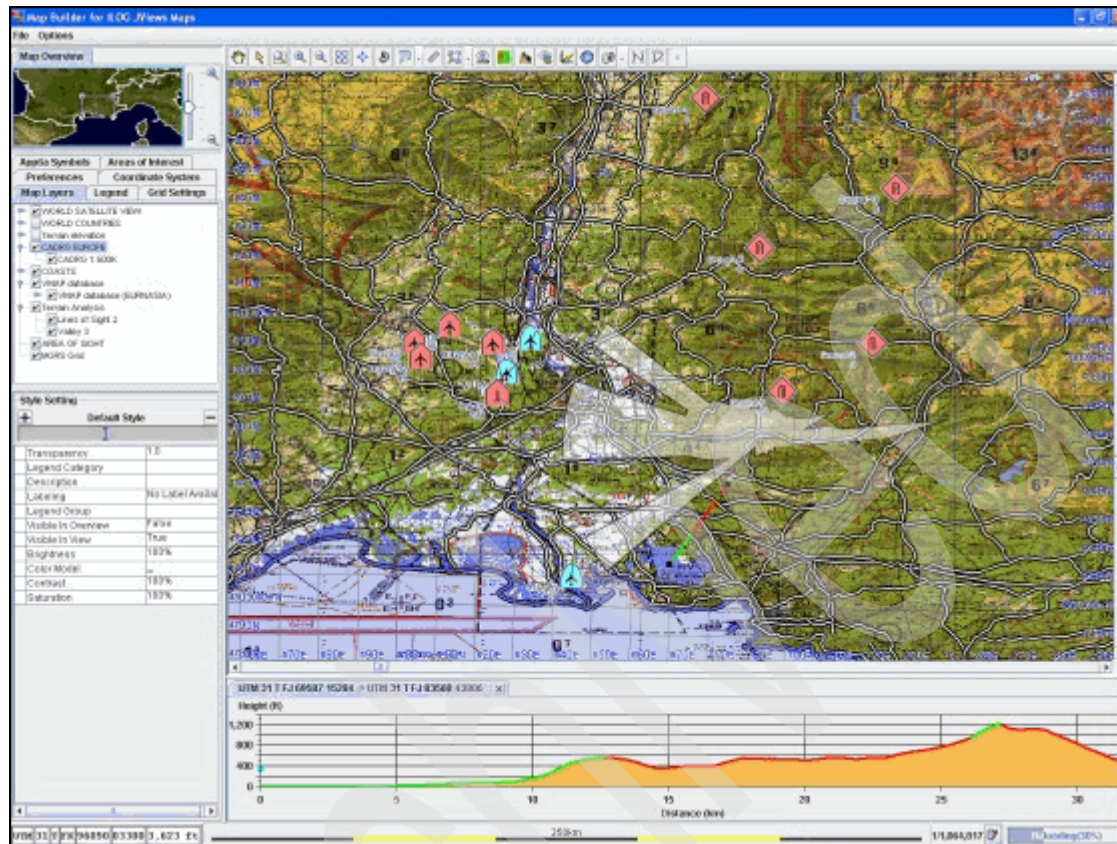


Figure 1-1 IBM ILOG JViews Telecom graphic objects

► IBM ILOG JViews Maps for Defense

Create realistic, dynamic map displays for command-and-control systems as shown in Figure 1-2 on page 5.



- ▶ IBM ILOG JViews Graph Layout for Eclipse

Enhance your Eclipse-platform based diagram drawing applications by adding world-class graph layout services as shown in Figure 1-3 on page 6.

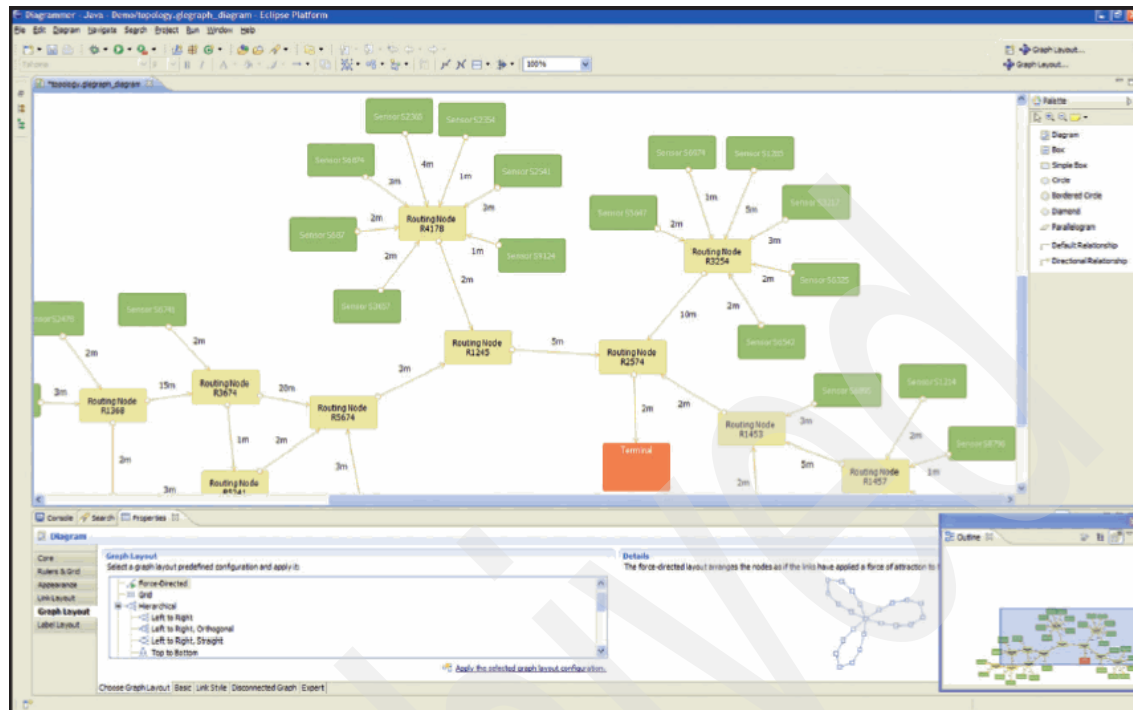


Figure 1-3 IBM ILOG JViews Graph Layout for Eclipse

IBM ILOG Visualization for Adobe Flex

IBM ILOG Elixir Enterprise extends the Adobe Flex and Adobe AIR platforms by adding user interface controls for more intuitive, interactive displays.

It provides a full spectrum of advanced graphical displays for the most demanding line-of-business applications. Developers can use Flex visualization displays to create applications that are functionally superior and visually appealing.

IBM ILOG Elixir Enterprise is discussed throughout this book.

IBM ILOG Visualization for .NET

IBM ILOG Visualization for .NET provides graphical user interface components for the .NET platforms: desktop, Ajax Web, and Silverlight. Developers can add sophisticated diagram, dashboard, and schedule displays to the user interfaces of their mission-critical business applications.

Using ILOG Visualization reduces the time and risk by allowing developers to create the user interface quickly and easily. Users benefit by interacting with

displays that present their data better, enabling them to understand more and react faster.

The IBM ILOG Visualization .NET solutions are not discussed in this book.

IBM ILOG Visualization for C++

IBM ILOG provides graphical user interface components for C++ developers, enabling them to deliver the information and interactivity that businesses need to stay agile.

With these products, developers can upgrade their user interfaces to highly-interactive, highly-graphical displays. These displays can include both standard user interface controls, such as menus, buttons; and advanced features such as maps, charts, diagrams, schedules, and schematics. These enable users to better model, monitor, and analyze their business information. ILOG Visualization greatly reduces development time and risk, and users benefit from more responsive, intuitive information displays.

All of these C++ products offer portability across multiple Unix, Linux®, and Windows® platforms.

The IBM ILOG Visualization C++ solutions are not discussed in this book.

1.2 What is IBM ILOG JViews Enterprise

Just as “a picture is worth a thousand words,” graphical displays have proven far better at communicating information than text displays. For many applications, text and numbers are not enough. The user needs to see data displayed graphically to be able to quickly comprehend its meaning. Representing data graphically in charts, diagrams, and maps, for example, helps users understand complex information and respond rapidly to changing events.

But delivering such graphics in user interfaces isn’t easy. Specialized skills, including graph layout theory, systems architectures, digital mapping, graphics design, and web deployment technologies, are required to create a robust solution.

IBM ILOG JViews Enterprise is a product specifically designed for organizations that are focused on creating user interfaces for advanced business software applications. Specifically, JViews Enterprise offers a rich set of charts, maps, diagrams, dashboards, and schedule displays that can be easily customized and integrated into a user interface. The result is that software projects reduce

development time and risk, and users understand their information better and faster.

IBM ILOG JViews Enterprise provides a broad range of commonly-used advanced display types, including the following:

- ▶ Diagrams, used to show objects and their interconnections. Typical uses include flowcharts, organization charts, network topologies, and process modelers.
- ▶ Dashboards, which are unified collections of individual display objects (for example, charts, gauges, and maps) that graphically convey current industrial control systems data to the user; HMI for SCADA systems are easily adaptable. In addition to industrial dashboards, which show the status of physical equipment, business dashboards and process control dashboards are also supported.
- ▶ Maps, which show where assets are physically located. Assets are often tied to underlying system data, and might change position or shape to reflect current status.
- ▶ Resources charts and Gantt charts which are a specialized types of chart displays used to show how business resources are allocated over time. Typical use cases include project management, manufacturing plans, and supply chain plans.
- ▶ Schedules, which are a specialized types of chart displays used to show how business resources are allocated over time.

1.2.1 Diagramming services

Many business applications require diagrams, which are specialized user interface displays that represent elements in a system and how they are related to each other. A business process modeler, for example, can use a diagram to represent tasks and the flow of work between them. In a telecom application, a diagram can be used to show a network and include graphic objects that change to depict the current status of equipment. Companies use diagrams to display networks, process flows, and organization charts: wherever they need to better understand business entities and their interconnections.

Another useful display is the dashboard, which is used to show the current status and performance of key performance indicators (KPI). It consists of custom graphic objects that are directly connected to underlying business objects.

Building a professional diagram or dashboard display is no easy task. It requires expertise in several areas, including graphics design and graph layout algorithms. Furthermore, the final display must be able to handle large data sets,

real-time data updates and, potentially, deployment to the web. See Figure 1-4 for examples.

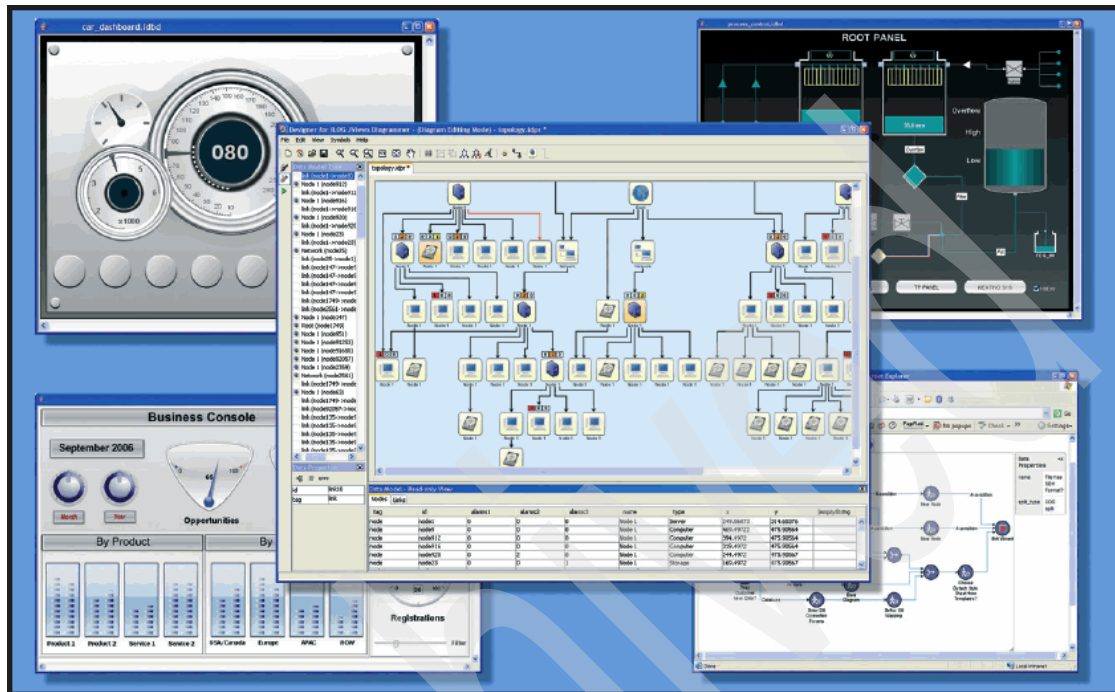


Figure 1-4 IBM ILOG JViews Diagrammer

Large diagrams with many objects and interconnections can appear cluttered and disorganized. Graph layout algorithms address this problem by automatically rearranging the window elements into a more readable form. JViews Diagrammer comes with a dozen layout algorithms, including ones for popular node, link, and label algorithms, that can be used separately or together, and across many domains.

1.2.2 Gantt charting services

Planning and scheduling applications often include timeline displays called Gantt charts, designed for viewing and editing information about resources and activities. These views show how tasks are allocated with respect to each other and enable companies to visualize how their resources are being utilized. Many industry applications, including those for manufacturing, transportation, and project management, depend on these charts for organizing operations and to boost efficiency.

Java developers tasked with including such displays in their user interfaces must ensure they provide the right Gantt charts, and that the charts customize easily to user-specific interactions and renderings. Furthermore, the displays must be able to handle large data sets quickly and efficiently. Finally, the ideal solution must be deployable to both the desktop and the web, with maximal reuse across platforms.

Comprehensive set of view types

Depending on their purpose, applications could require one or more types of Gantt displays. For instance, a factory scheduling system might need to display what equipment is doing at particular times, and a project planning application might need to depict activities and their interdependencies. These views are often combined and synchronized to form more elaborate displays. See Figure 1-5.

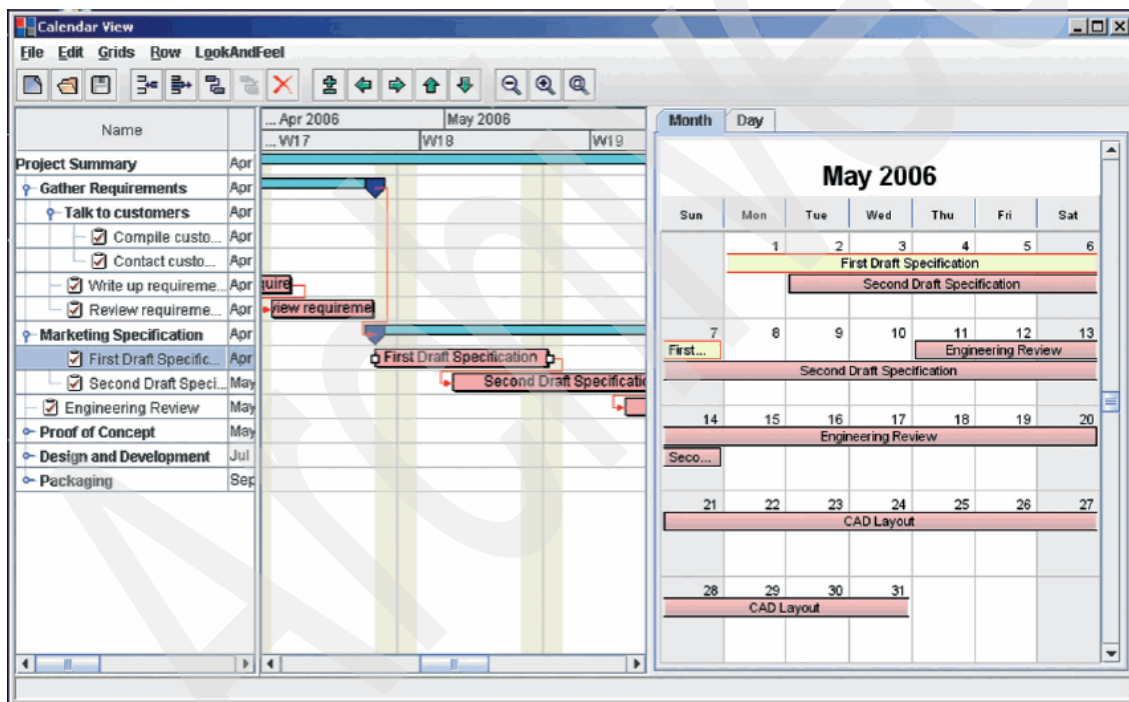


Figure 1-5 IBM ILOG JViews Gantt

IBM ILOG JViews Gantt supports the following display types:

- ▶ Activity-oriented chart (Gantt chart)

Allows users to see when tasks are scheduled and the constraints between tasks. This type of chart is often used in project planning applications.

- ▶ Resource-oriented chart (schedule chart)
Shows the activities and operational times assigned to resources. For example, a factory schedule might show machines and the times they are busy.
- ▶ Tree-table view
Derived from Swing's JTable, the tree-table view shows resources or activities in rows that can be expanded or collapsed (to represent subresources or subactivities).
- ▶ Resource load chart
Displays how the capacity of a set of resources is being used over time.
- ▶ Calendar views
Display tasks in traditional month-by-month or day views
- ▶ Critical path views
Highlights which paths in a task will cause an overall delay.

1.2.3 Mapping services

Many mission-critical planning and monitoring applications require an asset management map, typically a geospatial display consisting of a map overlaid with custom graphic symbols. These symbols need to change their location or appearance as the underlying business objects are updated. Application domains that require this type of map include physical network topologies, supply chain monitoring, highway traffic management, and air traffic control: any area in which it is critical for the user to know both the condition and location of business assets.

Creating such a map display from the ground up is a difficult undertaking. It must be designed to handle huge quantities of map and symbol data, and also to handle data updates efficiently. One can opt for adapting a traditional Geographic Information System (GIS) display, but its strong points - map data creation, distribution, and queries - are not needed for these types of asset map displays. Furthermore, GIS displays often fall short in their ability to handle custom symbols and update them quickly. See Figure 1-6 on page 12.

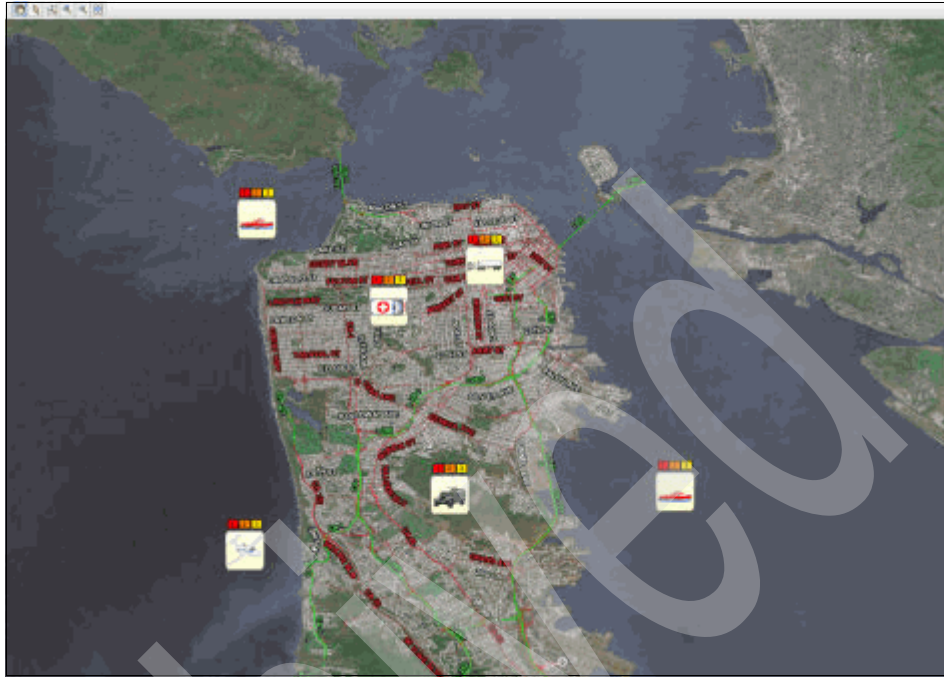


Figure 1-6 IBM ILOG JViews maps

Support for all major map formats

A common hurdle for most monitoring applications is reading map data. JViews Maps supports many extensible formats:

- ▶ Google Earth KML and KMZ files
- ▶ OpenGIS WMS online servers
- ▶ Oracle Spatial Map servers
- ▶ ESRI Shape Files (.shp)
- ▶ MapInfo (.mif, .mid)
- ▶ TIGER/Line (.RTx)
- ▶ GeoTIFF with support for tiled maps, JPEG in TIFF, and LZW compression
- ▶ Non georeferenced AutoCAD DXF, or images in GIF, JPEG, PNG, or TIFF format (with an accompanying tool for manual map or symbol placement)
- ▶ Integration with Safe Software's FME Engine, which allows for over 200 additional formats

Full set of mapping services provided

JViews Maps provides a full spectrum of typical mapping services. These can be used "as is" by the developer or extended for further specialized needs. They include:

- ▶ Interactive zooming, with zoom-dependent information display
- ▶ Map rotation and panning
- ▶ Area of interest markers
- ▶ Synchronized overviews and map scales
- ▶ Polyline measurements in orthodromy modes
- ▶ Numerous coordinate formatters in DMS latitude and longitude (Lat/Lon), geocentric, UTM, and geodesic coordinates
- ▶ Exhaustive coordinate systems: 1,944 predefined coordinate systems with 27 projections, 58 ellipsoids, and Molodenski datum
- ▶ Distinct measuring units for ground distances and elevations
- ▶ Adaptive grid, in Lat/Lon or UTM/MGRS reference systems
- ▶ True geodetic representation and dateline wrapping
- ▶ Map legend generation
- ▶ Annotation toolbar, with complete marking and labeling functionality

1.2.4 Charting services

Incorporating a chart into a Java-based graphical user interface (GUI) used to mean settling for a static display with crude user interaction capabilities and unattractive graphics. Moreover, if a huge data set needed to be displayed or a chart had to update in real time, the shortcomings of the underlying charting architecture would quickly become apparent. Also, if Java developers wanted to deploy the same charts to both the desktop and the web, they often had to redo much of their work using different products.

With IBM ILOG JViews Charts, there is no compromising on usability or performance. Developers get the ability to quickly and fully customize displays using a robust architecture that scales for large data sets and fast redraws. Both desktop and web deployment is supported from within the same product, with maximum reuse across the platforms. The end result is that the Java developer saves a significant amount of time, and users get chart displays that meet their exact needs.

Broad range of chart types

JViews Charts comes with a full set of chart types: 2D point, line, bar, stacked, bubble, area, high/low, candle, radar, polar, pie, and treemaps. These charts cover the most common needs and include the following ready-to-use and ready-to-customize features:

- ▶ Linear and logarithmic scales: Cartesian and polar charts
- ▶ Multiple, flipped, and swapped axes
- ▶ Smart tick-mark labels
- ▶ Labels on data points and user definable data point markers
- ▶ Legends explaining graphical elements

- Prebuilt navigation tools that include selection, panning, and zooming

See Figure 1-7 for examples.



Figure 1-7 IBM ILOG JViews charts

High-performance services

JViews Charts delivers the video-like refresh rates required by monitoring applications. Furthermore, all of its charts zoom and scroll, adapt their scales automatically, and support multiple axes. For applications with very large data sets, JViews Charts provides load-on-demand functionality that places data in a locally managed cache and displays it as needed.

Treemaps for data analysis

Treemaps are a type of chart for displaying large, multidimensional data sets. They bind the parameters of each record to display attributes such as color and size, allowing users to easily spot patterns or data outliers. The treemap chart included in JViews Charts lets users interact with the display, selecting and drilling down to get more information.

1.2.5 Rapid Application Development tools

This section discusses the Rapid Application Development (RAD) tools that can be used to develop diagramming, charting, and mapping interactive interfaces.

Diagramming

JViews Enterprise provides a powerful RAD tool for easily creating, animating, and deploying reusable symbols: graphic objects that represent elements in a display. In addition, the product includes three tools for assembling diagram and dashboard displays with these symbols.

- **Symbol Editor**

Developers use Symbol Editor to create, edit, and organize symbols. It works with authoring tools such as Adobe Illustrator to provide appealing visual elements. It can also be used to set symbols to change to show data changes. For example, its rules can define alarm conditions for making a factory symbol blink. Symbol Editor comes with extendible palettes of fully functional symbols, including gauges, buttons, meters, and dials.

- **Designer for Diagrammer**

Designer for Diagrammer is a point-and-click editor for easily specifying most aspects of a diagram. Its data definition wizard lets users set visual properties and data source parameters. Developers can specify the appearance of graphics under different data conditions. Graphic objects are taken from the Symbol Editor palettes and configured locally.

- **Dashboard Editor**

Dashboard Editor's point-and-click interface is used to create industrial and business dashboards. It helps the user draw or import a static background from a range of formats (SVG, DXF, JPEG, PNG, and GIF), place Symbol Editor symbols on top of the background, and then connect the symbols to the underlying data. All of this can be done without writing Java code. The resulting dashboard or schematic can be loaded into an application interface and fed data in real time.

Gantt charting

Designer for Gantt is a point-and-click RAD display editor that comes complete with wizards. It helps developers easily specify most aspects of a Gantt display, including setting its data source parameters (such as JDBC, XML, and CSV) and visual properties. Designer also comes with the unique Style Rule tool, which allows developers to define how graphics will appear on the window under different data conditions. The output of Designer is a project file that is loaded into an application at run time.

Mapping

JViews Enterprise accelerates mapping application development with two RAD tools: Map Builder and Symbol Editor. Map Builder helps the developer define the background of a map display, and Symbol Editor is used to create the graphic symbols that the application places on top of the map.

Map Builder enables developers to specify, with just a few mouse clicks, the map data that will be used in the display. Developers can import map data from a wide variety of data formats and then combine them on separate map “layers”; using these layers, both vector and raster data can be mixed to produce the final map. The same tool then makes it easy to alter the appearance of the data by setting parameters that affect the vectorial or image display attributes. The visibility and appearance of different data layers - how and when they appear under different conditions (e.g., zoom levels) - are also easy to define. Built with JViews Maps’ underlying SDK, Map Builder is provided with full source code to enable developers to extend and deliver custom versions of this tool to their customers.

To create custom symbols, JViews Maps is shipped with Symbol Editor (also used in IBM ILOG JViews Diagrammer). This tool helps developers define the visual aspects of a symbol, often by importing graphics from an authoring tool like Adobe Illustrator, and assign behaviors to them, such as how the symbols appear in response to updates in the underlying business object. After the visual and behavioral aspects of a symbol are defined, they can be used in the runtime application to represent the various domain-specific objects that will be monitored on the map.

Charting

Designer for Charts allows developers to easily specify most aspects of their chart displays using a point-and-click editor complete with wizards. It sets the data-source parameters (JDBC, XML, CSV, and others) in addition to the visual properties. Designer for Charts also includes an innovative styling feature that lets developers define precisely how the graphics will be displayed under different data conditions.

1.2.6 Customizable SDKs and MVC architecture

This section introduces software developer kits and architectures that are used to support development of diagramming, charting, and mapping interfaces.

Diagramming

The Diagramming support options include a fully customizable SDK and architecture.

Completely customizable SDK

JViews Diagrammer also comes with a rich, fully documented software development kit (SDK). This ensures that all facets of the user interface, such as user interactions and visual aspects, can be extended and fine-tuned to meet the end users' exact requirements.

Architected for mission-critical applications

The diagramming components are built on an optimized model view controller (MVC) architecture that is familiar to Java Swing developers. Prebuilt data connectors allow developers to easily read XML, flat files, and Java database connectivity (JDBC) tables. Updates can be automatically made whenever data changes. Changing the way window elements appear or update is simple because cascading style sheet (CSS) files make it easy to define or change the visual style of a diagram, even at run time.

This architecture design enables the product to efficiently handle very large data sets, using many techniques to optimize object initialization, connections to data, and window redraws. In addition, all of the layout services provided in the product have been tuned to work with thousands or even tens of thousands of objects - an important distinction because other commercially available graph layout algorithms work well with dozens or hundreds of nodes and links, but slow down and can become unusable with larger sets of data.

With diagramming components for JViews Enterprise, developers can satisfy the requirements of the most demanding applications, delivering diagrams capable of handling many window elements simultaneously with little degradation in performance.

Gantt charting

Gantt Charting support options include a developer's SDK and a robust MVC architecture.

Developer's SDK

The SDK enables Java developers to define custom data sources and create specialized visual renderings and graphical behaviors. The output of Designer for Gantt is a configuration file that can be used directly in a Java application or refined with the included software development kit (SDK).

Powerful MVC architecture

With a Swing-like model view controller (MVC) architecture, Gantt components for JViews Enterprise maintain a clear separation between data and window representations. The data model is completely open and extensible, enabling it to be connected to other display components. When the data model changes, the

connected displays are automatically updated, and when a user interacts with a display, the model changes accordingly.

This robust architecture enables the Gantt components to support high-performance features, including fast panning, video-like zooming and refreshes, search and filtering, and current time indicators. It also supports load-on-demand functionality for seamlessly managing extremely large data sets.

Mapping

Mapping support options include customizable displays, an SDK, and robust architecture.

Customizable, high-performance displays

The mapping components of JViews Enterprise offer a complete set of asset map display services for developers working with Java. These services include point-and-click RAD tools for easily defining map backgrounds and data-aware graphic symbols, an architecture that delivers real-time updates for very large datasets, and a software development kit (SDK) to customize all aspects of the map display. JViews Enterprise adds geospatial capabilities to a user interface whenever a flexible, high-performance solution is required.

Complete software development kit

If an application requires functionality that is not available within the two RAD editing tools, the developer can access the underlying Java SDK. Developers can use the SDK to fine-tune or extend the prebuilt functionality - anything from the look of a symbol to a new map interaction or a custom data source - to meet the end users' exact requirements.

Architected for optimal performance

The mapping components are a purely object-oriented toolkit based on a model view controller (MVC) architecture. As the model changes, the displays linked to it are automatically updated.

In addition, the mapping components also use techniques to ensure they can handle very large map data sets and provide fast window refreshes. These techniques include data subsampling, multi-threaded load-on-demand functionality, and zoom-dependent overlays. The result is that the user experiences smooth map panning, zooming, and image rendering at video-like rates (up to 60 frames per second).

Charting

Charting support options include an open API and MVC architecture.

Completely open API

The software development kit (SDK) enables Java developers to define custom data sources and tailor specialized rendering or graphical behaviors. The output of Designer for Charts is a chart configuration file that can be used directly in a Java application or refined with the included SDK.

Robust MVC Architecture

The charting components for JViews Enterprise implement a Swing-like model view controller (MVC) architecture that cleanly separates data from displays. The data model is completely open and extensible, and can be connected with other application components. Notifications are automatic and transparent. Displays will update as the data model changes, and the model changes with user interactions.

1.2.7 Deployment options available for IBM ILOG JViews Enterprise components

It has become more and more common to deliver the same application both as a rich desktop client for power users and over the web for remote users. JViews Enterprise displays can be deployed on various platforms:

- ▶ Rich client
- ▶ JSF/Ajax/Portals
- ▶ Eclipse

Traditional Java applets and applications are best suited for highly interactive tasks like workflow modeling. For workflow monitoring or administration, a thin client approach usually works best. JViews Enterprise supports both DHTML clients and traditional Java clients.

On the web side, there are dedicated JSP/JSF (JavaServer Faces) components hosted on a web server that generate the interface for the browser. By mixing images and JavaScript/DHTML code, these components deliver content for either traditional web pages or portals that implement the JSR 168 standard. JViews Enterprise components support the leading Ajax (Asynchronous JavaScript and XML) integration frameworks, such as ICESoft ICEFaces, Apache MyFaces Trinidad, and JBoss RichFaces. They are also able to deal with asynchronous requests to manage the Ajax behavior and minimize page refreshing. In addition to traditional visualization and monitoring functions, Ajax behavior provides powerful in-place editing capabilities, such as adding objects to a diagram, connecting entities interactively, showing contextual pop-up menus, and editing an object's attributes. It also provides immediate synchronization with the underlying data models on the server.

1.2.8 Required software for the creation of an IBM ILOG JViews Enterprise display

IBM ILOG JViews Enterprise 8.7 relies on the Java Platform, Standard Edition version 5.0 or later.

Requirements for running JViews Web applications

The following are the requirements to run JView Web applications.

Web browsers

IBM ILOG JViews Web applications support the following browsers:

- ▶ Internet Explorer 6.0, 7.0, or 8.0
- ▶ Firefox 3.x
- ▶ Chrome 3.x or 4.x
- ▶ Safari 4.x

JSF technologies

Requirements on the server depend on whether you deploy IBM ILOG JViews JSF or JavaScript applications.

For IBM ILOG JViews JavaScript applications you must have a web server that runs a Servlet 2.3 or later implementation.

For IBM ILOG JViews JSF applications you must have a web server that runs a JSF 1.1, 1.2, or 2.0 implementation.

IBM ILOG JViews has been tested and works on the following JSF implementations:

- ▶ Sun JSF 1.1 Reference Implementation
- ▶ Sun JSF 1.2 Reference Implementation (shipped with IBM ILOG JViews)
- ▶ Sun JSF 2.0 Reference Implementation
- ▶ Apache MyFaces Core 1.2.3, 1.2.8

IBM ILOG JViews JSF components have been tested and work with the following JSF technologies:

- ▶ ICEFaces 1.7.2 (shipped with IBM ILOG JViews), 1.8.2
- ▶ Apache MyFaces Trinidad 1.2.8 (shipped with IBM ILOG JViews), 1.2.12
- ▶ Apache MyFaces Tomahawk 1.1.6, 1.1.9
- ▶ JSF Facelets 1.1.14 (shipped with IBM ILOG JViews), 1.1.15
- ▶ JBoss RichFaces 3.3.2_SR1

IBM ILOG JViews JSF components have been tested and work with the following client-side technologies:

- ▶ Dojo 1.1.1, 1.4.1 (used in some JViews demos)
- ▶ OpenAjax 1.0, 2.0 compliant

Portals

To run IBM ILOG JViews JSF applications in a portlet, you must have a Portal server that runs a JSR 168 Portlet Implementation with a JSF-Portlet bridge.

IBM ILOG JViews has been tested and works on the following Portlet implementations:

- ▶ Apache Pluto 1.1.7 (tested with Apache Tomcat 6.0.14)
- ▶ JBoss Portal 2.6.7, 2.7.2
- ▶ Liferay 5.1, 5.2.3
- ▶ IBM WAS Portal 6.1.0.1

In the absence of a specified JSF-Portlet bridge to comply with, IBM ILOG JViews supports the following combination of JSF implementation and JSF-Portlet bridge:

- ▶ Sun JSF 1.2 Reference Implementation with Sun JSF-Portlet bridge 1.2.4
- ▶ MyFaces 1.1.5 with Apache JSF-Portlet bridge 1.0.4

Web application servers

IBM ILOG JViews has been tested and works on the following web application servers:

- ▶ Apache Tomcat 6.0.14 (shipped with IBM ILOG JViews), 6.0.18, 6.0.20
- ▶ IBM WebSphere Application Server 7.0
- ▶ IBM WebSphere Application Server Community Edition 2.1.1.3
- ▶ JBoss Application Server 4.2.3.GA, 5.1.0.GA
- ▶ Oracle WebLogic Server 10.3, 10.3.2.0

Important: Although intensive tests have been performed with the web technologies listed in this documentation, no guarantee can be given that all the combinations of these technologies will work. For example, difficulties have been encountered when running Apache Pluto 1.1.7 under Apache Tomcat 6.0.18 or 6.0.20.

1.3 What is IBM ILOG Elixir Enterprise

Adobe Flex and Adobe AIR form a powerful platform for creating graphically rich and highly interactive applications for the Internet (and desktop). It provides a wide variety of basic user interface components such as buttons, menus, and charts.

Many applications require user interfaces that are highly graphical, going beyond what is offered natively in the Adobe platform.

IBM ILOG Elixir Enterprise enhances the Flex and AIR platforms by adding a set of advanced user interface controls for more intuitive, interactive displays: 3D charts, radar charts, and treemap charts; dials and gauges; organization charts; maps and heatmaps; timelines and calendars; resource-oriented and task-oriented schedule displays; and advanced services for creating intuitive diagram displays.

Built on IBM ILOG's decades of graphics visualization know-how, these components reduce development time and risk, and deliver a superior user experience.

Professional-grade graphics components

Developing a sophisticated graphical display in-house can be difficult and expensive, and requires specific expertise. IBM ILOG has been developing professional-grade data display components for the past 20 years, and IBM ILOG Elixir Enterprise - created specifically for the Flex and AIR platform - embodies this expertise. IBM ILOG Elixir Enterprise takes the Adobe platform beyond the simple user interface; presenting data to users with innovative displays that help them understand the information more clearly, react faster, and make better decisions.

Integrated with Flash Builder

IBM ILOG Elixir Enterprise is tightly integrated with Flash Builder, the integrated development environment (IDE) for Flex and AIR, making Elixir's components easy to use: just drag and drop them into place and set their properties from within Flash Builder.

Completely customizable

IBM ILOG Elixir Enterprise components are ready to use "as is" or can be customized with their property sheets. Developers also have full control over the look and feel of Elixir's controls through a fully documented application programming interface (API), which is delivered with complete source code.

1.3.1 Components available in IBM ILOG Elixir Enterprise

IBM ILOG Elixir Enterprise offers the following advanced display components:

► 3D charts

Elixir comes with a full range of 3D charts that complement the 2D charts that Adobe provides. Upgrading to the third dimension and creating more appealing dashboards has never been easier. See Figure 1-8 for an example.

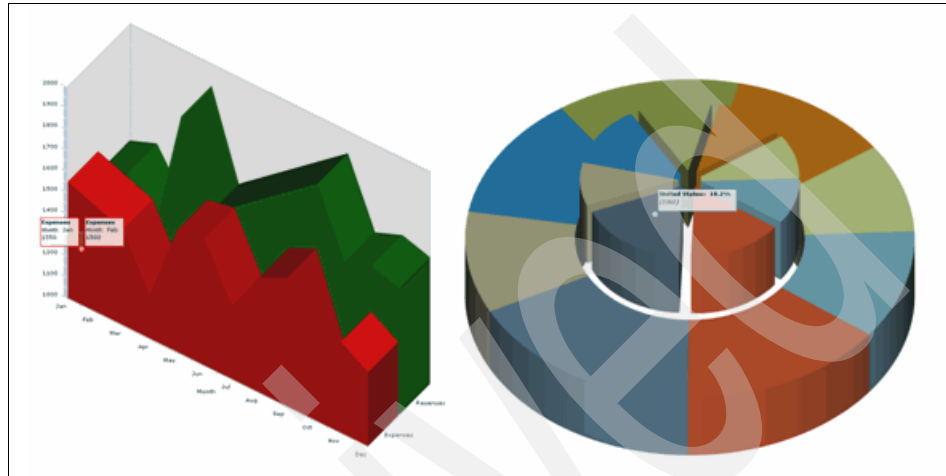


Figure 1-8 3D Charts

► Radar charts

Elixir radar charts are compact dashboard displays that are used to compare key performance indicators (KPI) between two or more entities. See Figure 1-9 on page 24 for an example.

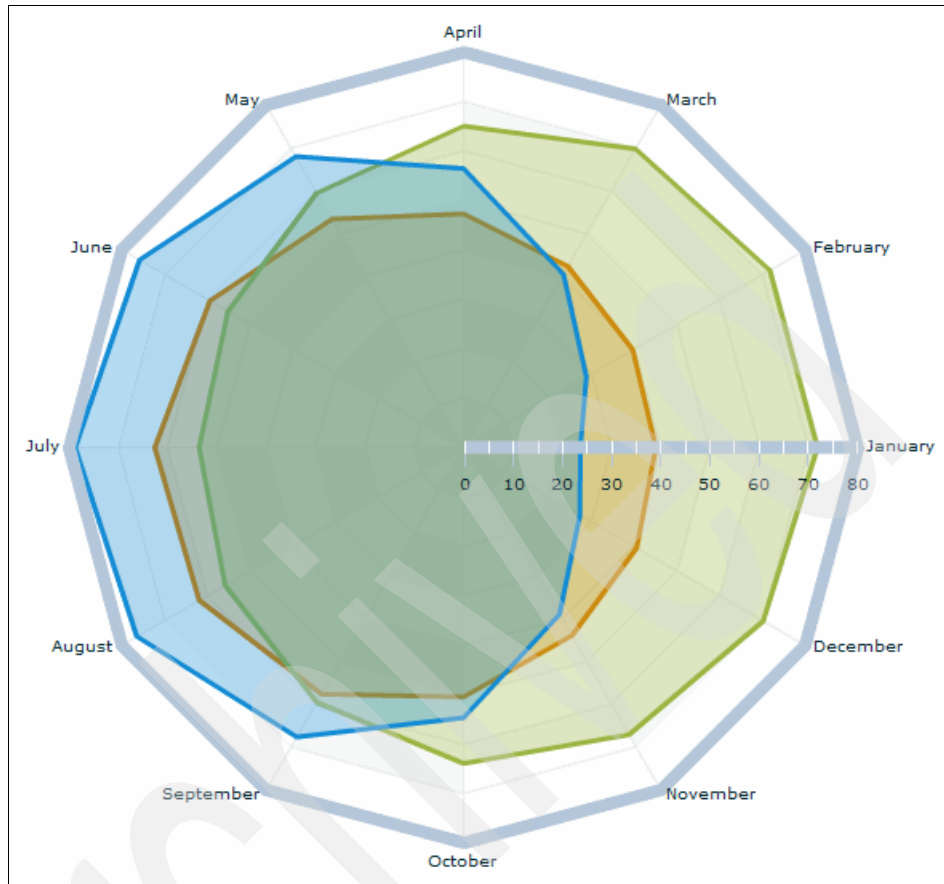


Figure 1-9 Radar chart

► Treemap charts

Elixir's treemaps are innovative charts for detecting trends and outliers in large data sets. They provide an interactive experience that includes dynamic drill down, user-defined clustering, and segment coloring and sizing. See Figure 1-10 on page 25 for an example.



Figure 1-10 Treemap chart

► Gauges and dials

Elixir gauges and dials are all fully interactive and connectable to real-time data sources. Developers can use them as-is or easily create new ones with Elixir's powerful API. See Figure 1-11 on page 26 for an example.

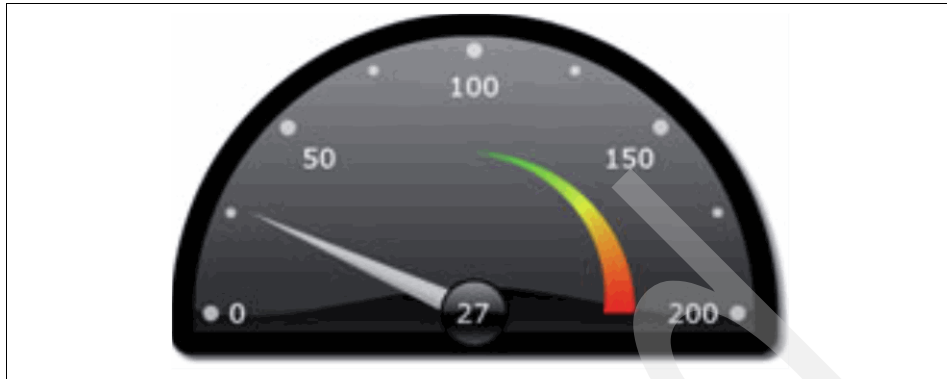


Figure 1-11 Gauge

► Organization charts

Advanced employee organization charts are easy to create with Elixir and can include smooth animation effects and zoom-level dependent information displays. See Figure 1-12 on page 27 for an example.

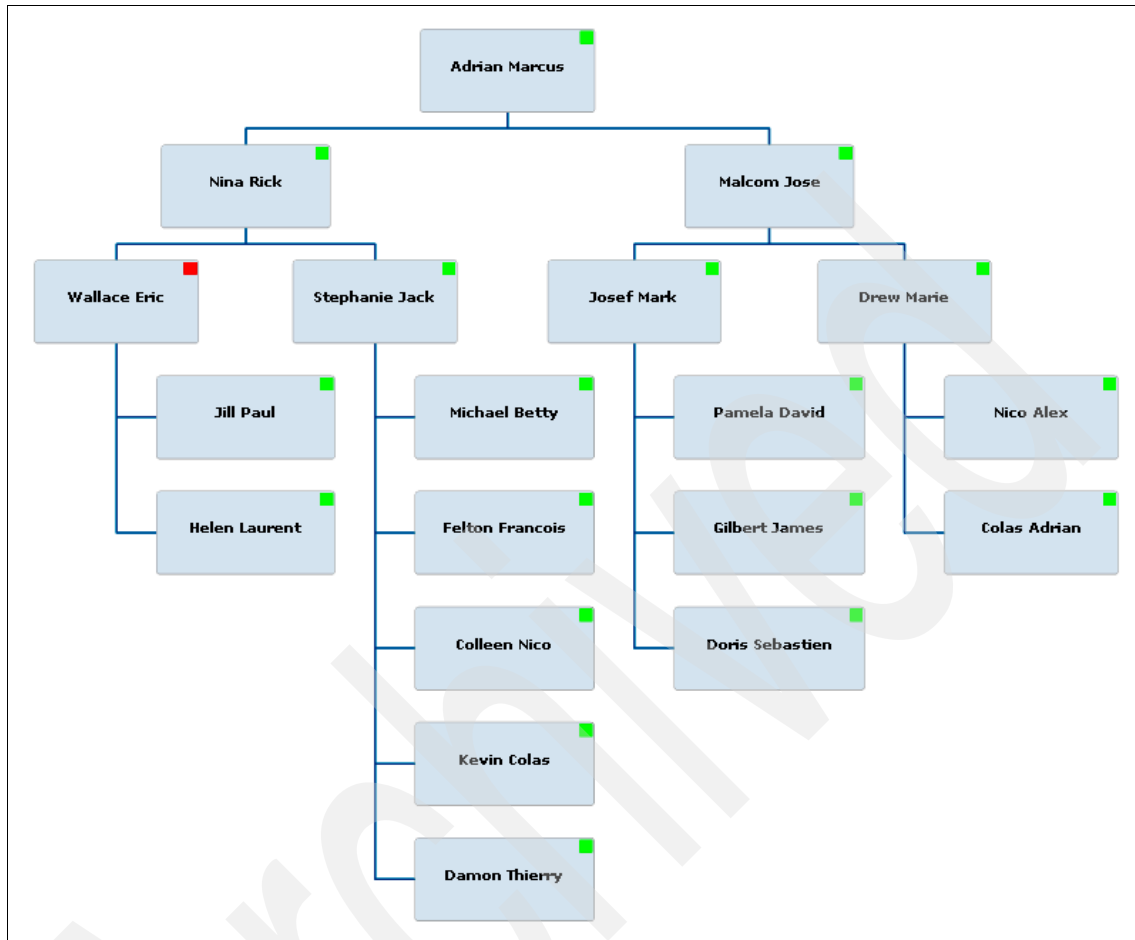


Figure 1-12 Organization chart

► Maps

Elixir makes it easy to add an intuitive map display. Areas can be colored to represent data such as sales volumes, and overlaid with Flex objects such as legends and labels. See Figure 1-13 on page 28 for an example.

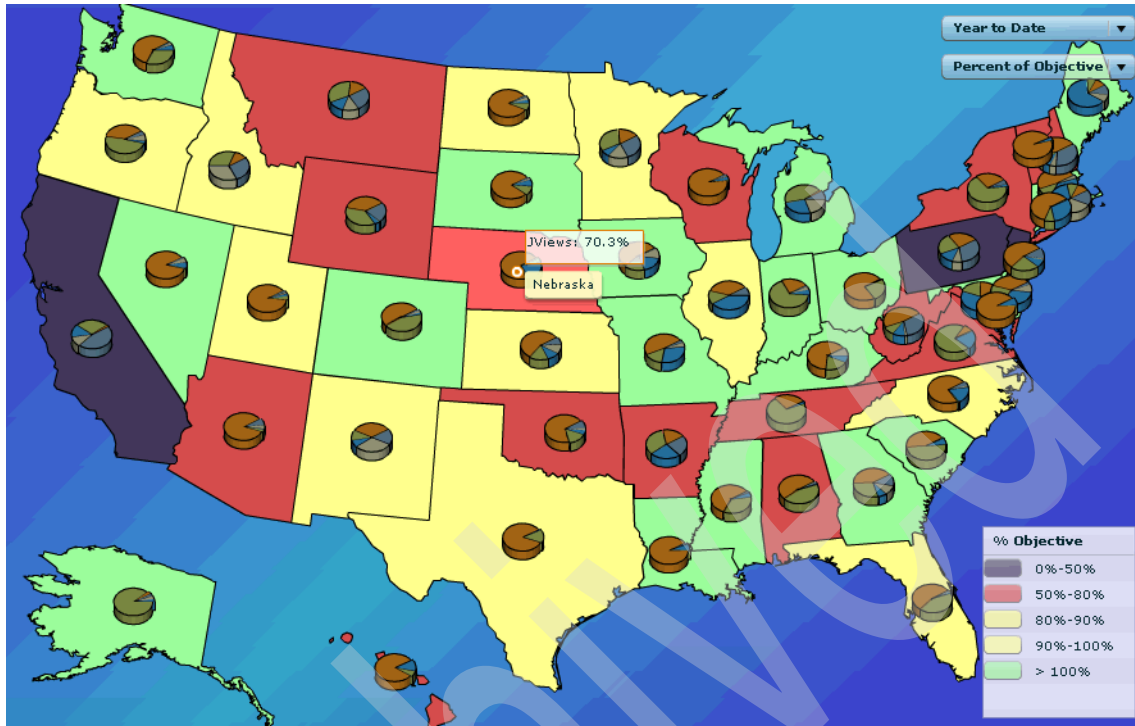


Figure 1-13 Maps

► Heat maps

Heat maps show how data is distributed throughout a region or any given 2D space, such as web pages. Elixir ships with value and density heat maps. See Figure 1-14 on page 29 for an example.

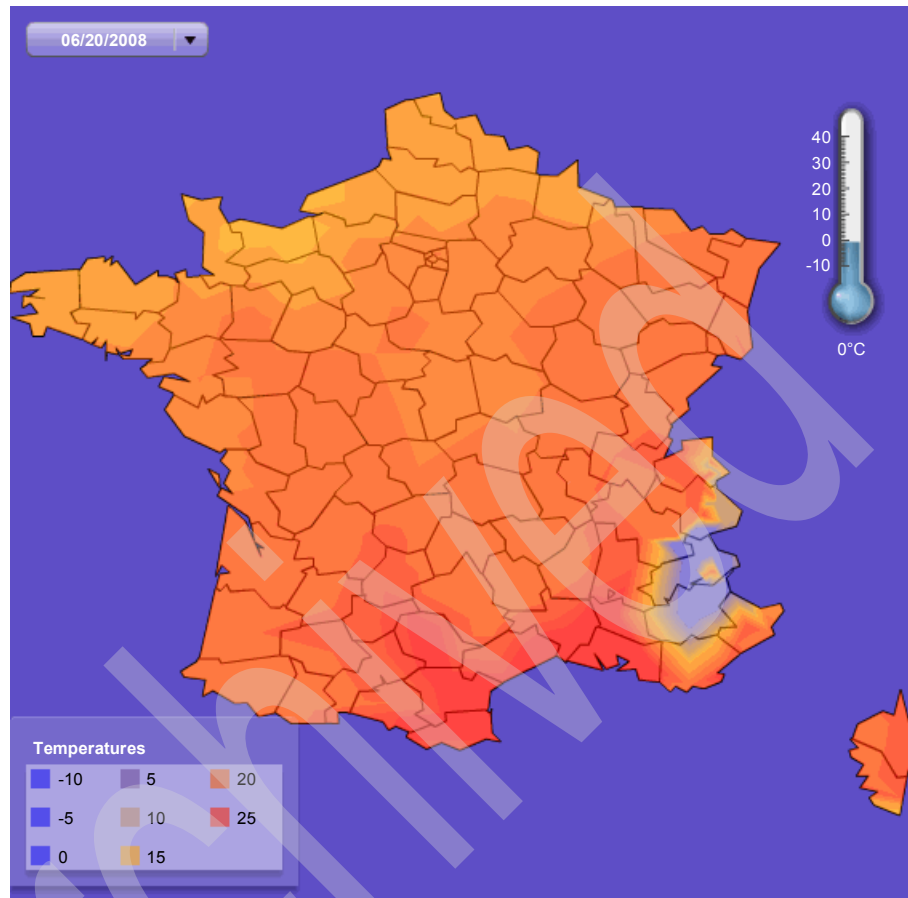


Figure 1-14 Heat Map

► Timeline

Elixir timelines helps you visualize and monitor events that occur over a span of time. See Figure 1-15 on page 30 for an example.

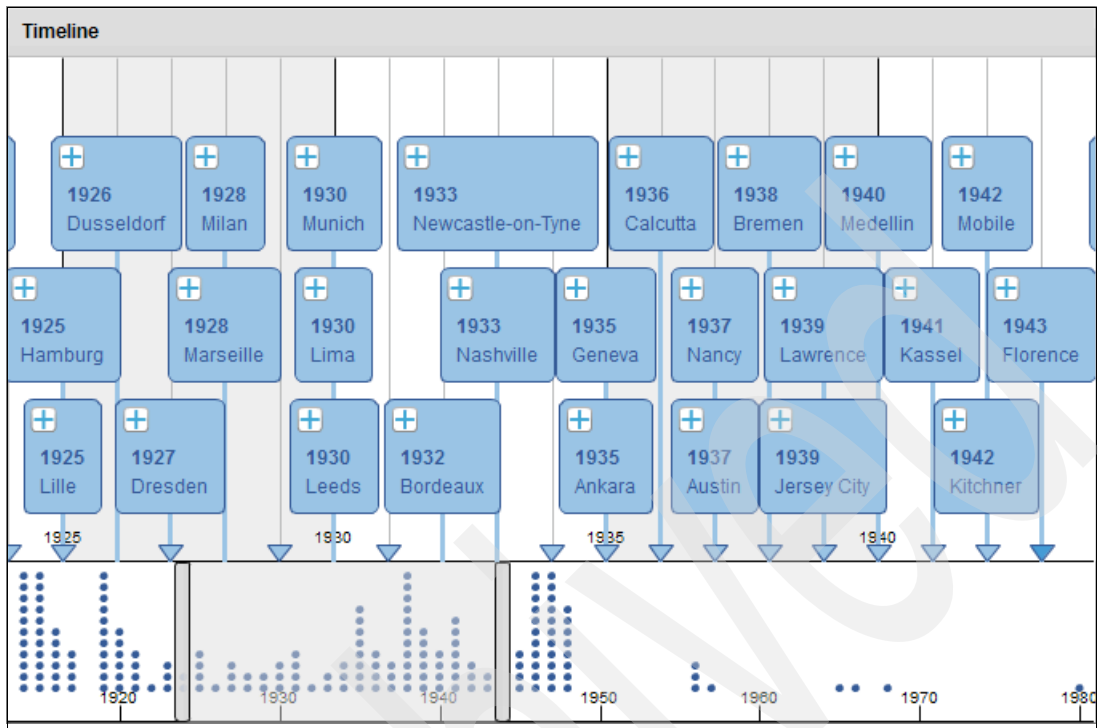


Figure 1-15 Timeline

► Calendar

Elixir's calendar component provides a full set of features, including editing, filtering, and styling events. The calendar helps users create one-time and recurring events, and it supports day, week, workweek, month, and custom views. See Figure 1-16 on page 31 for an example.

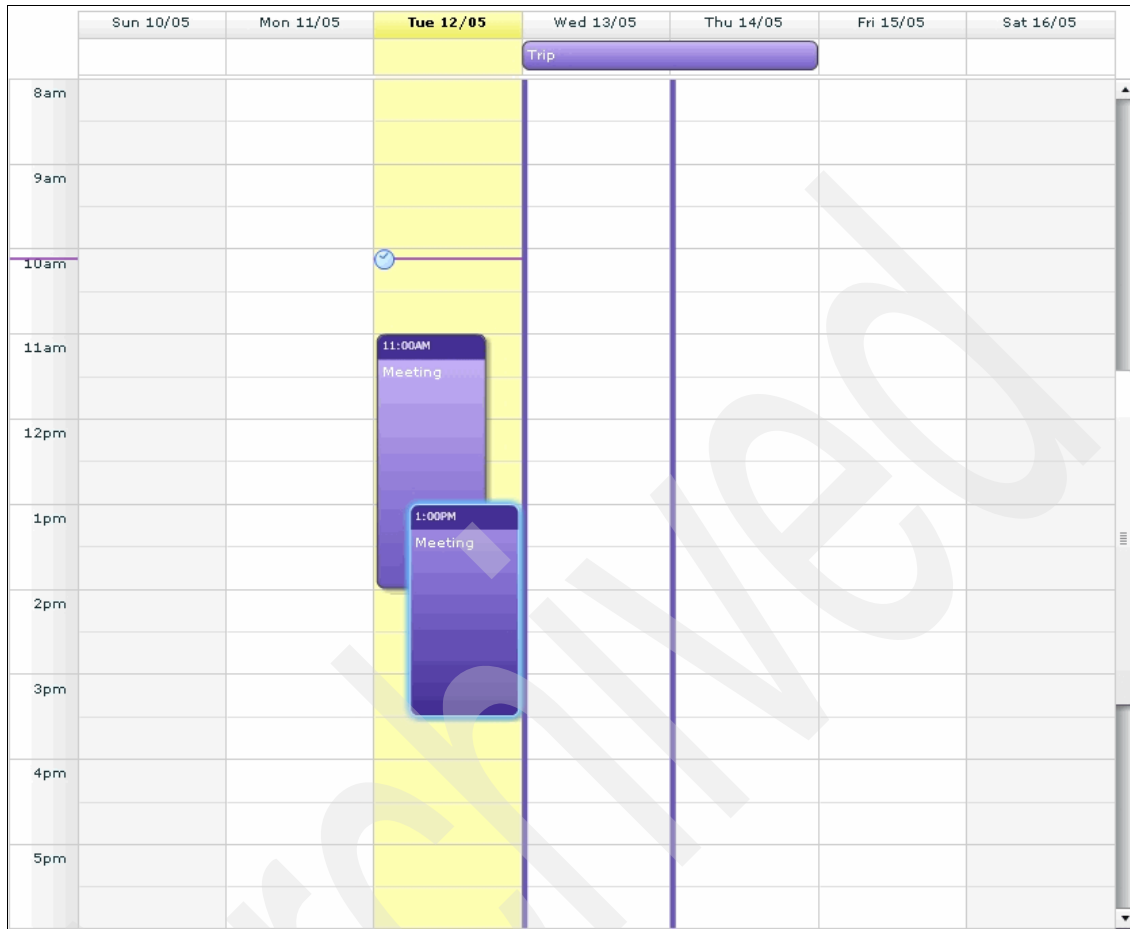


Figure 1-16 Calendar

► Resource- and Task-oriented

Elixir's Gantt task charts are used to create project management displays. Similar to Microsoft Project, they let users display and manage tasks and their dependencies. See Figure 1-17 on page 32 for an example.

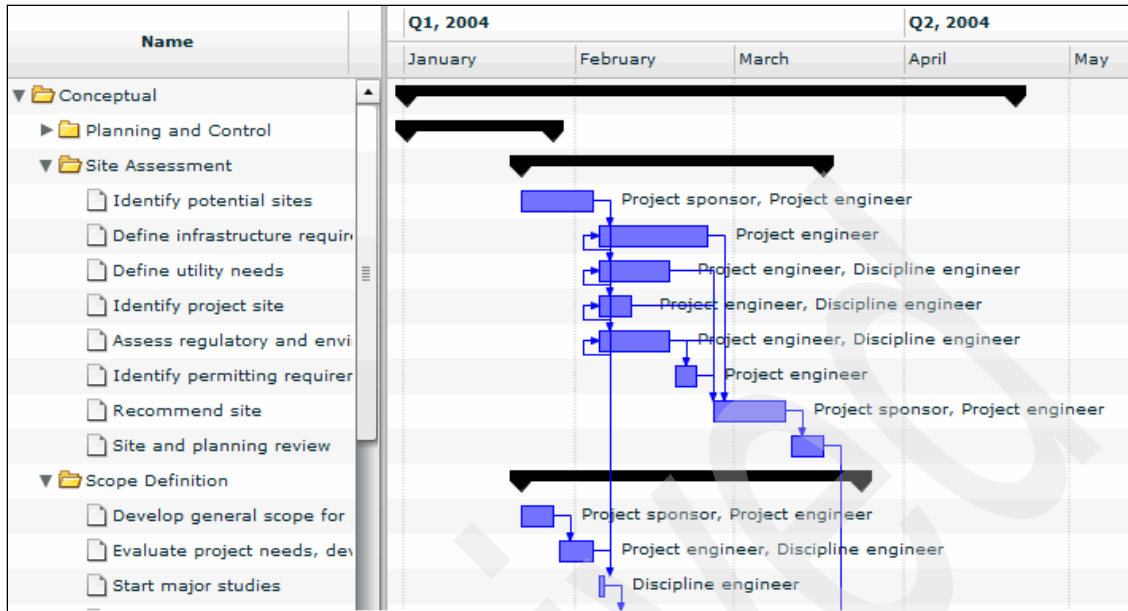


Figure 1-17 Resource- and Task-oriented chart

► Online Analytical Processing (OLAP) and Pivot charts

Elixir's OLAP and pivot charts provide interactive displays that represent and analyze data from multiple points of view. They offer complete control over data exploration, and drill-down, drag and drop, and animation functionality. See Figure 1-18 on page 33 for an example.

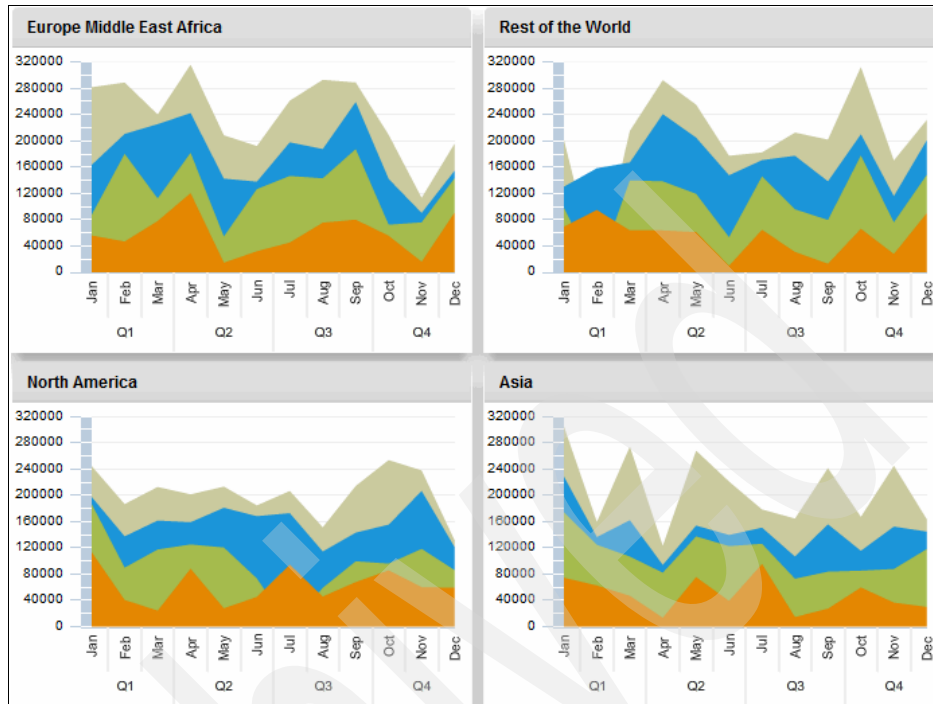


Figure 1-18 OLAP and Pivot charts

► Diagrams

The Diagram component part of the Elixir package provides components and services to build user-friendly interfaces. See Figure 1-19 on page 34 or an example.

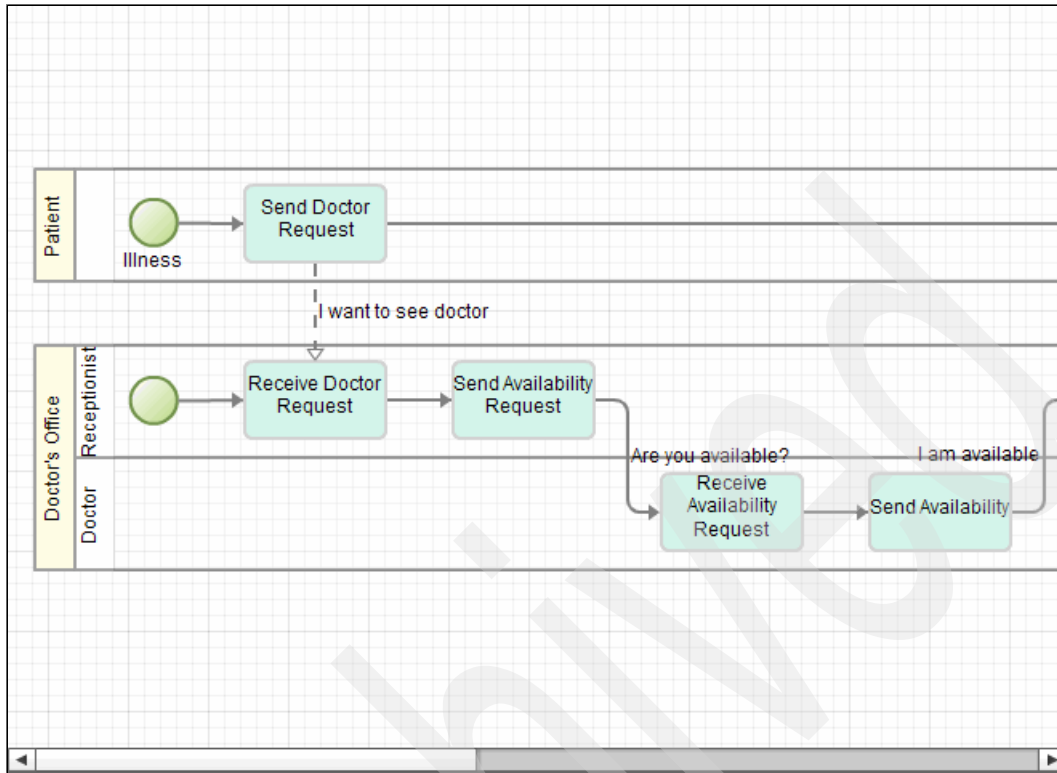


Figure 1-19 Diagram

1.3.2 Required software for creating a Flex application using IBM ILOG Elixir Enterprise

To use IBM ILOG Elixir Enterprise, you need the following software:

- ▶ Development time
 - Adobe Flex SDK 4.0
 - Optional: Adobe Flash Builder 4 or later
- ▶ Run time
 - Adobe FlashPlayer 10.0.12.36 or later, or Adobe AIR Runtime 1.5.3

Optionally, communication with the server can be done by using Adobe Livecycle Data Services ES. However IBM ILOG Elixir Enterprise components can communicate with the server using any standard protocol and format such as JSON or XML over HTTP.

1.3.3 Deployment options available for IBM ILOG Elixir components

Like any other Adobe Flex project, IBM ILOG Elixir Enterprise applications can be deployed on the web (run in FlashPlayer) or on Desktops (run in Adobe Integrated Runtime - AIR).

Adobe FlashPlayer 10.0.12.36 or later is required for web deployment and Adobe AIR 1.5.3 or later is required for Desktop deployment.

If your application makes use of many IBM ILOG Elixir Enterprise features, it is possible to use a Runtime Shared Library (RSL) to store the entire library in a cache on the client browser.

1.4 Scenarios for integration with other IBM software

The IBM ILOG Visualization suite creates components that are meant to be incorporated into existing or new applications. IBM ILOG Visualization components do not force the developer to use any specific software architecture or any specific data. It is the role of the developer to adapt the components to the architecture and data sources of its application.

The purpose of this book is to help software developers who would like to incorporate the IBM ILOG Visualization components with other IBM products that provide highly interactive web based interfaces or dashboards. This book certainly does not cover all the possible integration scenarios, but tries to cover scenarios where using advanced visualization components would be of the most benefit.

In this book, we have chosen to show step-by-step instructions to succeed in the following integration scenarios:

- ▶ Using WebSphere REST Technology with IBM ILOG Visualization products

In this scenario, you will see how IBM ILOG Visualization products can be used to visualize data retrieved from a RESTful web service.

- ▶ Using IBM ILOG Visualization products and IBM Cognos for dynamic and interactive web visualization

In this scenario, you will see how to create IBM ILOG JViews and IBM ILOG Elixir views from an IBM Cognos report using the Cognos Mashup Service (CMS).

- ▶ Creating IBM ILOG Elixir iWidgets for Mashup Center

In this scenario, you will see how an IBM ILOG Elixir component can be wrapped as an iWidget to allow for deployment into IBM Mashup Center.

- ▶ Business Monitoring with IBM ILOG Elixir, WebSphere Business Monitor and Business Space

In this scenario, you will see how IBM ILOG Elixir Enterprise components can be used inside the Business Space environment to provide new ways to visualize business data.

- ▶ Integration of ILOG JViews JSF components inside IBM WebSphere Dashboard Framework

In this scenario, you will see the integration of ILOG JViews JSF components into WebSphere Dashboard Framework.

Using WebSphere REST Technology with ILOG Visualization products

This chapter describes how ILOG Visualization products can be used to visualize data retrieved from a RESTful service.

The chapter uses concrete examples to demonstrate the integration scenarios. It describes creating a RESTful service, and shows how you can use this service as a data provider for ILOG Elixir and ILOG JViews Diagrammer components.

2.1 Overview of REST

Representational State Transfer (REST) is an architectural style for exposing resources over distributed hyper media systems such as the World Wide Web. The term was first introduced in 2000 by Roy T. Fielding in his doctoral dissertation and came into wide use since then.

REST strictly refers to a collection of network architecture principles that outline how resources are defined and addressed. The term is often used in a looser sense to describe any simple interface that transmits domain-specific data over HTTP without an additional messaging layer such as SOAP or session tracking by way of HTTP cookies.

2.1.1 Basic concepts

This section describes the basic concepts of the REST architecture.

Client server architecture

REST architecture is based on the client-server model, separating the user interface layer from the business/data storage layer. This simplifies component implementation, reduces the complexity of connector semantics, improves the effectiveness of performance tuning, and increases the scalability of pure server components. Layered system constraints allow intermediaries such as proxies, gateways, and firewalls to be introduced at various points in the communication without changing the interfaces between components.

Stateless

All Interactions between client and server are stateless, which means each request is self contained and has all the information needed by the server to understand and fulfill it. Servers does not hold any session state information: all of this information is maintained by the client.

Resource model

One important concept in REST is the existence of resources, which are sources of specific information, Each resource has a name and can be referred to using a global identifier (a URI). In order to manipulate these resources, clients connect to servers through a Uniform interface (HTTP) and exchange representations of these resources (the actual documents conveying the information).

Clients use basic HTTP methods POST, GET, PUT, and DELETE to perform CRUD operations on resources of a REST service.

For example, in table Table 2-1 we show how can we use REST to expose and manipulate books of a library.

Table 2-1 A REST implementation for a library service

Operation	URI	HTTP Method
Create book	../book	POST
Retrieve book	../book/{bookId}	GET
Update book	../book/{bookId}	PUT
Delete book	../book/{bookId}	DELETE
List all	../book	GET

This example creates a RESTful service with only one resource, which is book. The service provides the client with all of the CRUD operations. In order to invoke an operation the client has to access the correct URI using the correct HTTP method.

Note: A service that implements REST architecture is called **RESTful**.

2.1.2 Benefits of using REST

REST is receiving increasing interest from current Internet architects for the following reasons:

- ▶ REST divides application functionality into resources. Every resource is uniquely addressable using a universal syntax.
- ▶ All resources share a uniform interface for the transfer of state between client and resource, consisting of:
 - A constrained set of well-defined operations
 - A constrained set of content types
- ▶ Modern Internet browsers have the mechanics to invoke and consume asynchronous REST requests with no additional coding.
- ▶ REST payloads can contain several MIME types ranging from XML to JSON to TEXT.
- ▶ REST provides improved response times and server loading characteristics due to support for caching.
- ▶ Server scalability is improved by reducing the need to maintain communication state. This means that separate servers can be used to handle initial and subsequent requests.
- ▶ REST requires less client-side software to be written than other approaches because a single browser can access any application and any resource.

- ▶ REST depends less on vendor software than mechanisms that layer additional messaging frameworks on top of HTTP.
- ▶ Equivalent functionality is provided when compared to alternative approaches to communication.
- ▶ REST does not require a separate resource discovery mechanism due to the use of hyperlinks in content.
- ▶ Better long-term compatibility and evolvability characteristics than RPC are provided because of the following reasons:
 - The capability of document types such as HTML to evolve without breaking backwards-compatibility or forwards-compatibility
 - The ability of resources to add support for new content types as they are defined without dropping or reducing support for older content types.

2.2 Designing a RESTful service

In this section, we present a scenario and discuss the design of a RESTful service to implement that scenario.

2.2.1 A RESTful service scenario

ITSO Airlines is a well established airline. They have an existing flight route management system that is used on a daily basis to keep track of flight routes between major cities. The management system was developed as a Java desktop application. Check the class diagram in Figure 2-1 on page 41 to get a better understanding of the original system.

As a part of the organization initiative to move to Service Oriented Architecture (SOA), all data and functionality in the flight route system will be exposed over the network so that they can be used by users remotely or by other services. The architect chose to expose them using a RESTful service hosted on WebSphere Application Server.

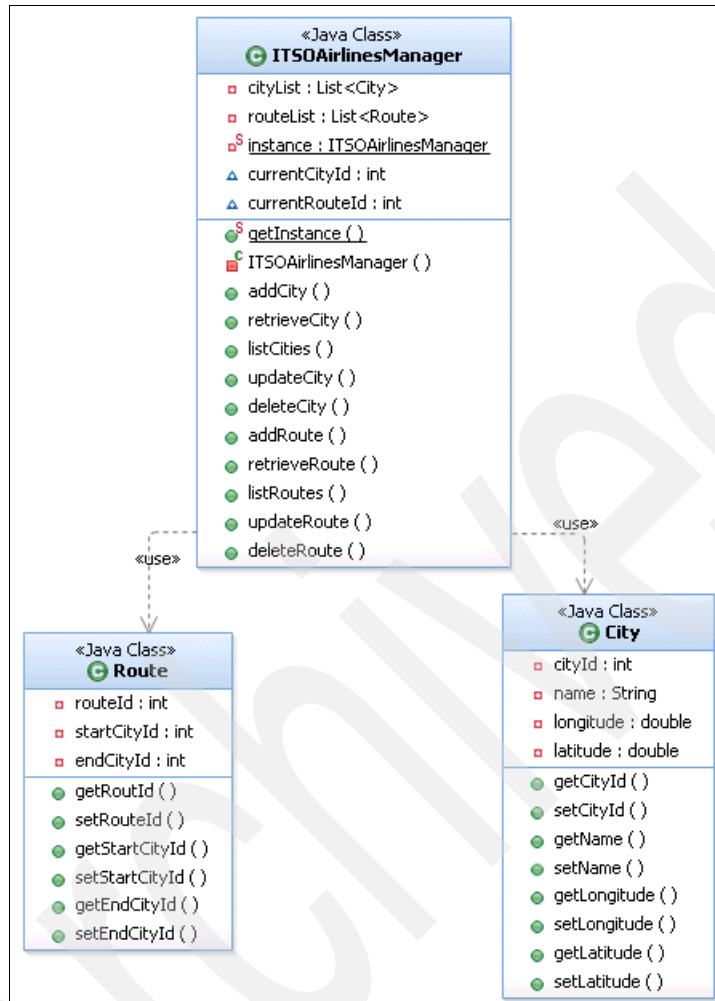


Figure 2-1 Class diagram for ITSOAirlines original system

2.2.2 Defining the service resources

The first step in designing a RESTful service is to define the resources and operations provided by this service. For ITSOAirlines the resources are summarized in Table 2-2 on page 42

Table 2-2 ITSOAirlines service resources

Resource	URLI	Property	Description
City	../city	CityId	A unique ID for each city in the system
		Name	Name of the city
		Longitude	City longitude
		Latitude	City latitude
Route	../route	RouteId	A unique ID for each route created on the system
		StartCityId	ID of the starting city
		EndCityId	ID of the end city

Basically we have two resources: the City resource that represents a city on the air route, and the Route resource that represents an air way between two cities. The resources need to perform the following operations:

1. The City operation interface

The City operation interface allows a user to do Create and Retrieve operations, and to list all cities. In the future, this could be expanded to provide filtering and querying capabilities. Table 2-3 shows the operations map for this interface.

Table 2-3 City interface RESTful API

Operation	HTTP method	URLI	params
List all	../city	GET	None
Delete	../city/{cityId}	DELETE	ID of the city to delete
Retrieve	../city/{cityId}	GET	ID of the city to retrieve

2. The Route operation interface

The Route operation interface allows a user to do basic CRUD operations, and to retrieve a list of all routes. Table 2-4 on page 43 shows the operations map for this interface.

Table 2-4 Route interface RESTful API

Operation	HTTP method	URL	params
Create	../route	POST	a route object in the POST data
Retrieve	../route/{routeId}	GET	ID of the route to retrieve
Delete	../route/{routeId}	DELETE	ID of the route to delete
List all	../route	GET	None

2.2.3 Defining the data format

Choose a data format for the data coming from clients, and data being delivered by the server. The REST protocol allows for a client and server to define the data formats accepted by each component.

For ITSOAirlines, data between the client and server will be represented in JSON format. See Example 2-1 for a sample response.

Example 2-1 Sample JSON response

```
{
  id : "CITY004",
  name : "Miami",
  longitude: "-80.28",
  latitude : "45.82"
}
```

2.3 Implementing a RESTful service

In this section, we discuss the steps to implement a RESTful service in WebSphere Application Developer 8.0 based on the design outlined in 2.2, “Designing a RESTful service” on page 40. The service will use the Java API for RESTful Web Services (JAX-RS).

JAX-RS is a Java API for developing REST applications quickly. This standard API continues to gain support throughout the Java community. Although JAX-RS provides a faster way of developing web applications than servlets, the primary goal of JAX-RS is to build RESTful services. JAX-RS 1.0 defines a server-side

component API to build REST applications. IBM JAX-RS provides an implementation of the JAX-RS (JSR 311) specification.

IBM JAX-RS libraries are available in WebSphere Application Server 8.0 and in WebSphere Application Server 7.0 with Web 2.0 Feature Pack installed.

Note: The complete source code for all the samples provided in this chapter is available in Project Interchange format. It is provided with the additional material for this book. See Appendix A, “Additional material” on page 287 for more detail. The source is included in the folder /Chapter 2/CompletedExamples/.

2.3.1 Software requirements

In order to complete the next steps, you will need the following software to be installed:

- ▶ IBM WebSphere Application Server 7.0 with Web 2.0 Feature Pack installed.
- ▶ IBM Rational® Application Developer 8.0.

2.3.2 Implementing the service

Creating WebSphere Application Server

Before creating the web service, ensure that you have a server that has Java 5.0 or later JVM support defined and started. By default a server is created for you when you install WebSphere Application Server. This server can be seen in the Servers view. However, if you want to create a new WebSphere Application Server, perform the following steps:

1. In Rational Application Developer click **File** → **New** → **Other** → **Server** → **Server** → **Next**.
2. Select **WebSphere Application Server v7.0** as the server type. Click **Next** as in Figure 2-2 on page 45.

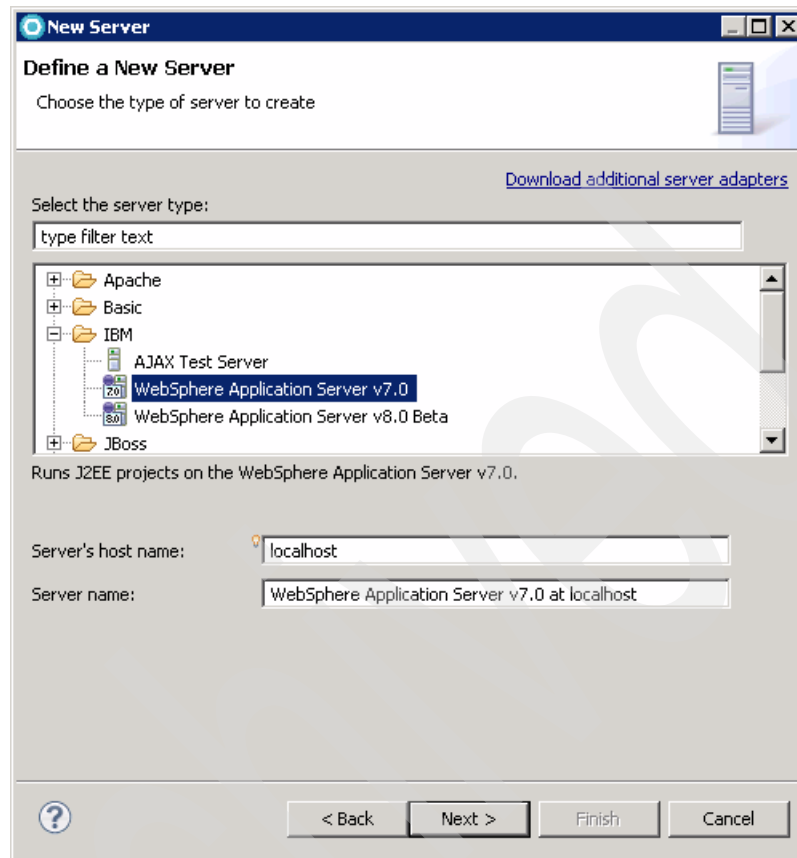


Figure 2-2 Create a new server

- Optional: If this runtime environment has not been created in your workspace, you will be prompted to select the installation directory for the server as in Figure 2-3 on page 46. Click **Next**.

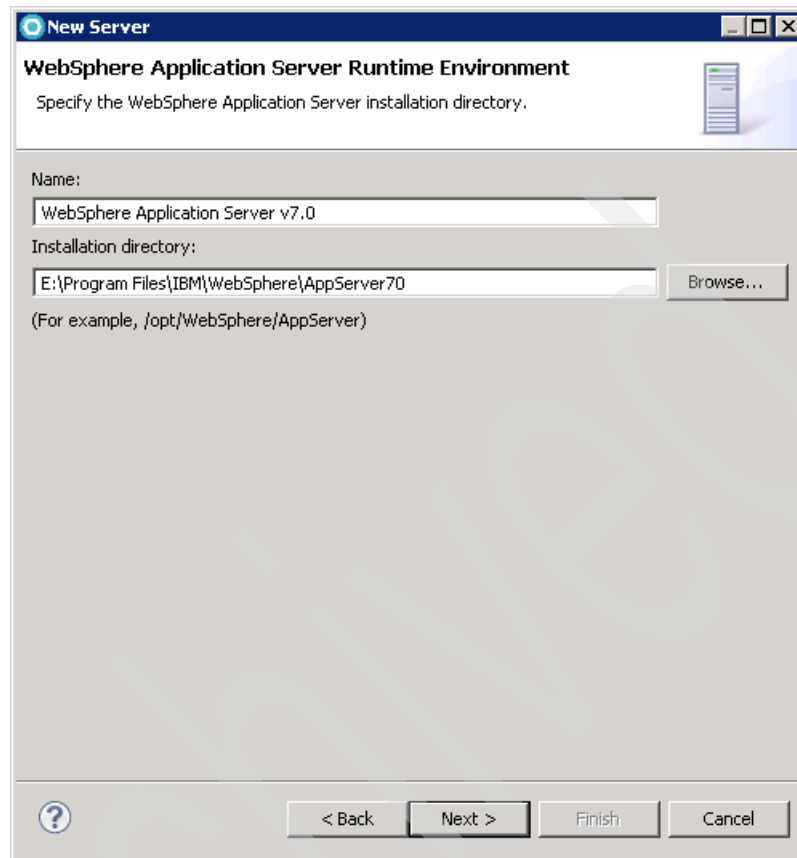


Figure 2-3 Define server runtime environment

4. Accept the default server settings and click **Finish**.
5. Wait for the server to start. After it has started, the Console view will display **Server server1 open for e-business**. If the server does not start automatically, select it in the Servers view and click the **Start** icon.

Service implementation

The following steps will guide you to create and configure a RESTful service project for WebSphere Application Server using Rational Application Developer:

1. Click **File** → **New** → **Dynamic Web Project**.
2. In the New Dynamic Web Project window, type **ITS0Airlines_Service** in the project name field. For the Target Runtime select **WebSphere Application Server v7.0**, and for the Configuration select **IBM JAX-RS Configuration** as

shown in Figure 2-4. Make sure the box next to **Add project to an EAR** is selected.

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: ITSOAirlines_Service

Project location
☒ Use default location
Location: C:\RSA_Workspaces\JJAirlines_TEMP\ITSOAirlines_Service Browse...

Target runtime
WebSphere Application Server v7.0 New Runtime...

Dynamic web module version
2.5

Configuration
Default Configuration for WebSphere Application Server v7.0 Modify...
A good starting point for working with WebSphere Application Server v7.0 runtime. Additional facets can later be installed to add new functionality to the project.

EAR membership
☒ Add project to an EAR
EAR project name: ITSOAirlines_ServiceEAR New Project...

Working sets
☐ Add project to working sets
Working sets: Select...

? < Back Next > Finish Cancel

Figure 2-4 Create a new Dynamic Web Project

3. Click **Next** until you reach JAX-RS Capabilities window (Figure 2-5 on page 48). In the JAX-RS Implementation Library box, select **IBM WebSphere JAX-RS Library for WAS 7.0**, select the box next to **Include library with this application**, select **Shared Library**, and click **Finish**.

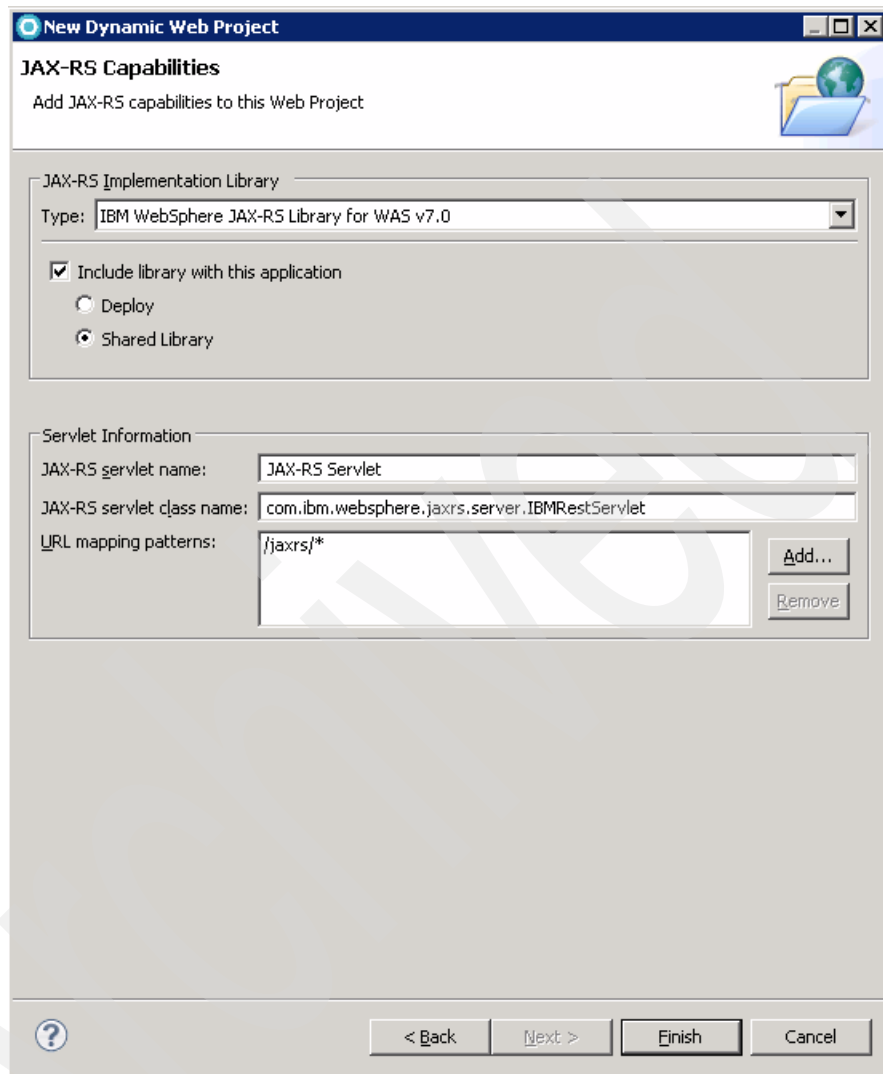


Figure 2-5 JAX-RS Capabilities window

4. You need to import the ITSOService application in your workspace. Click **File** → **Import**, select **Existing Projects into Workspace** and click **Next**. Click **Select archive file**, then click **Browse** and navigate to the directory where you extracted the additional material for this book. Select the file **ITS0Airlines.zip** under the sub directory **/Chapter 2/legacy/**. Click **Finish**.
5. Now you need to reference the original project from your service project and add the original project to be a part of the deployment of the service project:

- a. Right-click the project **ITSOAirlines_Service**, click **Properties** → **Java Build Path** → **Projects**, and click **Add**.
- b. Select project **ITSOAirlines** and click **OK**.
- c. From the properties on the left click **Deployment Assembly**. You will be prompted to save your changes. Click **Apply**.
- d. Click **Add**, select **Project** and click **Next**.
- e. Select **ITSOAirlines** and click **Finish**.
- f. Click **OK** to close the project properties window.

In the next steps you will create the handler classes for the application resources City and Route.

1. Right-click the project **ITSOAirlines_Service** and click **New** → **Class**. Type `itsoairlines.routing.service` in the package field and `CityHandler` as the class name as shown in Figure 2-6 on page 50. Click **Finish**.

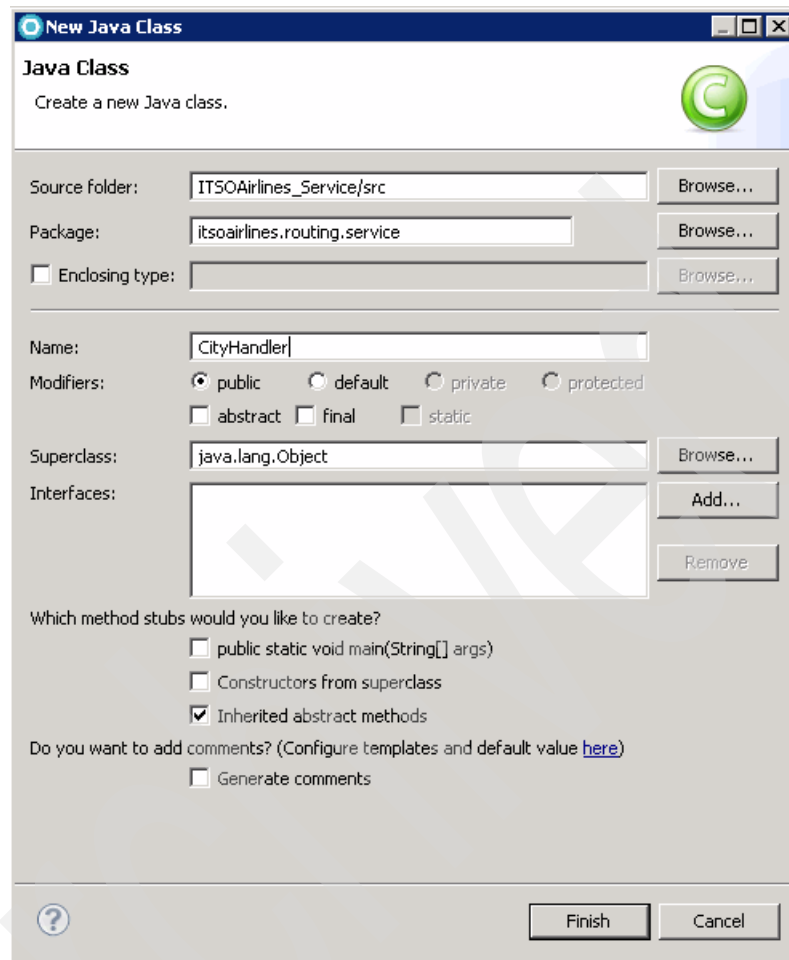


Figure 2-6 Create class CityHandler

2. Create a default constructor for the CityHandler class and create a member variable of type ITSOAirlinesManager as in Example 2-2.

Example 2-2 Create a default constructor

```
public class CityHandler {

    public CityHandler() {
    }

    ITSOAirlinesManager manager = ITSOAirlinesManager.getInstance();
    ...
}
```

3. Define a relative URI for the City resource. This is done by configuring the Path annotation just before the class name as shown in Example 2-3.

Example 2-3 Using the Path annotation

```
@Path(value = "/city")
public class CityHandler {
    ...
}
```

4. Add JSON support to your project. Right-click **ITSOAirlines_Service** and select **Properties** → **Java Build Path** → **Libraries** → **Add External JARs**. Browse to the directory `WAS_INSTALLATION_DIRECTORY\web2fep\optionalLibraries\JSON4J` and select `JSON4J.jar`. Click **OK** to close the Properties window.
5. Now you are ready to write the code that converts the City java object to JSON. See the code snippet in Example 2-4 for an example.

Example 2-4 Convert City object to JSON

```
private JSONObject cityToJSON(City city) {
    if (city != null) {
        JSONObject json = new JSONObject();
        json.put("cityId", city.getCityId());
        json.put("name", city.getName());
        json.put("longitude", city.getLongitude());
        json.put("latitude", city.getLatitude());
        return json;
    } else {
        return null;
    }
}
```

6. Write the code that handles the List operation for the City resource as in Example 2-5. The `@GET` annotation is used to instruct the server that this method handles HTTP GET methods on the City resource. The `@Produces` annotation declares the response type for this method.

Example 2-5 List all Cities

```
@GET
@Produces("application/json")
public JSONArray handleList() {
    List<City> cityList = manager.listCities();
    System.out.println(cityList.size());
    JSONArray jsonArr = new JSONArray();
}
```

```

        for (Iterator<City> cityItr = cityList.iterator();
cityItr.hasNext();) {
            City city = cityItr.next();
            jsonArr.add(cityToJSON(city));
        }
        System.out.println(jsonArr.size());
        return jsonArr;
    }

```

7. Likewise, use the code snippet in Example 2-6 to write the methods handling retrieve and delete operations on the City resource.

Example 2-6 Retrieve and delete operations

```

@GET
@Path(value =("/{cityId}")
@Produces("application/json")
public JSONObject handleRetrieve(@PathParam("cityId") int cityId)
{
    return cityToJSON(manager.retrieveCity(cityId));
}

@DELETE
@Path(value =("/{cityId}")
public void handleDelete(@PathParam("cityId") int cityId) {
    manager.deleteCity(cityId);
}

```

8. Repeat the previous steps to create class RouteHandler, which is the handler class for the Route resource. You can use the file RouteHandler.java as a reference for creating your handler. This file comes with the additional material for this book in directory /Chapter 2/rest. Refer to Appendix A, “Additional material” on page 287 for more information.

In the next steps you will create the `javax.ws.rs.core.Application` subclass to define in the JAX-RS runtime environment which classes are part of the JAX-RS application. The resource classes are returned in the `getClasses()` method.

1. Right-click the project **ITSOAirlines_Service** and click **New** → **Class**. Type `itsoairlines.routing.service` in the package field and `ITSOAirlinesApplication` as the class name. In the Super Class field type `javax.ws.rs.core.Application` as shown in Figure 2-7 on page 53. Click **Finish**.

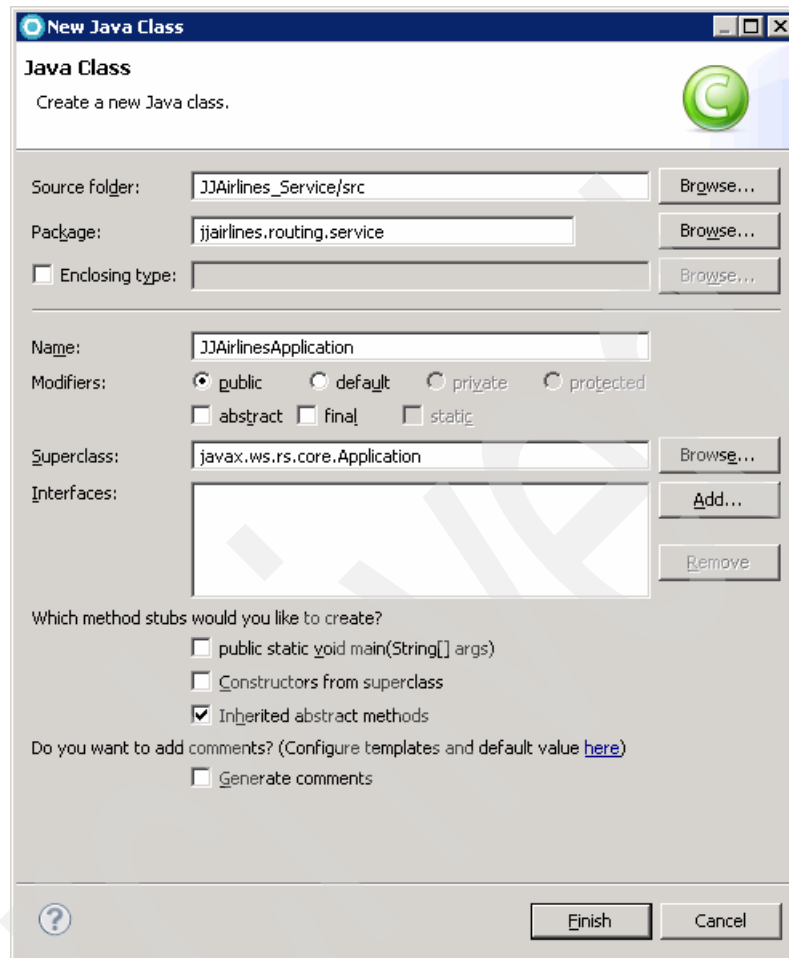


Figure 2-7 Create the Application class

2. Use the code in Example 2-7 to create your Application class.

Example 2-7 Class ITSOAirlinesApplication

```
package itsoairlines.routing.service;

import java.util.HashSet;
import java.util.Set;
import javax.ws.rs.core.Application;

public class ITSOAirlinesApplication extends Application {
```

```

@Override
public Set<Class<?>> getClasses() {

    Set<Class<?>> classes = new HashSet<Class<?>>();
    classes.add(CityHandler.class);
    classes.add(RouteHandler.class);
    return classes;
}
}

```

3. Open WebContent/WEB-INF/web.xml. In the Design view as in Figure 2-8, select the Servlet **JAX-RS Servlet** and click **Add**. Add an Initialization parameter to the JAX-RS servlet, leaving the name field empty.

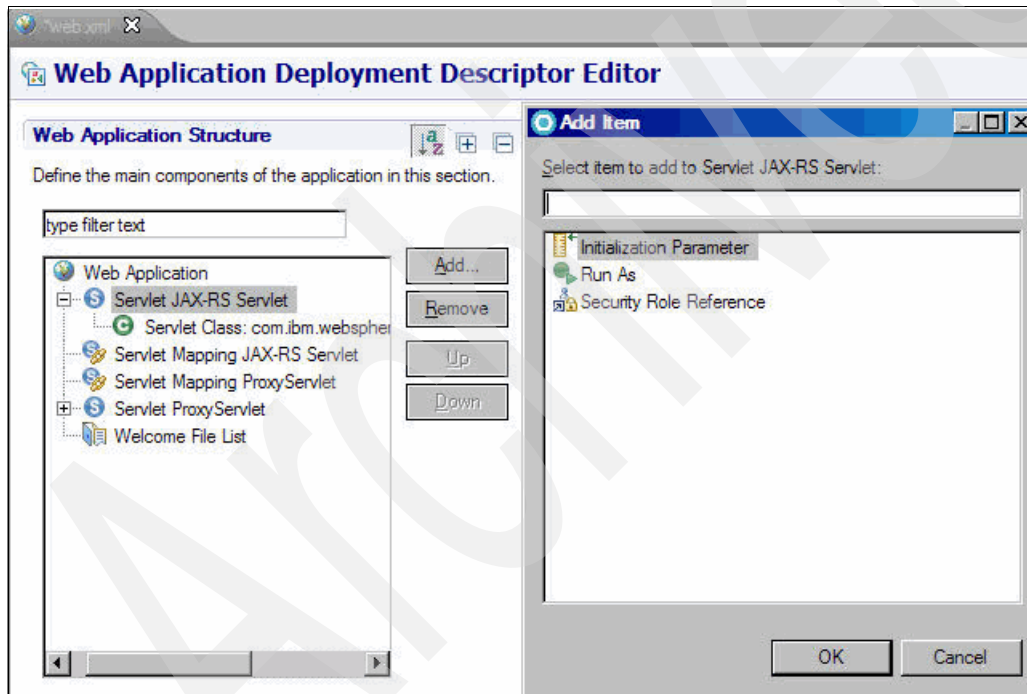


Figure 2-8 Add an initialization parameter

4. Save **web.xml**, ignoring any errors that might be displayed.
5. In the Problems view, right-click **the JSR-311 warning** stating that the param-name should be javax.ws.rs.Application, and select **Quick Fix** as shown in Figure 2-9 on page 55.

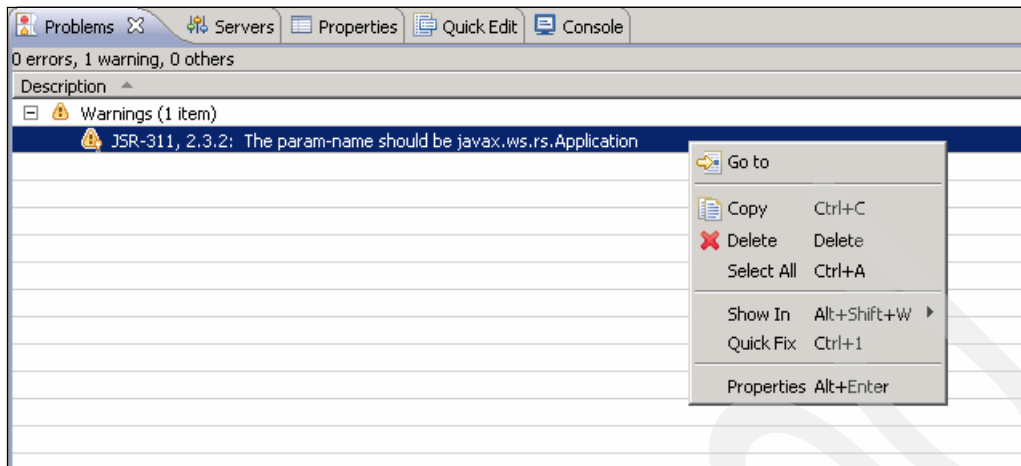


Figure 2-9 Resolve JSR-311 warning

6. Select **Browse for an existing JAX-RS Application sub-class** and set the **param-value** in the **web.xml** and click **Finish**.
7. Select the class **ITSOAirlinesApplication** and click **OK** as shown in Figure 2-10 on page 56.

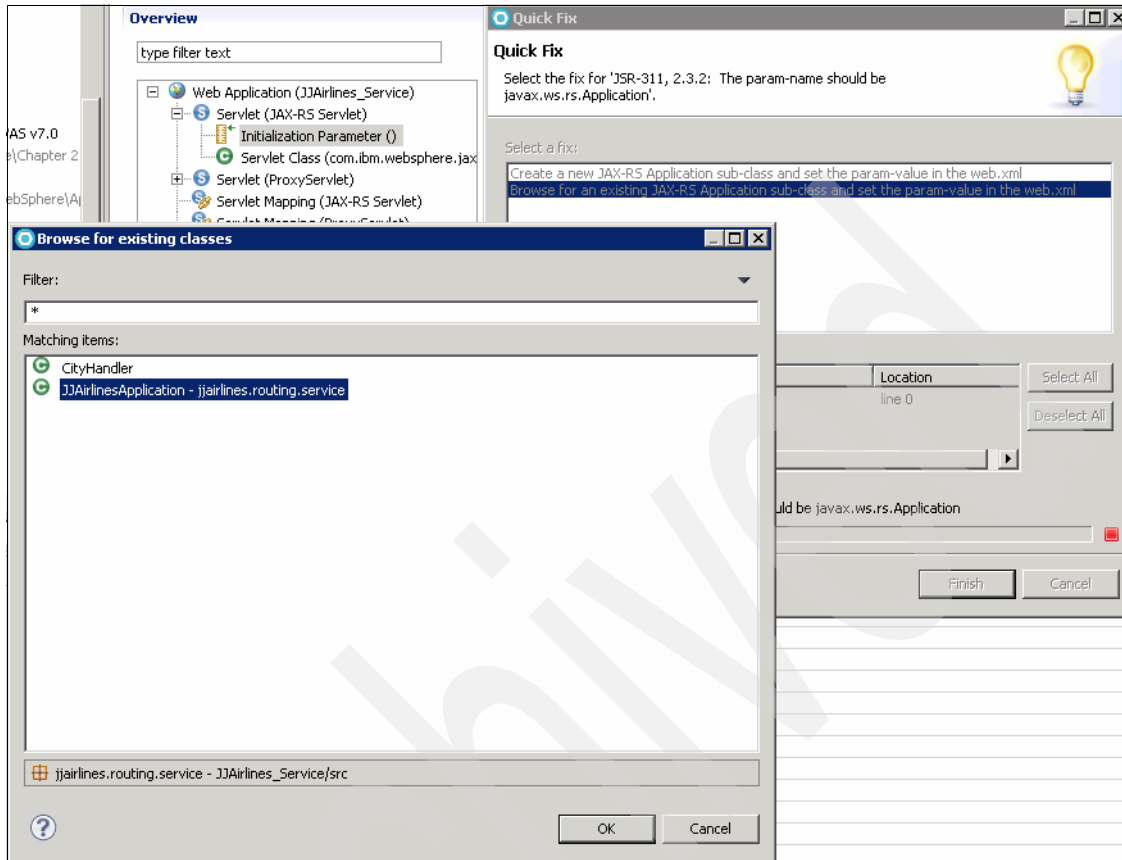


Figure 2-10 Select the correct Application sub-class

2.3.3 Deploying and starting the service

In this section, we discuss the steps to deploy a RESTful service in Rational Application Developer based on the service implemented in 2.3.2, “Implementing the service” on page 44.

When we create the project for the service, we create a Dynamic Web Project and an EAR Application Project. We need to deploy that EAR.

Perform the following steps to deploy the EAR file:

1. In the Servers window, right-click your **WebSphere Application Server 7.0** and select **Add and Remove Projects**.
2. In the Available projects list, select the **ITSOAirlines_ServiceEAR** project and click **Add**. See Figure 2-11 on page 57. Click **Finish**.

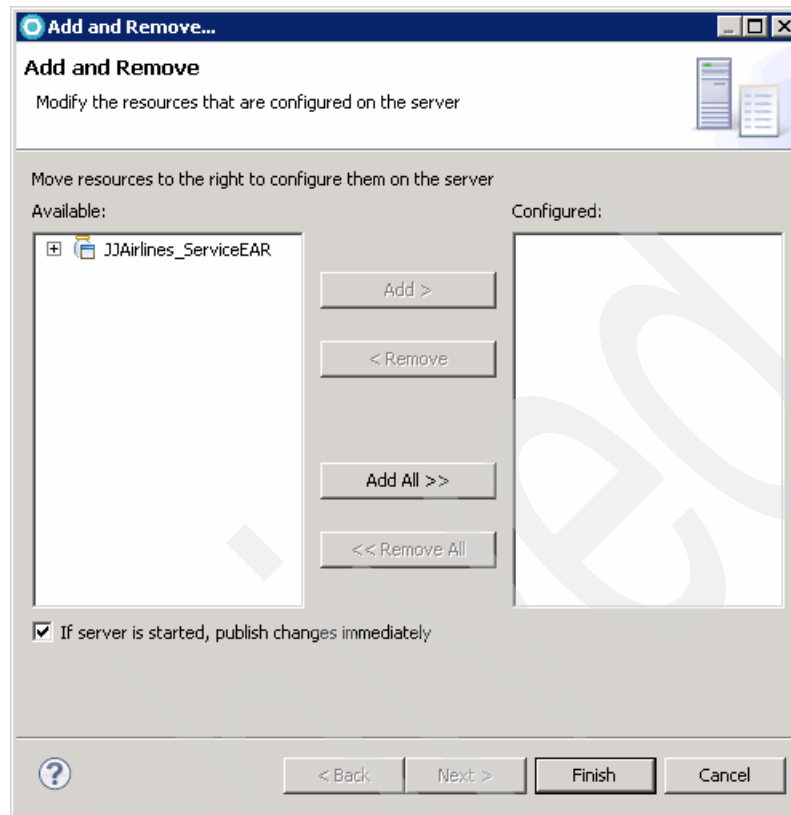


Figure 2-11 Deploy the EAR file to WebSphere Application Server 7.0

3. After the Publish step completes, your service will be deployed and started.

2.4 Using the IBM ILOG Elixir Enterprise product to visualize data from a RESTful service

In this section you are going to use the ILOG Elixir Diagrammer component to draw a user friendly diagram for all of the cities and routes served by ITSOAirlines.

2.4.1 Overview

The IBM ILOG Elixir Enterprise product consists of a family of related components that let you design, develop, and deploy an entirely new class of

Rich Internet applications (RIA). The IBM ILOG Elixir components conform to the Adobe Flex platform and existing Adobe Flex component paradigms. Each component complies with Adobe Flex component architecture.

The Diagram component is a data-driven component that displays diagrams composed of nodes and links. In the Diagram component, a diagram is an instance of the `Diagram` class that is able to display nodes and links, creating their graphical representation based on business data. Nodes are specified with the `nodeDataProvider` property, whereas links are specified with the `linkDataProvider` property. With the Diagram component, you can lay out objects by using graph layout algorithms or by specifying their position according to properties in the business data objects. Diagrams also provide support for the select, move, zoom, pan, and scroll operations, and for customized interactions such as editing and drill-down.

2.4.2 Software requirements

To complete the next steps, you need to have the following software installed:

- ▶ IBM ILOG Elixir Enterprise 3.0
- ▶ Adobe Flash Builder 4.0
- ▶ Firefox 3.x with Adobe Flash Player add-on.

2.4.3 Integration scenario

The diagram in Figure 2-12 on page 59 explains the whole integration scenario:

- ▶ When the web page is loading, the Flex framework initializes the diagram component and calls the function **initializeHandler**.
- ▶ **initializeHandler** takes control and calls the functions **retrieveCities** and **retrieveRoutes**.
- ▶ The `retrieveXXXX` method initiates an asynchronous call to the ITSOAirlines service we created.
- ▶ The appropriate resource handler at ITSOAirlines service is invoked to handle the request, retrieving data from the original ITSOAirlines application and wrapping it in JSON format, and sending the response back.
- ▶ Flex framework receives response for the asynchronous call and calls the appropriate callback function to handle the response.
- ▶ The callback function `parseXXXX` parses the JSON response, creates cities and routes and populates the diagram component.

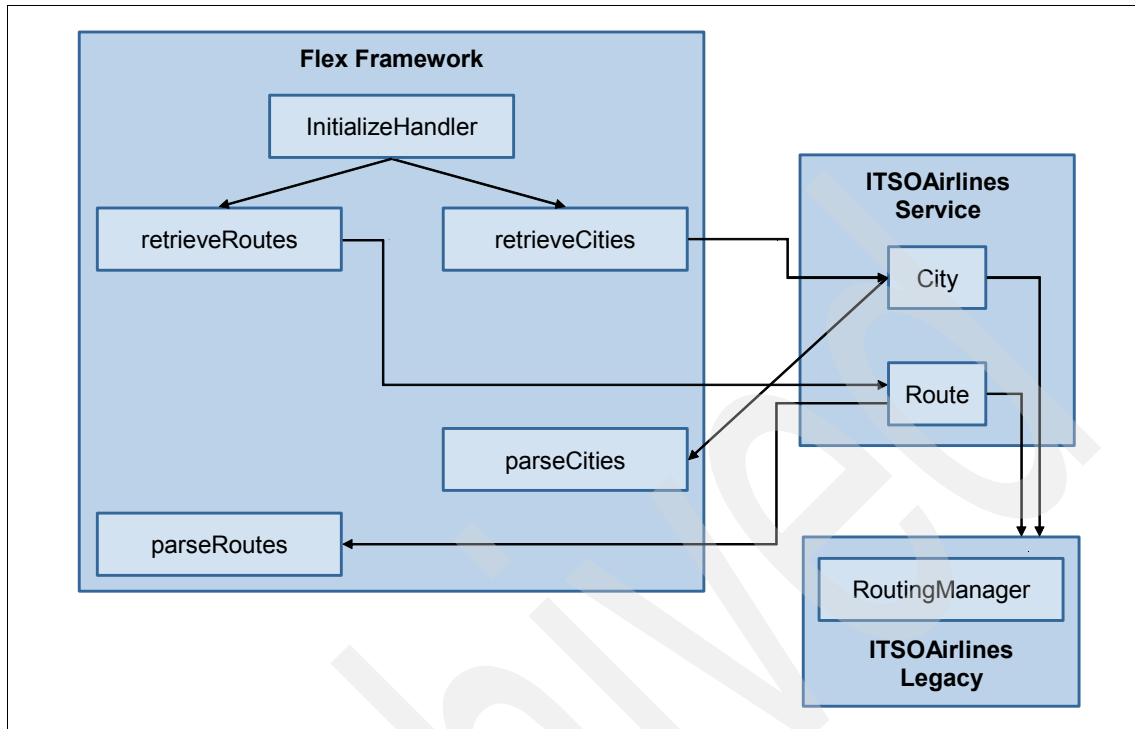


Figure 2-12 Integration scenario

2.4.4 Elixir diagram implementation

The following section covers implementing the elixir diagram.

Project creation

Perform the following steps to create the ITSOAirlines diagram:

1. In Adobe Flash Builder click **File** → **New** → **Flex Project**.
2. Enter ITSOAirlines_Diagram as the project name and click **Next**. For the next window, accept default settings and click **Next**.
3. In the window shown in Figure 2-13 on page 60, select **Merged into code** for the Framework linkage, click **Add SWC** and go to the Elixir Enterprise installation directory. Add the following files:
 - \$ELIXIR_ENTERPRISE_3.0_DIRECTORY/frameworks/libs/ilog-elixir-enterprise.swc
 - \$ELIXIR_ENTERPRISE_3.0_DIRECTORY/frameworks/locale/en_US/ilog-elixir-enterprise_rb.swc

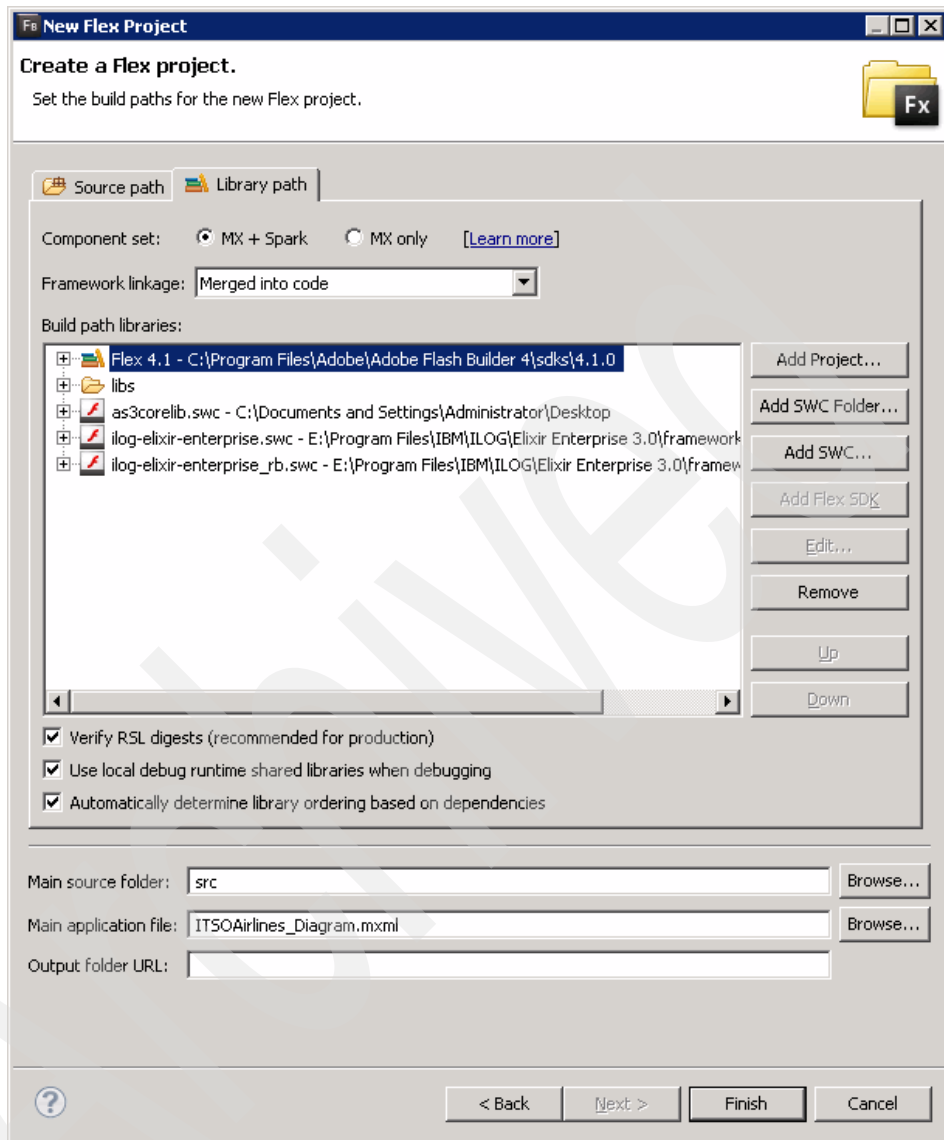


Figure 2-13 Build path options for the new Flex project

4. In order to parse the JSON response, you will need a JSON parser. The Flex platform does not provide such a parser by default. There are several third party libraries for handling the JSON format on the Flex platform. In our example, we use an open source library (under the BSD license) named `as3corelib` developed by Mike Chambers. You can find it at the following URL:

<http://github.com/mikechambers/as3corelib>

Select the library and click **Add SWC**. Locate the file `as3corelib.swc`. Click **Finish** to complete the New Flex Project wizard.

5. Next we are going to organize files in the project a little bit better. In the Package Explorer view right-click the project **ITSOAirlines_Diagram** and select **New** → **Package**, create a package named `itsoairlines.routing.ui` and click **Finish**.
6. Right-click `ITSOAirlines_Diagram.mxml` and select **Move**. Select the package you just created as the destination and click **OK**.

Importing resource files

Next, you are going to import resource files that you will need to create the diagram view.

1. Follow the instructions in step 5 to create a new package with the name `itsoairlines.routing.ui.skins`.
2. Right-click the package `itsoairlines.routing.ui.skins` and click **Import**.
3. Select **File System** and click **Next**.
4. In the window shown in Figure 2-14 on page 62, click **Browse** and navigate to where you extracted the additional materials for this chapter. Locate the directory `Chapter 2/elixir/skins` and click **OK**.

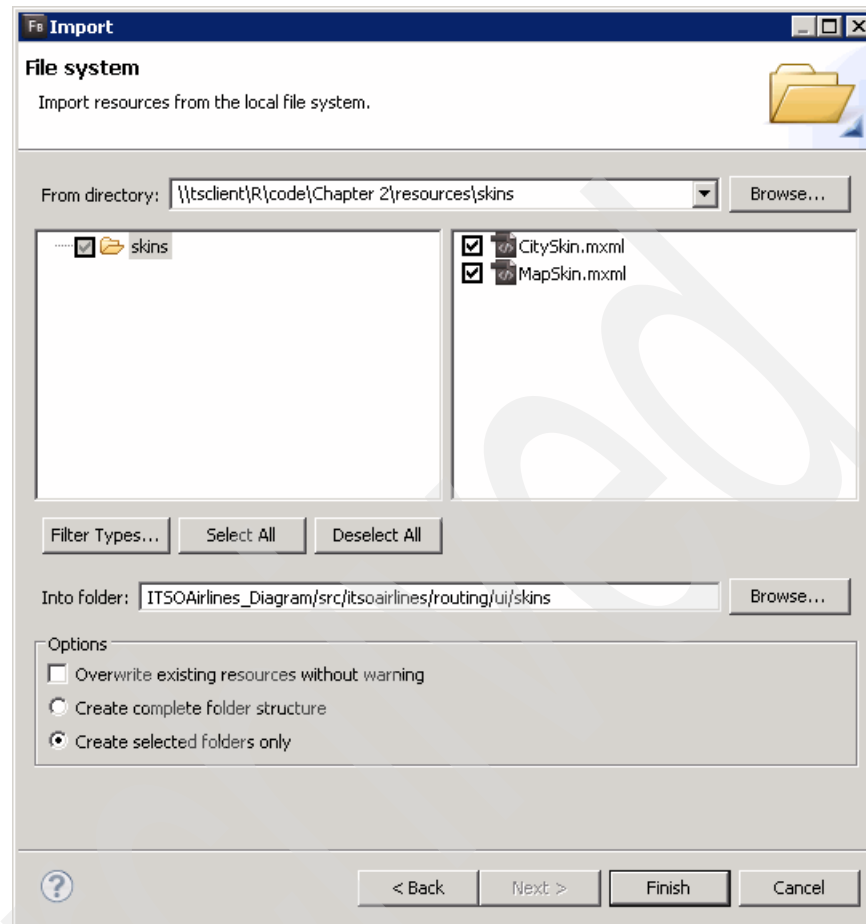


Figure 2-14 Import skins from the resources folder

5. From the right pane select the files MapSkin.mxml and CitySkin.mxml and click **Finish**.
6. Right-click the project **ITSOAirlines_Diagram** and click **Import**.
7. Select File System and click **Next**.
8. Navigate to the attached material folder and locate the folder Chapter 2/elixir/. On the left pane expand the elixir folder and select the folders locale and images (do not select the skins folder) and click **Finish** as shown in Figure 2-15 on page 63.

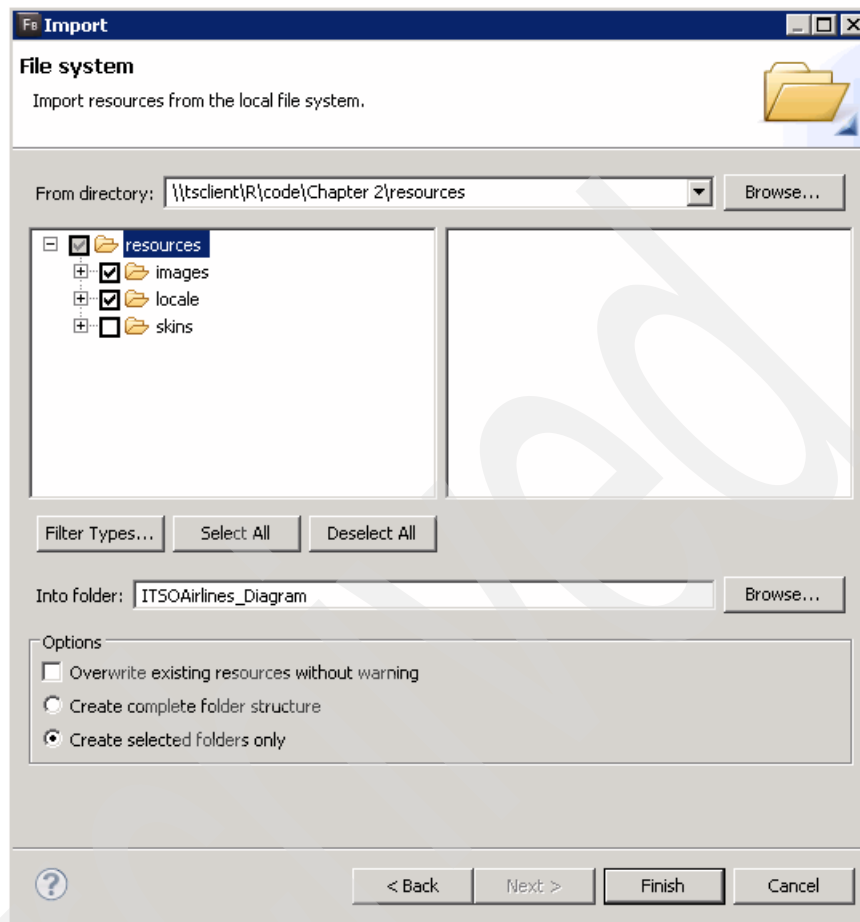


Figure 2-15 Import locale and image files

Writing the diagram code

The next steps configure the application and writes the necessary code for the diagram

1. Open the file ITSOAirlines_Diagram.mxml in the MXML editor.
2. Include the diagram XML namespace in your application by adding the line in Example 2-8 to `<s:Application>` declaration element.

Example 2-8 Include the diagram namespace

```
<s:Application
    ....
    xmlns:ibm="http://www.ibm.com/xmlns/prod/ilog/elixir/diagram/2010"
```

```
....  
>
```

3. Create and configure the diagram. Use the code in Example 2-9.

Example 2-9 Create the diagram

```
<ibm:Diagram  
    skinClass="itsoairlines.routing.ui.skins.MapSkin"  
    id='diagram'  
    width="100%"  
    height="100%"  
    allowMovingNodes="false"  
    contentBackgroundColor="#212629"  
    selectionMode="single"  
    automaticGraphLayout="false"  
    labelField="name"  
    centerNodesOnLocation="true"  
    xLocationFunction="longitudeToCoordinate"  
    yLocationFunction="latitudeToCoordinate"  
    startNodeFunction="getStartNode"  
    endNodeFunction="getEndNode"  
    startNodeField="startCityId"  
    endNodeField="endCityId"  
    >  
</ibm:Diagram>
```

The diagram you created will use the skin class you imported in step 2 on page 61. Because the nodes will be cities on a map, our diagram will not allow moving nodes.

4. You will see the errors in your project. Implement the missing functions that the diagram needs by adding a flex script tag and using the code in Example 2-10 on page 65 to implement the missing functions.

```
<fx:Script>
<![CDATA[
import mx.collections.ArrayCollection;
private var mapImageHeight:Number=546, mapImageWidth:Number=947;

/**
 * Converts longitude to coordinate in the diagram.
 * The Map background is defined in Lat/Long
 * Latitudes on the map are from 18N to 55.5 N
 * Longitudes are from 60W to 125W.
 */
public function longitudeToCoordinate(item:Object):Number
{
return mapImageWidth - ((-item['longitude'] - 60) * mapImageWidth /
(125 - 60));
}
/**
 * Converts latitude to coordinate in the diagram.
 * The Map background is defined in Lat/Long
 * Latitudes on the map are from 18N to 55.5 N
 * Longitudes are from 60W to 125W.
 */
public function latitudeToCoordinate(item:Object):Number
{
return mapImageHeight - (item['latitude'] - 18) * mapImageHeight /
(55.5 - 18);
}
/**
 * Gets the starting node object for a given link in a given diagram
 */
public function getStartNode(diagram:Diagram,linkItem:Object):Object
{
return getNode(diagram.nodeDataProvider, linkItem['startCityId']);
}
/**
 * Gets the end node object for a given link in a given diagram
 */
public function getEndNode(diagram:Diagram,linkItem:Object):Object
{
return getNode(diagram.nodeDataProvider, linkItem['endCityId']);
}
/**
 * A helper method to get a node from a list by its ID
 */
public function getNode(list:Object, nodeId:Number):Object
{
var nodes:ArrayCollection = list as ArrayCollection;
return nodes[nodeId-1];
}
]]>
</fx:Script>
```

5. Now write the code to get the cities and routes from the RESTful service and create nodes and routes from them. You will need this code to be executed when the diagram is initializing. Add an initializer function to the `<s:Application>` declaration element as shown in Example 2-11.

Example 2-11 Add an initializer function

```
<s:Application
    ....
    initialize="initializeHandler(event)"
    ....
>
```

6. Write the functions as shown in Example 2-12 that will get cities and routes from ITSOAirlines Service:
- `initializeHnaldler`: This function is invoked on diagram object initialization to start getting data for the diagram
 - `retrieveCities` and `retrieveRoutes`: Starts the asynchronous call to the RESTful service to retrieve a list of cities/routes.
 - `parseCities` and `parseRoutes`: A callback function for the asynchronous call performed in `retrieveCities/parseRoutes`, it parses the response to get a list of cities/routes and sets the appropriate property for the diagram.
 - `onError`: The callback function for the asynchronous call in case of failure: this just displays an alert with the received error.

Example 2-12 Code to do asynchronous calls to get the list of cities and routes

```
<fx:Script>
    <![CDATA[
        import mx.controls.Alert;
        import com.adobe.serialization.json.JSON;
        import mx.events.FlexEvent;

        private var SERVICE_URL:String =
"http://localhost:9080/ITSOAirlines_Service/jaxrs/";
        /**
         * Initializer function that will be called upon
application initialization to
         * retrieve cities and routes
         */
        protected function initializeHandler(event:FlexEvent):void
        {
            if (ExternalInterface.available)
                ExternalInterface.call("ElixirSample.initialize");
        }
    ]]>
```



```

        retrieveCities();
        retrieveRoutes();
    }
    /**
     * initiates an asynchronous call to retrieve a list of
cities from the service
     */

    public function retrieveCities():void
    {
        var request:URLRequest = new URLRequest();
        request.url = SERVICE_URL + "city";
        request.method = "GET";
        var loader:URLLoader = new URLLoader();
        loader.addEventListener(Event.COMPLETE,function
onResult(event:Event) : void {
            parseCities(event);
        });
        loader.addEventListener(IOErrorEvent.IO_ERROR,onError);
        loader.load(request);
    }

    /**
     * The call-back function for the retrieveCities
asynchronous call, it parses the result
     * and set the node data provider for the diagram.
     */
    public function parseCities(event:Event):void
    {
        var result:String = event.target.data
        var arr:Array = (JSON.decode(result) as Array);
        var dp:ArrayCollection = new ArrayCollection(arr);
        diagram.nodeDataProvider = dp;
    }

    /**
     * initiates an asynchronous call to retrieve a list of
routes from the service
     */
    public function retrieveRoutes():void
    {
        var request:URLRequest = new URLRequest();
        request.url = request.url = SERVICE_URL + "route";
        request.method = "GET";
        var loader:URLLoader = new URLLoader();

```

```

        loader.addEventListener(Event.COMPLETE,function
onResult(event:Event) : void {
            parseRoutes(event);
        });
        loader.addEventListener(IOErrorEvent.IO_ERROR,onError);
        loader.load(request);
    }

    /**
     * The call-back function for the retrieveRoutesb
asynchronous call, it parses the result
     * and set the link data provider for the diagram.
     */
    public function parseRoutes(event:Event):void
    {
        var result:String = event.target.data
        var arr:Array = (JSON.decode(result) as Array);
        var dp:ArrayCollection = new ArrayCollection(arr);
        diagram.linkDataProvider = dp;
    }

    /**
     * The error handler for the call back functions
     */
    private function onError(event:IOErrorEvent):void
    {
        Alert.show(event.text);
    }
    ]]>
</fx:Script>

```

-
7. Now you are ready to see your diagram in action. From the package explorer view, select project ITSOAirlines_Diagram and click **Run** → **Run ITSOAirlines_Diagram**. A browser window will open displaying your diagram. as seen in Figure 2-16 on page 69.

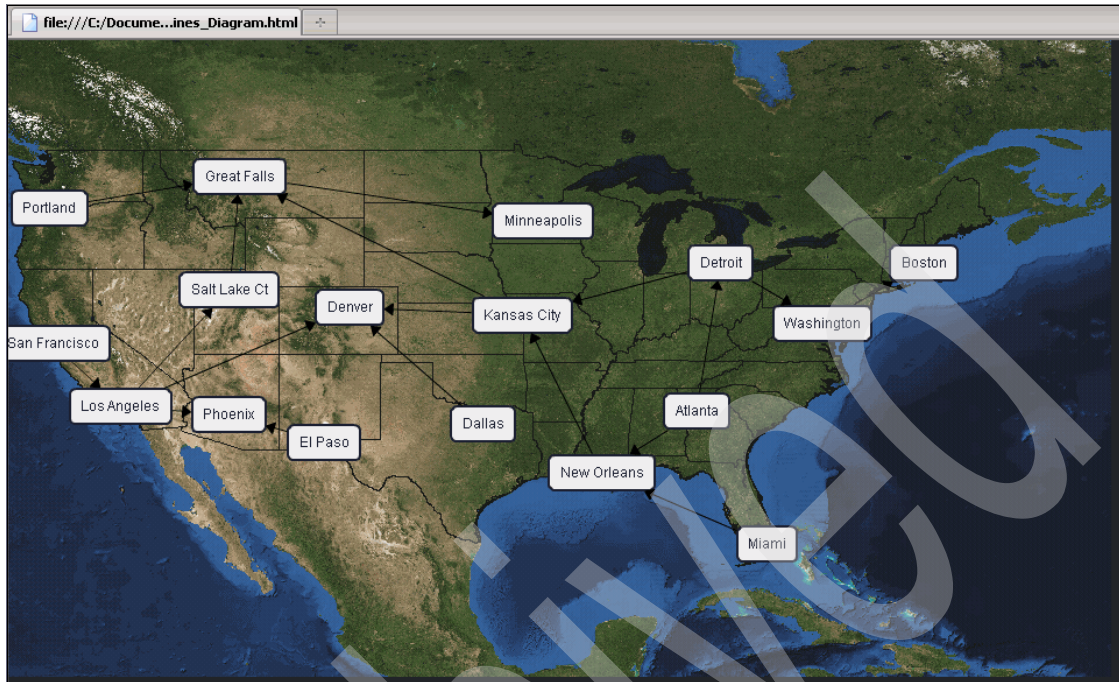


Figure 2-16 ITSO Airlines map

8. You might want change the style of your nodes and links. Add the code in Example 2-13 inside your application to make routes more visible and cities more user friendly.

Example 2-13 Add styling for links and nodes

```
<fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    @namespace ibm
    "http://www.ibm.com/xmlns/prod/ilog/elixir/diagram/2010";

    global{
        fontSize: 9pt;
    }
    ibm|Node{
        skinClass:
        ClassReference("itsoairlines.routing.ui.skins.CitySkin");
    }
    ibm|Link {
        stroke-color: #FFE0E0;
        stroke-width: 2
    }
```

```
}  
</fx:Style>
```

9. Run your application again. Now the map looks a little bit better as shown in Figure 2-17.

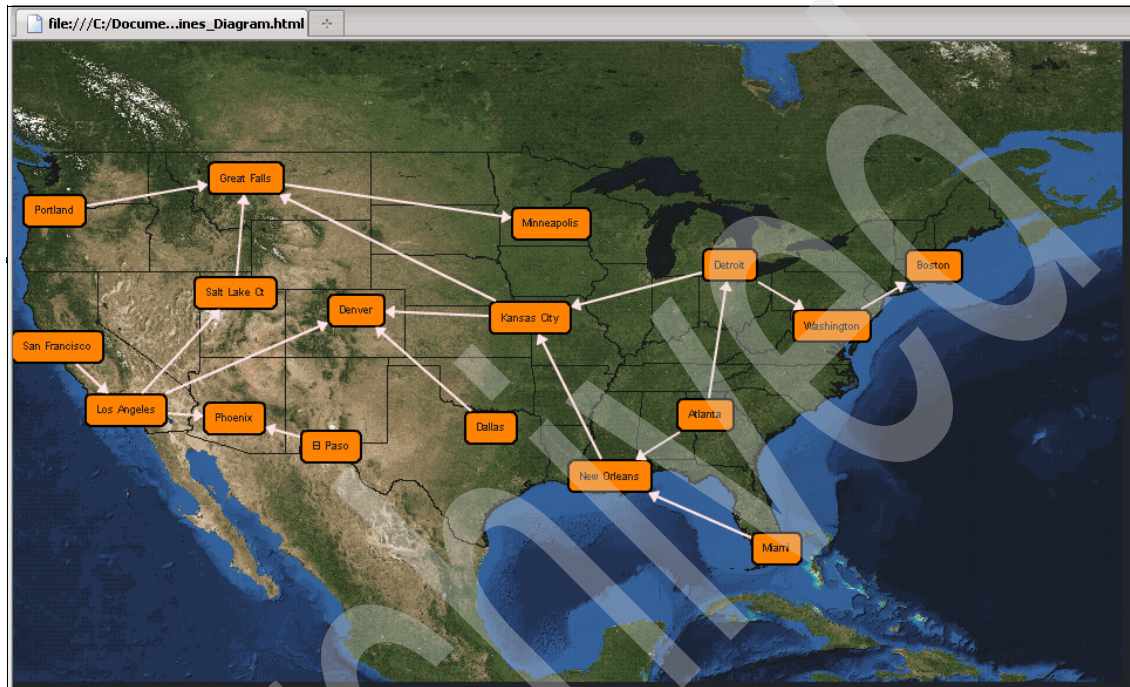


Figure 2-17 Styled ITSOAirlines map

2.5 Using the IBM ILOG JViews Enterprise product to visualize data from a RESTful service

In this section you are going to use the ILOG JViews Enterprise component to show a user friendly diagram for all of the cities and routes served by ITSOAirlines.

2.5.1 Overview

IBM ILOG JViews Enterprise provides web development capabilities that enable the design and deployment of internet-based applications, or “web applications”, based on the JViews web technology.

This technology provides the necessary client-server infrastructure to display your JViews product data on a web browser without any additional plug-ins. It also includes an extensive set of controls and modifiers that allow you to manipulate your graphic data directly from the browser.

Web applications exploit the convenience of using the web browser as the client, making application deployment transparent and updates immediately available to all users. Application management can be centralized on a few localized servers, therefore requiring fewer administration resources and maximizing the availability of the application.

JViews web technologies provide advanced capabilities for application designs with great flexibility because they can rely on JSF as the server-side component model, with the Ajax-based tag library as the client-side display technology. Or they can rely on server-side Java Servlets interacting with the Ajax-based JavaScript client-side display technology. This combination of different web technologies facilitates development work and provides easier integration with third-party components and tools.

In this section you are going to create a web application that uses ILOG JViews Diagrammer to show a user friendly diagram for all of the cities and routes served by ITSOAirlines.

2.5.2 Software requirements

In order to complete the next steps, you will need the following software to be installed.

- ▶ IBM ILOG JViews Enterprise 8.7.
- ▶ IBM Rational Application Developer 8.0
- ▶ IBM WebSphere Application Server 7.0 with Web 2.0 Feature Pack installed and the latest fix pack (We are using fix pack 11 in the chapter)

2.5.3 JViews diagram implementation

Development environment setup

In the next steps we will prepare our development environment for developing a JViews Diagrammer application on Rational Application Developer:

1. You need to have WebSphere Application Server defined in your development environment. Follow the steps in “Creating WebSphere Application Server” on page 44 to add WebSphere Application Server to your list of servers in Rational application developer if you have not done it already.

2. When developing JViews Diagrammer JSF applications, you must include the JViews Framework, JViews Diagrammer, and JSF supported jars, and distribute them with your web applications. Follow the next steps to create a user library for the jars required when developing JViews Diagrammer JSF applications:
 - a. Click **Window** → **Preferences** → **Java** → **Build Path** → **User Libraries**
 - b. Click **New** and create a new user library with the name JViews Diagrammer Framework
 - c. Select the library and click **Add Jars** and add the jars in Table 2-5 to your library. After you are done adding all of them, click **OK**.

Table 2-5 Build path jars needed for Diagrammer web applications

Jar name	Location
jviews-framework-all.jar	../jviews-framework87/lib
jsf-api-1.2_07-b03-FCS.jar	../jviews-framework87/lib/external
jsf-impl-1.2_07-b03-FCS.jar	../jviews-framework87/lib/external
jstl-1_1-mr2-api.jar	../jviews-framework87/lib/external
jviews-diagrammer-all.jar	../jviews-diagrammer87/lib
jviews-palette-shared-symbols-8.7.jar	../jviews-diagrammer87/palettes

Note: The JARs locations in this chapter are based on IBM ILOG JViews Enterprise 8.7. Specific locations of the JARs might vary according to the version you are using.

Project creation

Follow these steps to create the new project:

1. Click **File** → **New** → **Dynamic Web Project**.
2. In the window shown in Figure 2-18 on page 73, type ITS0AirLines_JDiagram in the Project name field. For the Target Runtime, select **WebSphere Application Server v7.0**, select the option of **add the project to an EAR project**, and click **Finish**.

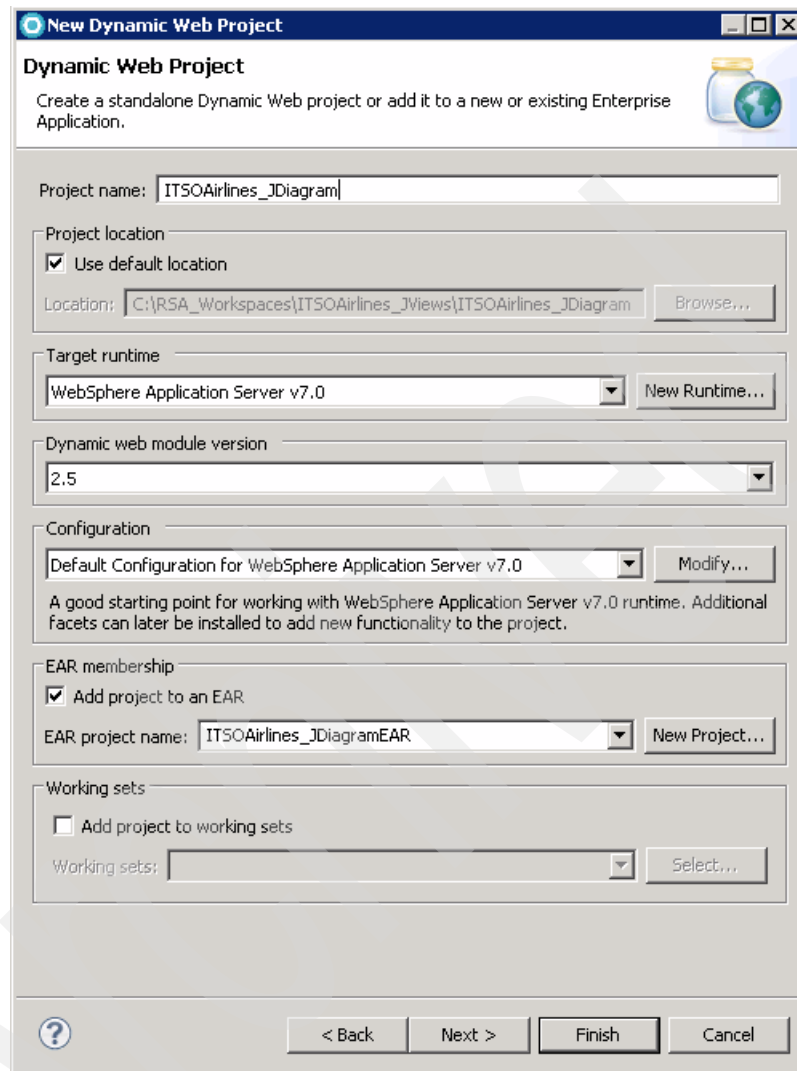


Figure 2-18 New dynamic web project

3. You need to add the necessary libraries to the build path of your project. Right-click the project **ITSOAirlines_JDiagram** and click **Properties** → **Java Build Path** → **Libraries**.
 - a. To add the JViews framework library we created in “Development environment setup” on page 71, click **Add Library** → **User Library** and click **Next**. Check the box next to **JViews Diagram Framework** and click **Finish**.

- b. JViews Framework libraries need to be exported with your application. To do so, click **Deployment Assembly** you will be prompted to save your changes, click **Apply**. Click **Add** select **Classpath Container** and click **Finish**. Select **User Library** and click **Next**. Select **JViews Diagram Framework** and click **Finish**.
- c. To add JSON4J library support click **Project Facets**. Expand the **Web 2.0** tree and check the box next to **Server-side technologies** and click **OK** to close the Properties window.

Import resource files

Next, you are going to import resource files that you will use in building this example.

1. Expand project ITS0Airlines_JDiagram and right-click the folder **WebContent** and click **New** → **Folder**.
2. Type resources as the folder name and click **Finish**.
3. Right-click the resources folder and click **Import**.
4. From the list select **File System** and click **Next**.
5. Click **Browse** and locate the additional material for this book. See Appendix A, “Additional material” on page 287 for the additional material supplied with this book. Select folder \Chapter 2\jviews\resources.
6. On the list on the left check the box next to the folder **resources** and click **Finish**. Check that three files are added to your resources folder.

Create the managed bean

Because JViews web applications are basically a JSF application, the first thing you need to do is to create and configure the managed bean class that will create the diagram.

Follow the next steps to create the managed bean for the ITS0Airlines example:

1. In the **Package Explorer** view, right-click the src folder and select **New** → **Class**. Use `itsoairlines.routing.ui` as the package name and `ITS0AirlinesBean` as the class name and click **Finish**.
2. Create a new variable of type `IlvDiagrammer` and create a getter method for it as shown in Example 2-14. Also implement the method `initializeDiagram()` and leave it empty. We will write its code later.

Example 2-14 Create the diagram variable

```
public class ITS0AirlinesBean {
    ...
    private IlvDiagrammer diagrammer;
```



```

public IlvDiagrammer getDiagrammer() {
    if (diagrammer == null) {
        diagrammer = new IlvDiagrammer();
        initializeDiagram();
        diagrammer.fitToContents();
    }
    return diagrammer;
}

private void initializeDiagram(){
}

...
}

```

3. Now you will implement helper methods that you will use in creating and populating the diagram:
 - The first method creates a new node and adds it to the diagram. Use the code snippet in Example 2-15.

Example 2-15 Add a new node to the diagram

```

private Object addNode(String nodeName,
    double xCord, double yCord, String label, String cityId) {
    Object node = diagrammer.createNode(nodeName);
    diagrammer.setObjectProperty(node, "x", xCord);
    diagrammer.setObjectProperty(node, "y", yCord);
    diagrammer.setObjectProperty(node, "label", label);
    diagrammer.addObject(node, null);

    return node;
}

```

- The next method to implement will be a generic method to read a URL and get the response as a string as shown in Example 2-16.

Example 2-16 Read a remote URL

```

public String queryService(String strURL) {
    URL url;
    InputStream stream = null;

    try {
        url = new URL(strURL);
        stream = url.openStream();
    } catch (Exception e) {
        e.printStackTrace();
    }
}

```

```

    }

    if (stream != null) {
        BufferedReader br = new BufferedReader(
            new InputStreamReader(stream));
        StringBuffer sb = new StringBuffer();
        String line = null;
        try {
            while ((line = br.readLine()) != null) {
                sb.append(line);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
        return sb.toString();
    } else {
        return "";
    }
}

```

- Based on the method `queryService` you wrote in the previous step, you can create two separate methods to query `ITSOService` and get JSON representations of cities and routes. See Example 2-17.

Example 2-17 Get JSON document for cities and routes

```

...
private final String serviceURL =
"http://localhost:9080/ITSOAirlines_Service/jaxrs/";
...
public String getCities() {
    return queryService(serviceURL + "city");
}

public String getRoutes() {
    return queryService(serviceURL + "route");
}

```

Note: Change the service URL and port to match your WebSphere Application Server configuration.

- The last two methods you are going to implement will be used to convert from map longitude and latitude to map image coordinates as in Example 2-18 on page 77.

Example 2-18 Convert from longitude and latitude to image coordinates

```
...
private final float mapImageHeight = 546;
private final float mapImageWidth = 947;
...

public double LongitudeToCoord(double longitude) {
    return mapImageWidth - ((-longitude - 60) * mapImageWidth /
(125 - 60));
}

    public double LatitudeToCoord(double latitude) {
        return mapImageHeight - (latitude - 18) * mapImageHeight /
(55.5 - 18);
    }
}
```

4. Now write the methods to get a list of Cities and Routes from ITSOService and parse it to create nodes and links on the diagram. See Example 2-19.

Example 2-19 Query ITSOService and create cities and routes on the diagram

```
...
private HashMap<Object, Object> nodesMap;
...

public void createCities(String jsonStr) {
    try {
        JSONArray arr = JSONArray.parse(jsonStr);
        for (Iterator iterator = arr.iterator();
iterator.hasNext();) {
            JSONObject city = (JSONObject) iterator.next();
            String name = (String) city.get("name");
            double longitude = (Double) city.get("longitude");
            double latitude = (Double) city.get("latitude");
            Object cityId = city.get("cityId");
            Object node = addNode(name, LongitudeToCoord(longitude),
LatitudeToCoord(latitude), name,
cityId.toString());
            nodesMap.put(cityId, node);
        }
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

```

    public void createRoutes(String jsonStr) {
        try {
            JSONArray arr = JSONArray.parse(jsonStr);
            for (Iterator iterator = arr.iterator();
iterator.hasNext();) {
                JSONObject city = (JSONObject) iterator.next();
                Object startCityId = city.get("startCityId");
                Object endCityId = city.get("endCityId");
                Object startCity = nodesMap.get(startCityId);
                Object endCity = nodesMap.get(endCityId);
                Object link = diagrammer.createLink("link", startCity,
endCity);
                diagrammer.addObject(link, null);
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}

```

5. To create the cities and routes during the diagram initialization, write the code in Example 2-20 for the method `initializeDiagram` that we created and left empty earlier.

Example 2-20 Initialize diagram

```

private void initializeDiagram(){
    nodesMap = new HashMap<Object, Object>();
    String citiesJSON = getCities();
    createCities(citiesJSON);
    String routesJSON = getRoutes();
    createRoutes(routesJSON);
}

```

Declaring the managed bean

You must now declare this class as a managed bean so that it can be accessed by JViews JSF components. To declare the class as a managed bean, follow the next steps:

1. In the Project Explorer view, right-click the **WebContent/WEB-INF** folder and click **New** → **Other**, select **File** and click **Next**.
2. Name the file `faces-config.xml` and click **Finish**.

3. Declare class `ITSOAirlinesBean` as a managed bean for this application by adding the configuration in Example 2-21 to `faces-config.xml`.

Example 2-21 Declare class as a managed bean

```
<faces-config xmlns="http://java.sun.com/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-facesconfig_1_2.xsd"
  version="1.2">
  <managed-bean>
    <description>ITSO Airlines bean</description>
    <managed-bean-name>itsoBean</managed-bean-name>
    <managed-bean-class>itsoairlines.routing.ui.ITSOAirlinesBean</manage
d-bean-class>
    <managed-bean-scope>session</managed-bean-scope>
  </managed-bean>

</faces-config>
```

Configure the JViews servlet

Declare and configure the JViews servlet used by your web application by opening the file `web.xml` located in the `WebContent/WEB_INF` folder and add the part shown in Example 2-22.

Example 2-22 Configure the application web.xml

```
<servlet>
  <servlet-name>Faces Servlet</servlet-name>
  <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
  <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
  <servlet-name>Faces Servlet</servlet-name>
  <url-pattern>/faces/*</url-pattern>
</servlet-mapping>

  <context-param>
    <param-name>ilog.views.faces.CONTROLLER_PATH</param-name>
    <param-value>/_contr</param-value>
  </context-param>
  <servlet>
    <servlet-name>Controller</servlet-name>
    <servlet-class>ilog.views.faces.IlvFacesController</servlet-class>
    <load-on-startup>1</load-on-startup>
```

```

</servlet>

<servlet-mapping>
  <servlet-name>Controller</servlet-name>
  <url-pattern>/_contr/*</url-pattern>
</servlet-mapping>

<listener>
<listener-class>ilog.views.util.servlet.DeploymentLicenseRequired_for_IBM_ILOG_JViews_Diagrammer_Deployment</listener-class>
</listener>

```

Create the main JSP page

To show your graphic view in web browsers, you must create a Java Server Pages (JSP) page:

1. In the **Package Explorer** view, right-click the **WebContent** folder and then select **New** → **Others** → **Web** → **JSP**.
2. Click **Next** and type `index.jsp` as the file name.
3. Click **Next** and select **New JavaServer Faces (JSF) Page (html)** as the template. Click **Finish**.
4. Open the file `index.jsp`. Add support for the diagrammer tag library and then create the diagrammer View inside the body element as in Example 2-23.

Example 2-23 Create the main JSP page

```

...
<%@ taglib
  uri="http://www.ilog.com/jviews/tlds/jviews-diagrammer-faces.tld"
  prefix="jvdf"%>
...
<body>

  <jvdf:diagrammerView id="diagrammer"
    style="width:947px;height:546px"
    styleSheets="/resources/itsoairlines.css"
    diagrammer="#{itsoBean.diagrammer}" editable="false"
    backgroundColor="#000000" resizable="true"
    waitingImage="/resources/loading.gif">
  </jvdf:diagrammerView>

</body>

```

Note: The diagrammer name supplied in the attribute diagrammer must match the managed bean name that was declared in file `faces-config.xml`. See “Declaring the managed bean” on page 78.

Deploying and starting the application

In this section, we discuss the steps to deploy the web application that utilizes your diagram. When we created the project for the diagram, we created a Dynamic Web Project and an EAR Application Project. We need to deploy that EAR project into WebSphere Application Server.

Perform the following steps to deploy the EAR project:

1. In the Servers view, right-click your **WebSphere Application Server 7.0** and select **Add and Remove Projects**.
2. In the Add and Remove window shown in Figure 2-19, select the **ITSOAirlines_JDiagram** from the available projects list and click **Add**.

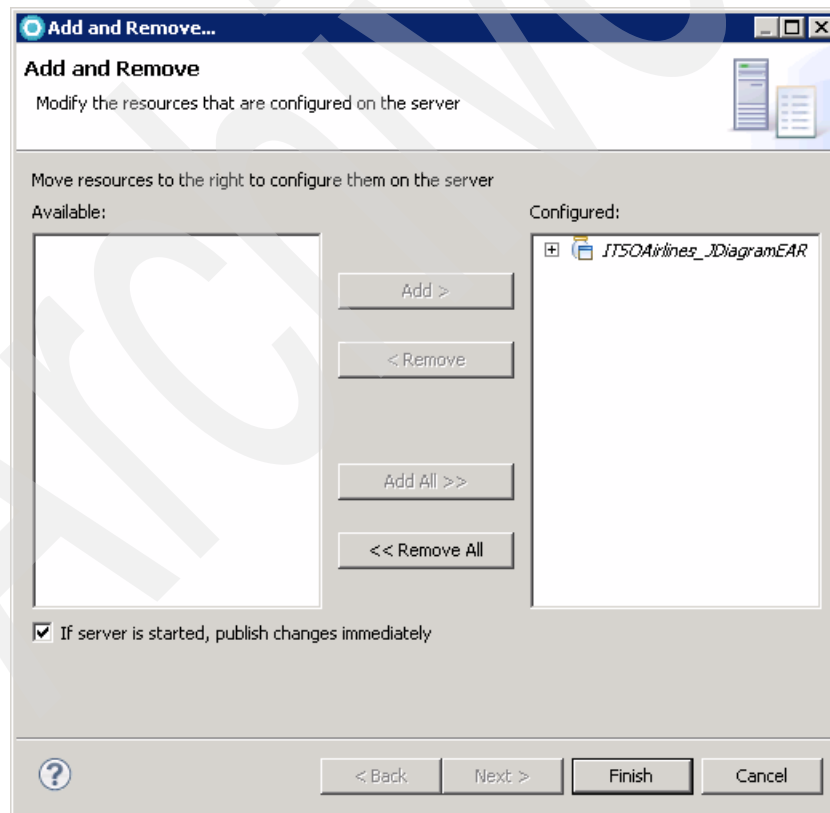


Figure 2-19 Deploy the EAR file to WebSphere Application Server 7.0

3. Click **Finish**.
4. After the Publish step completes, your application will be deployed and started.

Testing the application

To see your diagram in action, point your browser to the following URL:

http://localhost:8080/ITSOAirlines_JDiagram/faces/index.jsp

You should see the diagram as in Figure 2-20.

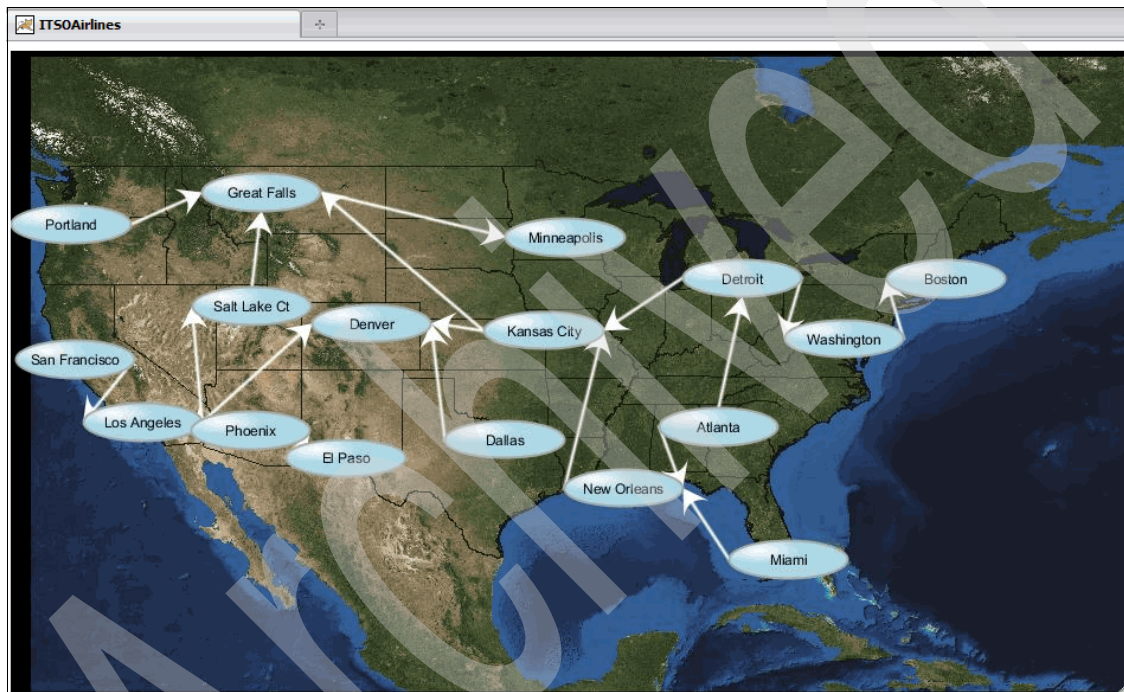


Figure 2-20 Final JView diagram

2.6 Conclusion

REST is a modern web architecture that is getting a lot of interest from Internet architects because of its unique ability to handle the web's scalability and growth requirements in a simple and understandable way.

IBM ILog visualization is an advanced set graphical displays and visualization software for user interface developers to create highly graphical and interactive displays quickly and efficiently.

Combining the two together allows the developer to quickly build rich applications that provide the user with a highly graphical user interface that simplifies the increasing complexity of modern business backend services.

Archived

Using IBM ILOG Visualization and IBM Cognos for web applications

This chapter describes how to create IBM ILOG JViews Enterprise and IBM ILOG Elixir Enterprise visualizations from IBM Cognos content using the IBM Cognos Mashup Service (CMS).

It follows the process of creating ILOG JViews and ILOG Elixir Gantt charts using an IBM Cognos report as a source. It also covers the process of publishing the visualizations in the IBM Cognos Business Insight dashboard.

Disclaimer: All names and associated information appearing in this work are fictitious. Any resemblance to real persons, living or dead, is purely coincidental.

3.1 Introduction to IBM Cognos Mashup Service

The IBM Cognos Mashup Service gives you a simplified programmatic access to IBM Cognos content. This service displays the application content built with IBM Cognos products as web services. Both SOAP and REST formats are supported.

This allows you to integrate IBM Cognos content into new client environments like rich Internet applications, mashups, workflow processes, desktop widgets, and alternate visualizations like IBM ILOG JViews Enterprise and IBM ILOG Elixir Enterprise.

The Mashup Service transforms all IBM Cognos content into a single format called Layout Data (LDX) format. This format allows you to customize the presentation of IBM Cognos content using a simple API. The LDX format captures the logical structure of the content and formatting information. For example, list grouping, crosstab dimensions, data values and styling information are represented in an LDX instance.

The Mashup Service can transform LDX instances into in a variety of formats, including HTML, HTMLFragment, and JSON, to facilitate the integration of IBM Cognos content into your applications.

The Mashup Service allows you to access existing IBM Cognos content, but does not provide any authoring functionality. If you need to author new content, refer to the IBM Cognos Software Development Kit (SDK).

Note: The IBM Cognos Mashup Service is only available in IBM Cognos BI versions 8.4.1 or later.

3.1.1 The CMS DataSet format

The IBM Cognos version 10 has introduced a new CMS output data format called DataSet. A DataSet format is a simplified XML format that contains only the report data, without any formatting information. DataSet format output consists of a dataSet root element containing one or more dataTable elements. Each dataTable element corresponds to a report part, or to a page of report part output if the report is requested with page breaks.

Requests for DataSet formatted output will usually be for a single report part without page breaks. The sample shown in Example 3-1 illustrates the main parts of a DataSet document for a grouped list report.

Example 3-1 CMS DataSet sample code

```
<?xml version="1.0" ?>
<dataSet
xmlns="http://www.ibm.com/xmlns/prod/cognos/dataSet/201006">
  <dataTable>
    <id>List1</id>
    <row>
      <Name>Angelo Farrell</Name>
```

```

<Location>New York</Location>
<Genre>male</Genre>
<StartDate>2010-02-15T00:00:00.000</StartDate>
<EndDate>2010-02-21T00:00:00.000</EndDate>
<Reason>Travel</Reason>
</row>
<row>
<Name>Angelo Farrell</Name>
<Location>New York</Location>
<Genre>male</Genre>
<StartDate>2010-03-27T00:00:00.000</StartDate>
<EndDate>2010-04-04T00:00:00.000</EndDate>
<Reason>Travel</Reason>
</row>
<row>
<Name>Georges Flecher</Name>
<Location>Paris</Location>
<Genre>male</Genre>
<StartDate>2011-07-01T00:00:00.000</StartDate>
<EndDate>2011-07-04T00:00:00.000</EndDate>
<Reason>Travel</Reason>
</row>
</dataTable>
</dataSet>

```

- ▶ The id child element of the dataTable contains the name of the report part from Report Studio.
- ▶ Each row in the table is represented by a row element.
- ▶ Element names inside the row element are based on the column title names in the report, and the content of these elements is the cell value from the table.

Note: The DataSet output format is only available for IBM Cognos version 10, which will be used for the integration scenarios in this chapter.

3.2 Creating the IBM Cognos sample report

This section will provide the steps to accomplish the following:

1. Create a data source using the IBM Cognos Framework Manager from an XML file and publish a Framework Manager package

2. Create a listing report using the IBM Cognos Report Studio

3.2.1 Software requirements

To perform the steps in this section, you will need to have the following applications installed:

- ▶ IBM Cognos BI 10
- ▶ IBM Cognos Framework Manager

For additional information about IBM Cognos BI V10.1, go to the following URL:

<http://publib.boulder.ibm.com/infocenter/cbi/v10r1m0/index.jsp>

3.2.2 The XML sample data source file

Before we start with the XML sample data import process, it is necessary to check if the file is compliant with the schema found in the `xmldata.xsd` (this file is located in *IBM Cognos install location\c10\bin*).

Figure 3-1 on page 89 shows a fragment of a IBM Cognos compliant XML schema.

```

<?xml version="1.0" encoding="UTF-8"?>
<dataset xmlns="http://developer.IBM Cognos.com/schemas/xmldata/1/"
xmlns:xs="http://www.w3.org/2001/XMLSchema-instance">
  <metadata>
    <item name="Name" type="xs:string"/>
    <item name="Location" type="xs:string"/>
    <item name="Genre" type="xs:string"/>
    <item name="StartDate" type="xs:date"/>
    <item name="EndDate" type="xs:date"/>
    <item name="Reason" type="xs:string"/>
  </metadata>
  <data>
    <row>
      <value>Angelo Farrell</value>
      <value>New York</value>
      <value>male</value>
      <value>2010-02-15</value>
      <value>2010-02-21</value>
      <value>Travel</value>
    </row>
    <row>
      <value>Charles Cournes</value>
      <value>New York</value>
      <value>male</value>
      <value>2010-03-5</value>
      <value>2010-03-13</value>
      <value>Vacation</value>
    </row>
  </data>
</dataset>

```

Figure 3-1 XML compliant code example

3.2.3 Creating the data source in IBM Cognos Framework Manager

The following steps will be performed in the IBM Cognos Framework Manager application. At this point, we assume that you have it properly configured to connect to the IBM Cognos BI server:

Note: To recreate the scenarios described in this chapter, you should download the sample files included in the additional material available with this book. See Appendix A, “Additional material” on page 287 for details on the additional sample scenario material for this chapter included with this book.

Before proceeding, you should download the `hr.xml` file as described in the appendix.

1. Click **File** → **New**.
2. Input a project name to the FM model and click **OK**. Note that you have an option to customize the path for where the project will be saved. Figure 3-2 shows the New Project window.

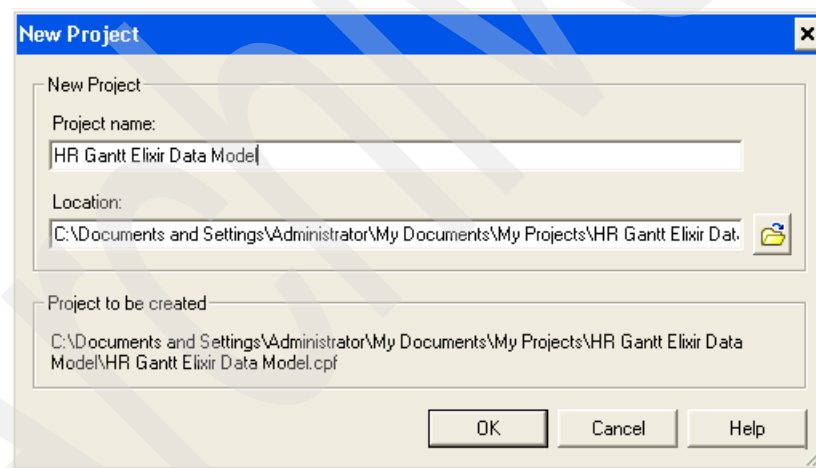


Figure 3-2 Main Project properties window

3. Select the desired language for the project by clicking **English** → **OK**.
4. In the Select Metadata Source window, click **Data Sources** and then **Next**. Figure 3-3 on page 91 shows the Select Data Source window.

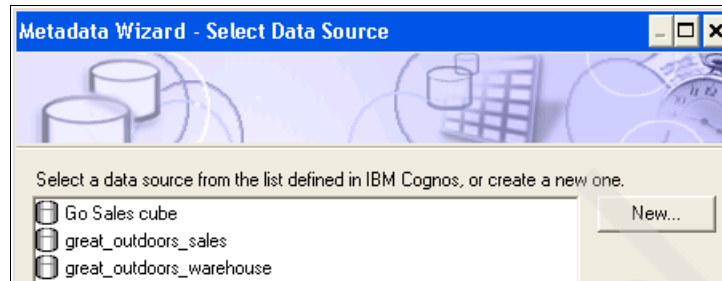


Figure 3-3 Select Data Source window

5. In the Select Data Source window click **New**. This opens the New Data Source wizard. Click **Next**.
6. Give the Data Source a **Name**, optionally a **Description**, and a **Screen tip**, then click **Next**. Figure 3-4 shows the Data Source properties.

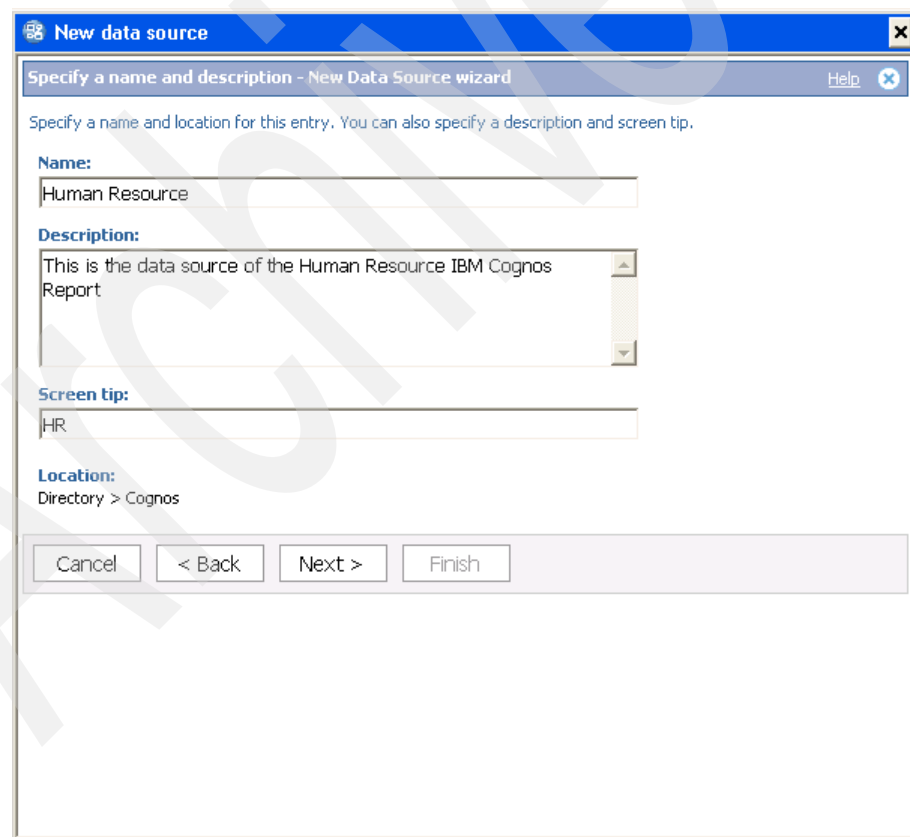


Figure 3-4 New Data Source wizard

7. Specify the connection type as XML in the drop-down field and click **Next**.
8. Enter the XML connection string.

It is important to test the connection to make sure the path is correct and the file is valid. For this example we are going to use the `hr.xml` file previously downloaded in the root folder of the `C:\` drive.

Figure 3-5 shows the Connection string window.

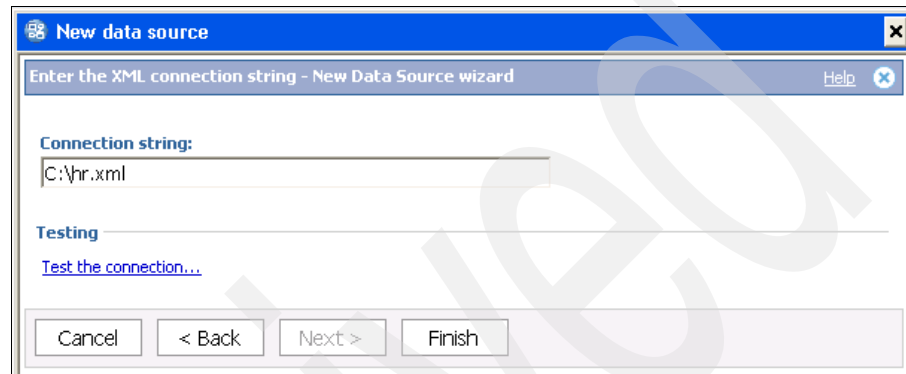


Figure 3-5 Connection string window

Note: It is possible to use an HTTP address for the **Connection string** field filling it with the full path to the file, such as:

`HTTP://server/ibmIBM Cognos/hr.xml`

9. Test the connection by clicking the **Testing** link. You will find a window with the connection and dispatcher information. Select **Test**
10. If the connection is successful, you will get a confirmation as shown in the Figure 3-6.

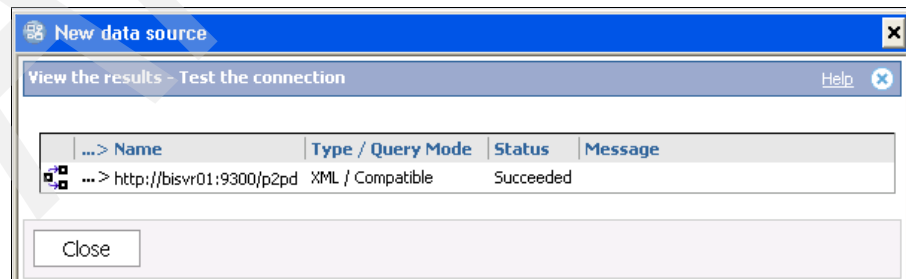


Figure 3-6 Connection successful

11. Now **Close** both testing screens and click **Finish**.

A confirmation message will be displayed as shown in Figure 3-7.

The New Data Source wizard successfully created a data source and a first connection in the IBM Cognos software server.

Figure 3-7 Connection process completed message

12. Click **Close**.

We have successfully created the Human Resources data source. You will now be able to find it as a new source in the select Data Source window.

For the next steps, we are going to import the data model and then publish it as a package. It will be used for the Report Studio listing report development section later.

1. Select the Human Resources data source and click **Next**.
2. Expand the source to a table level and select the Human Resources table. Click **Next** and then **Import**. A message as shown in Figure 3-8 will be displayed.

Completed the import process.

Created the following objects:
Type: Query Subject, Count: 1

Figure 3-8 Process completed message

3. Click **Finish**.

3.2.4 Publishing a Framework Manager package for the IBM Human Resources sample report

This section covers the process of creating a Framework Manager package and how to publish it. This is a required step for the listing report development which will be used for the ILOG Visualization integration scenarios. This is the IBM Cognos report that the Cognos Mashup Service will retrieve the data as a source for the Gantt charts.

You have now reached the Framework Manager workspace. Refer to Figure 3-9 on page 94.

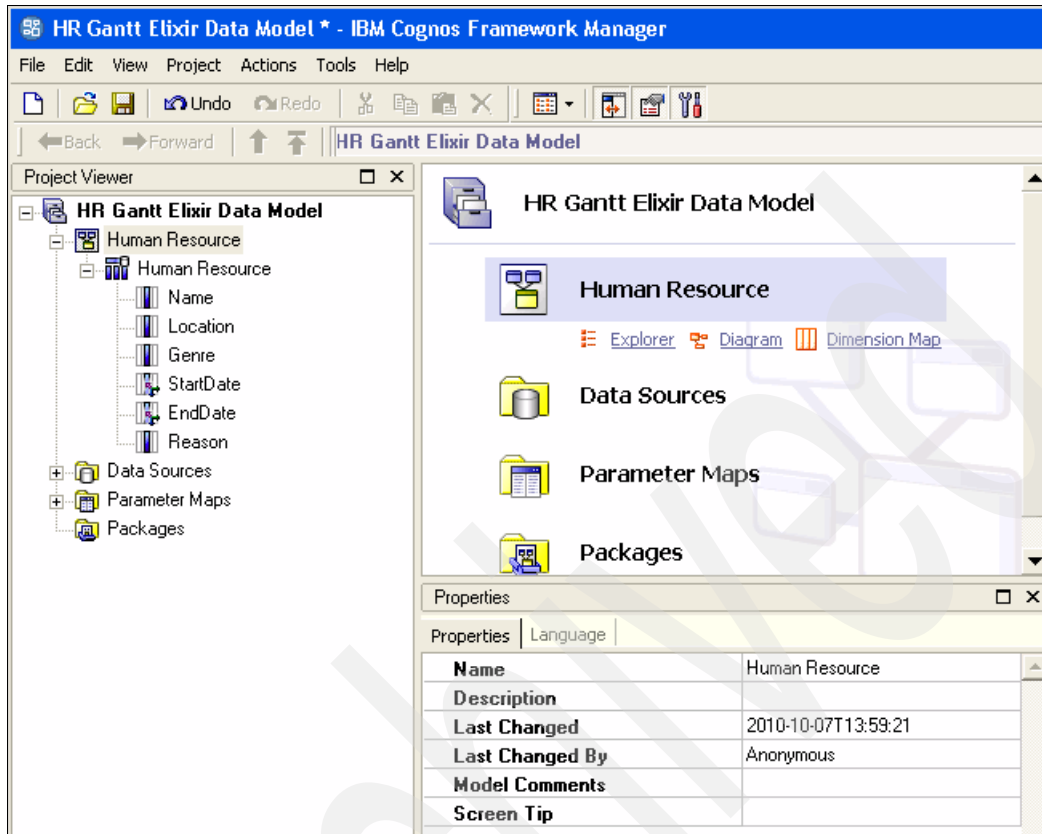


Figure 3-9 The IBM Framework Manager workspace

Note: Before we proceed with the package process it is important to test the data imported from the XML.

1. In the Project Viewer panel, expand to the field level of your data source and right-click **Name**, select **Test**, and then click **Test Sample**. Figure 3-10 on page 95 shows a sample of a successful query test.

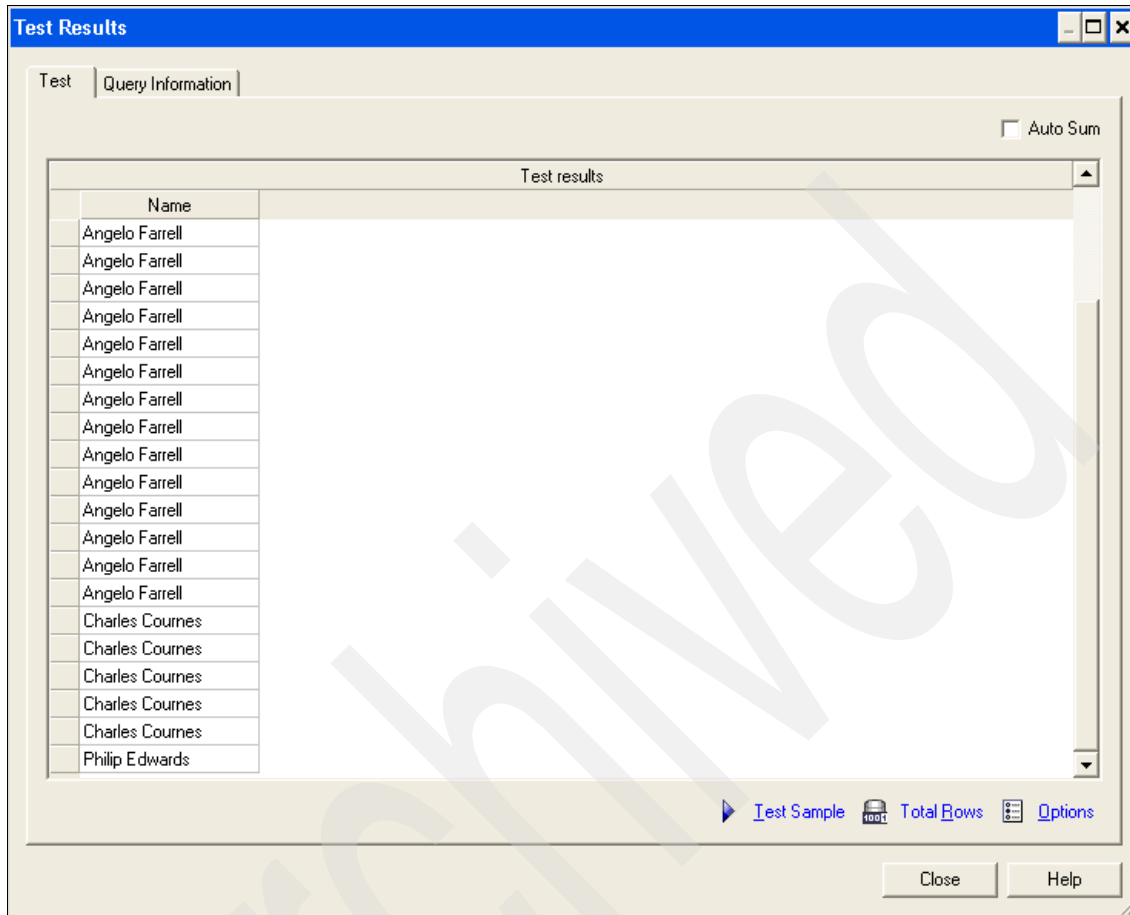


Figure 3-10 Sample data

2. Close the **Test Results** box.
3. In the Project Viewer Panel, right-click **Packages** and select **Create** → **Package**.
4. Give the package the name ILOG Visualization HR. Click **Next** → **Finish**.
At this point, the package has been successfully created. You will be presented with an option to publish the package using a wizard.
5. Click **Yes**.
6. Follow with the defaults up to the Publish window and then select **Publish**.

We are now ready to move on to the next section and create the listing report with the IBM Cognos Report Studio.

3.2.5 Creating the listing report using the IBM Cognos Report Studio

In this section we will cover the process of creating a listing report to be used as an example for the ILOG integration.

Note: We will be using the same Human Resources report as a source for the both ILOG JViews and Elixir Gantt chart visualizations.

In the IBM Cognos main window, go to **My Actions** and click the **Author advanced report** option.

1. In the Select a Package window, locate and select the package that you created, ILOG Elixir HR.
2. In the IBM Cognos Report Studio application, click **Create new**. Figure 3-11 shows the report options available.
3. Click **List** and then **OK**.

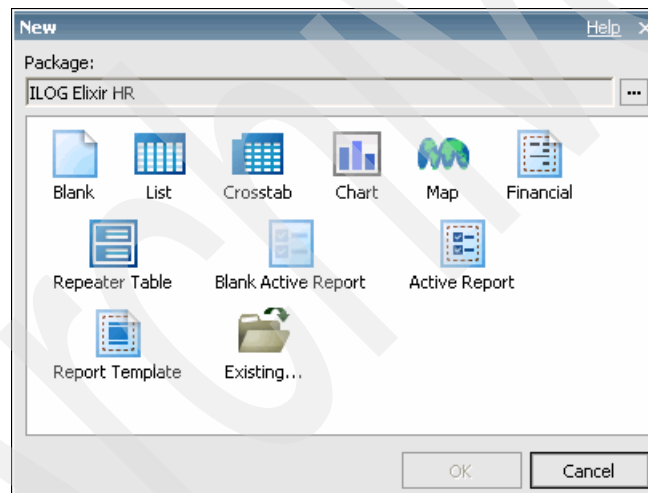


Figure 3-11 New report window

4. Move the mouse over the **Query Explorer** and click the **Query1** item. Figure 3-12 on page 97 shows how to edit a query in the **Query Explorer**.

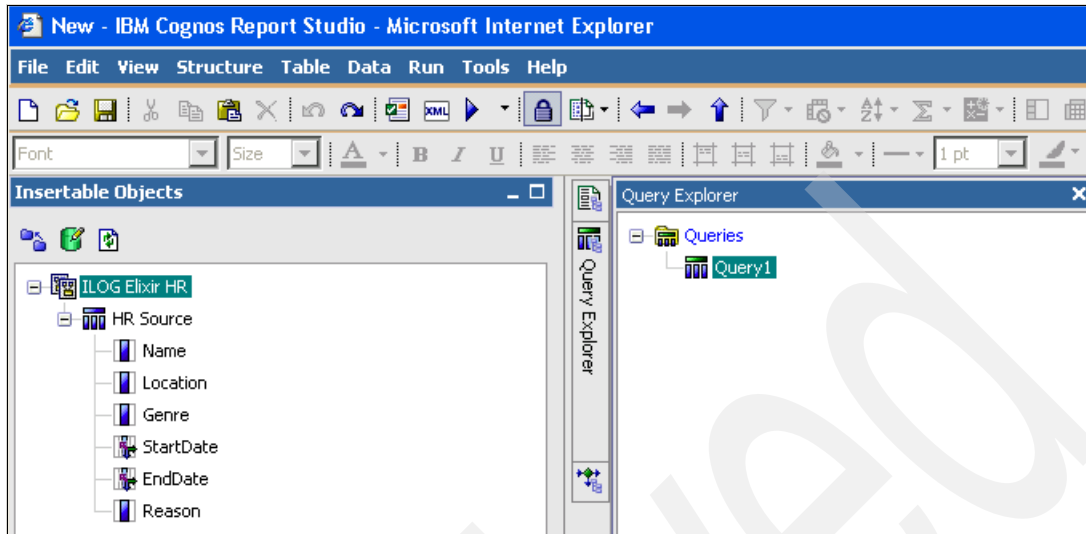


Figure 3-12 Query properties

5. Drag and drop all the columns available in the package to the **Query Explorer** section.

Figure 3-13 shows the Query Explorer with the proper query items.

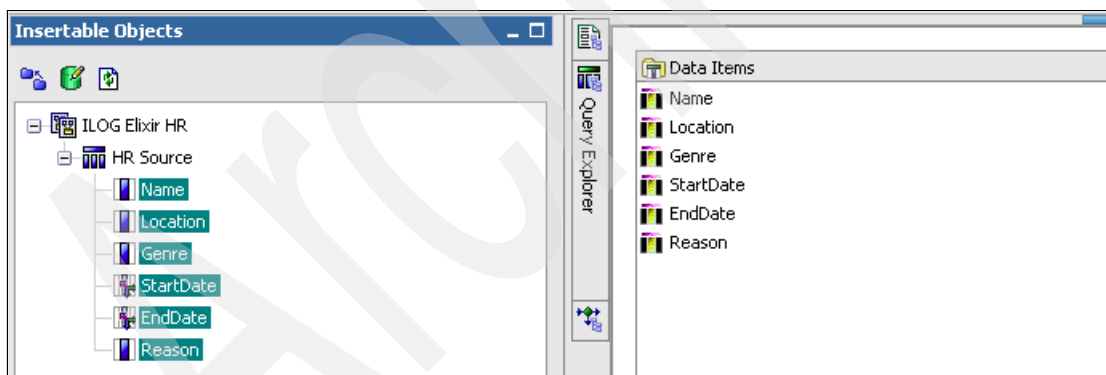


Figure 3-13 Query items added

6. Move the mouse over the **Page Explorer** panel and select the **Page 1** item.

Figure 3-14 on page 98 shows the report explorer panel.

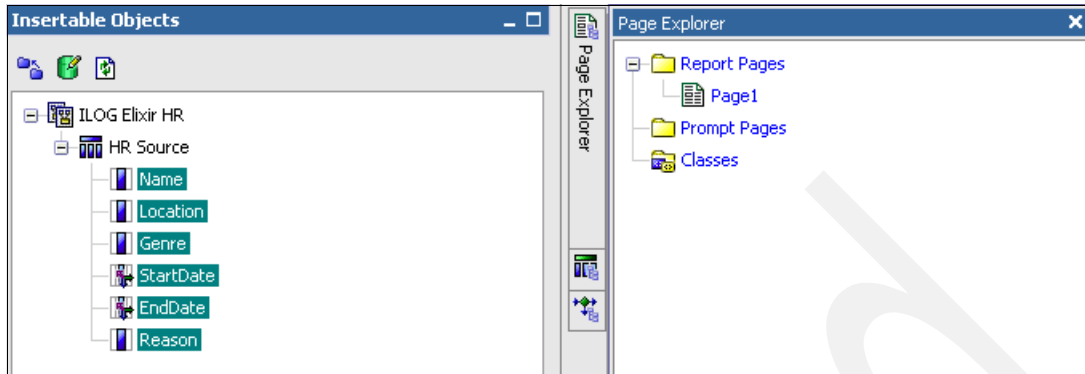


Figure 3-14 Report explorer panel

7. In the Insertable Objects window, switch from Data Sources to Data Items as shown in the Figure 3-15.

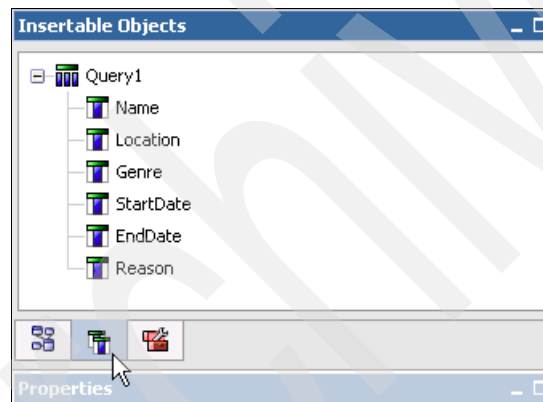
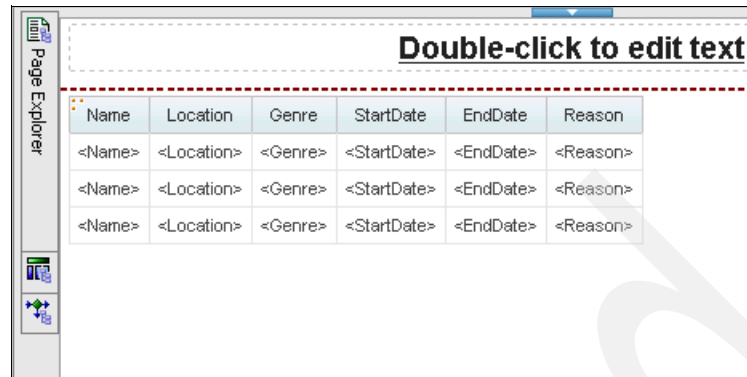


Figure 3-15 Insertable objects panel

8. Drag and drop all the items from the Insertable Objects section to the Report section as shown in the Figure 3-16 on page 99.



Name	Location	Genre	StartDate	EndDate	Reason
<Name>	<Location>	<Genre>	<StartDate>	<EndDate>	<Reason>
<Name>	<Location>	<Genre>	<StartDate>	<EndDate>	<Reason>
<Name>	<Location>	<Genre>	<StartDate>	<EndDate>	<Reason>

Figure 3-16 The IBM Cognos list report

9. Save the report. The name must be Absences. Run the report to make sure that it looks like the one shown in Figure 3-17 on page 100.

IBM Cognos Viewer					
Name	Location	Genre	StartDate	EndDate	Reason
Angelo Farrell	New York	male	Feb 15, 2010	Feb 21, 2010	Travel
Angelo Farrell	New York	male	Mar 27, 2010	Apr 4, 2010	Travel
Angelo Farrell	New York	male	Apr 27, 2010	May 8, 2010	Travel
Angelo Farrell	New York	male	Jun 18, 2010	Jun 25, 2010	Travel
Angelo Farrell	New York	male	Jul 6, 2010	Jul 9, 2010	Travel
Angelo Farrell	New York	male	Aug 7, 2010	Aug 22, 2010	Vacation
Angelo Farrell	New York	male	Sep 25, 2010	Oct 3, 2010	Travel
Angelo Farrell	New York	male	Oct 31, 2010	Nov 5, 2010	Mission
Angelo Farrell	New York	male	Dec 6, 2010	Dec 11, 2010	Vacation
Angelo Farrell	New York	male	Jan 23, 2011	Jan 28, 2011	Travel
Angelo Farrell	New York	male	Apr 3, 2011	Apr 9, 2011	Travel
Angelo Farrell	New York	male	May 5, 2011	May 12, 2011	Sickness
Angelo Farrell	New York	male	Jun 16, 2011	Jun 19, 2011	Travel
Angelo Farrell	New York	male	Jul 1, 2011	Jul 7, 2011	Sickness
Angelo Farrell	New York	male	Jul 21, 2011	Jul 25, 2011	Travel
Angelo Farrell	New York	male	Aug 11, 2011	Aug 22, 2011	Mission
Angelo Farrell	New York	male	Sep 24, 2011	Sep 29, 2011	Travel
Angelo Farrell	New York	male	Nov 4, 2011	Nov 17, 2011	Vacation
Angelo Farrell	New York	male	Dec 16, 2011	Dec 23, 2011	Travel
Charles Courmes	New York	male	Mar 5, 2010	Mar 13, 2010	Vacation

[Top](#)
[Page up](#)
[Page down](#)
[Bottom](#)

Figure 3-17 The IBM Cognos Absences list report

Important: We have completed the IBM Cognos report development at this point. It is important to save the report with the Absences name because it is hard-coded in the sample code that we will be using in our integration scenarios.

3.3 Integrating ILOG JViews Gantt visualization with IBM Cognos report

This section covers how to integrate IBM ILOG JViews and IBM Cognos. Planning and scheduling applications often require Gantt chart (timeline) displays to view, edit, and monitor the status of the resources, tasks, and activities in the underlying application.

We will cover the step-by-step process of creating a basic web project to display the IBM Cognos Absences report data into an ILOG JViews Gantt chart visualization format.

Gantt charts are specialized bar charts that help clearly represent how tasks and resources are allocated over time in planning, project management, and scheduling applications. For our integration scenario, we will use the Resource-oriented Gantt chart type.

3.3.1 Software requirements

In order to accomplish the tasks in this section, you will need to have the following applications installed:

- ▶ IBM Cognos 10
- ▶ IBM Rational Application Developer (RAD) 8.0
- ▶ IBM WebSphere Application Server (WAS) 7.0 w/ Fixpack 11
- ▶ IBM ILOG JViews Enterprise 8.7

3.3.2 Creating the IBM Human Resources JViews Gantt chart project

Using the Rational Application Developer application, we need to create a new Dynamic Web Project:

1. Go to **File** → **New** → **Dynamic Web Project**.
2. Type IBM Human Resources as the **Project name**. See Figure 3-18 on page 102.

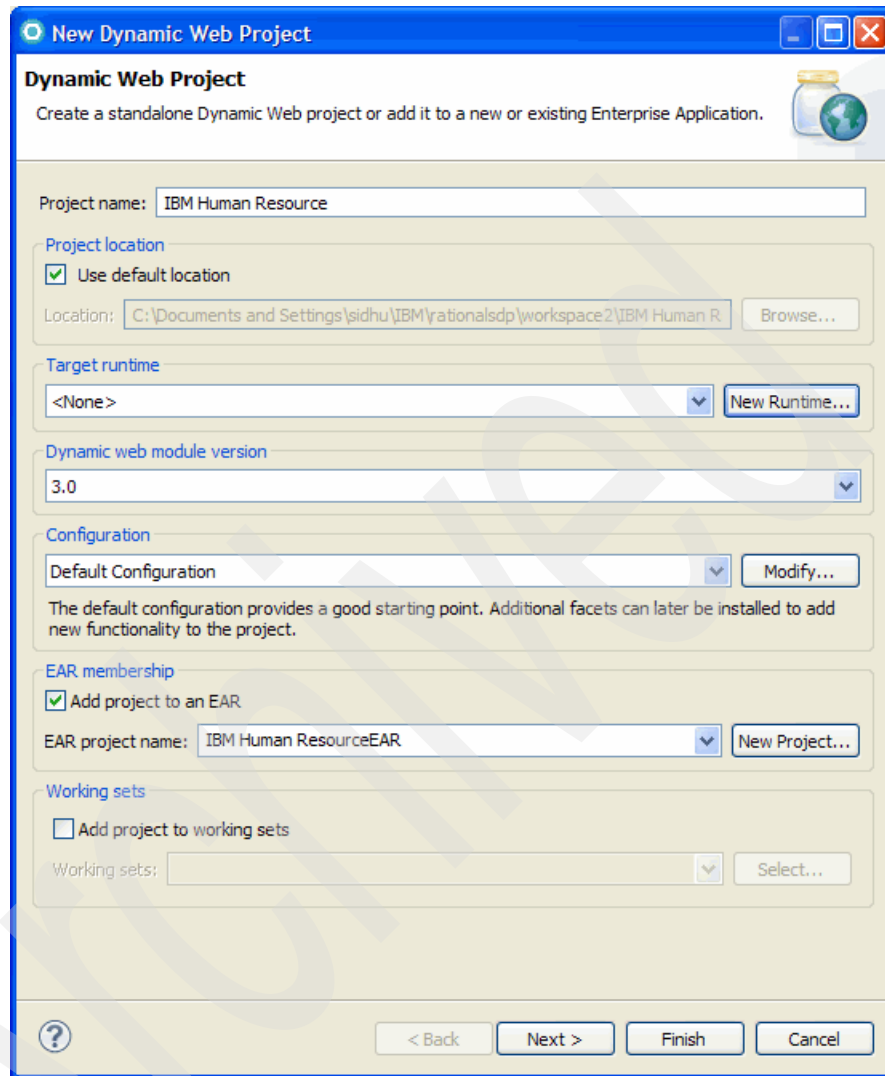


Figure 3-18 Project properties

3. Click **New Runtime**.
4. In the **IBM** folder, select **WebSphere Application Serve v7.0** and then click **Next**. See Figure 3-19 on page 103.

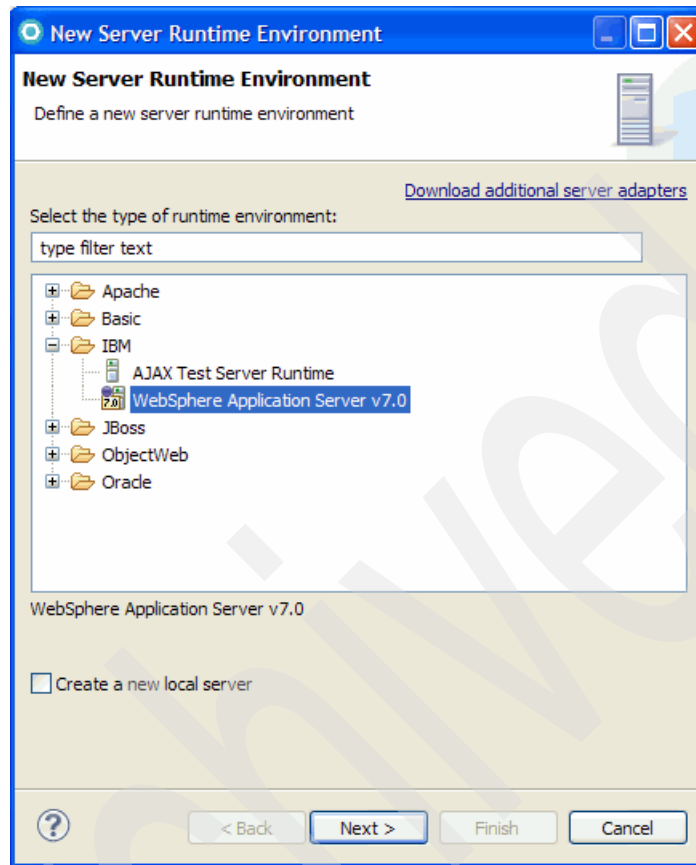


Figure 3-19 Runtime properties

5. For the Installation directory section, click **Browse**, locate your WebSphere Application Server installation, and click **Finish**. See Figure 3-20 on page 104.

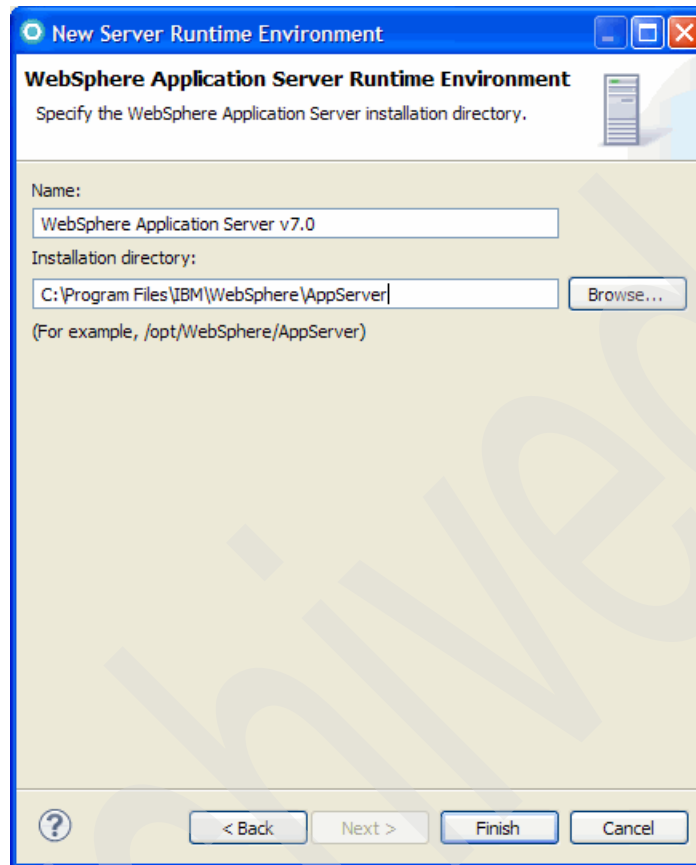


Figure 3-20 Runtime environment

6. Click **Finish** to finish the New Dynamic Web Project creation.

At this point, we have created our new base project. Before we start with the code, it is necessary to configure the IBM ILOG JViews libraries:

1. In the **Enterprise Explorer** panel, right-click the **IBM Human Resources** project and select **Properties**.
2. Go to the **Java Build Path** configuration and click the **Libraries** tab.
3. Click **Add External JARs** and add the following files:
 - `installation_path/IBM/ILOG/jviews-gantt87/lib/`
 - `jviews-gantt-all.jar`
 - `installation_path/IBM/ILOG/jviews-framework87/lib/`
 - `jviews-framework-all.jar`

- *installation_path*/IBM/ILOG/jviews-framework87/lib/external/
 - jstl-1_1-mr2-api.jar
 - jsf-impl-1.2_07-b03-FCS.jar
 - jsf-api-1.2_07-b03-FCS.jar
 - commons-logging-api-1.1.jar
 - commons-logging-1.1.jar
 - commons-lang-2.1.jar
 - commons-fileupload-1.2.1.jar
 - commons-digester-1.7.jar
 - commons-collections-2.1.jar
 - commons-beanutils-1.6.jar

3.3.3 Components for Java and web deployment

The project requires three main files to be created:

1. A client side web page called `testTable.jsp`.
2. A server side JSF bean called `TestTableBean.java`.
3. An XML reader Java Class called `CognosDocumentReader.java`.

Note: The source code for the RAD project that is created in this section is available on the web. Check Appendix A, “Additional material” on page 287 for details on how to download it.

The `testTable.jsp` file goes in the directory `WebContent/WEB-INF`. The other two files go in Java Resources/`src`/(default package) as shown in Figure 3-21.

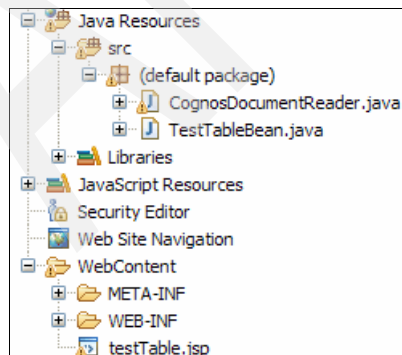


Figure 3-21 RAD Enterprise Explorer project files location

The following sub-sections show snippets of code from the respective files.

The testTable.jsp file

This file implements the ILOG JViews Gantt chart and parses it in the web browser as shown in Example 3-2.

Example 3-2 The .jsp client side web page

```
<jvgf:scheduleView id="gantt"
binding="#{testTableBean.ganttView}"
chart="#{testTableBean.ganttView.chart}"
style="width:1200px;height:800px" imageFormat="PNG"
resizable="true"
messageBoxId="messages"
waitingImage="data/images/ilog-anim.gif"
tableInteractorId="tableExpand"
sheetInteractorId="sheetExpand" sheetVScrollable="true">
</jvgf:scheduleView>
```

The TestTableBean.java file

This is the JSF Bean of our application where we connect to the IBM Cognos CMS and interface with CognosDocumentReader.java and testTable.jsp. See Example 3-3.

Example 3-3 The TestTableBean.java bean

```
public class TestTableBean {

    private IlvFacesHierarchyChartView ganttView = new
    IlvFacesDHTMLGanttChartView();

    public TestTableBean() {
        IlvScheduleChart gantt = new IlvScheduleChart();
        try {
            /*
             * Construct the IBM Cognos CMS URL and parse it to get the Gantt
             * model
             */
            URL url = new
            URL("http://localhost/ibmcognos/cgi-bin/cognos.cgi/rds/reportData/path/
            Public%20Folders/ILOG%20Elisir%20HR/Absences?async=OFF&fmt=DataSet");
            CognosDocumentReader reader=new CognosDocumentReader();
            IlvGanttModel model = reader.getModel(url, gantt);
            // gantt.expandAllRows();
```



```

        gantt.setGanttModel(model);
        ganttView.setChart(gantt);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }
}

public IlvFacesDHTMLGanttChartView getGanttView() {
    return (IlvFacesDHTMLGanttChartView) ganttView;
}

public void setGanttView(IlvFacesDHTMLGanttChartView ganttView) {
    this.ganttView = ganttView;
}
}

```

To get data from a report, the default URI to the IBM Cognos Mashup Service REST interface is:

/ibmcognos/cgi-bin/cognos.cgi/rds/reportData/path/report path

where *ibmcognos* can change depending on your server configuration.

The CognosDocumentReader.java file

This is the Java file where we implement the classes that process the XML resource and task information for the Gantt chart. See Example 3-4.

Example 3-4 The CognosDocumentReader.java class

```

public IlvGanttModel getModel(URL url, IlvScheduleChart gantt)
throws Exception {

    /*
    * Read the URL and parse it to DocumentBuilder
    */
    InputSource source = new InputSource(url.toString());

    DocumentBuilderFactory domFactory =
    DocumentBuilderFactory.newInstance();
    domFactory.setNamespaceAware(true); // never forget this!
    DocumentBuilder builder = domFactory.newDocumentBuilder();
    Document doc = builder.parse(source);
}

```

```

XPathFactory factory = XPathFactory.newInstance();
XPath xpath = factory.newXPath();
xpath.setNamespaceContext(new CognosNamespaceContext());
XPathExpression expr =
xpath.compile("/pre:dataSet/pre:dataTable//pre:row");

Object result = expr.evaluate(doc, XPathConstants.NODESET);

NodeList nodes = (NodeList) result;

/*
 * Set root activity
 */
IlvGanttModel ganttModel = new IlvDefaultGanttModel();
IlvSimpleActivity rootActivity = new IlvNullActivity();
ganttModel.setRootActivity(rootActivity);

/*
 * Set root resource
 */
IlvSimpleResource rootResource = new IlvNullResource();
ganttModel.setRootResource(rootResource);

/*
 * Read XML elements and feed it into the Gantt Model
 */
for (int i = 0; i < nodes.getLength(); i++) {
    Node row = nodes.item(i);

    Element name = this.findChildByName(row, "Name");
    String nameValue = name.getTextContent();

    String actityid = "activity_" + i;
    String resourceid = "resource_" + i;

    Element location = this.findChildByName(row, "Location");
    String locationValue = location.getTextContent();

    Element genre = this.findChildByName(row, "Genre");
    String genreValue = genre.getTextContent();

    Element startDate = this.findChildByName(row, "StartDate");
    String startDateValue = startDate.getTextContent();
    Date start = parseDate(startDateValue);

```

```

Element endDate = this.findChildByName(row, "EndDate");
String endDateValue = endDate.getTextContent();
Date end = parseDate(endDateValue);

Element reason = this.findChildByName(row, "Reason");
String reasonValue = reason.getTextContent();
IlvActivity activity = new IlvSimpleActivity(actityid, reasonValue,
    start, end);
ganttModel.addActivity(activity, rootActivity);
cacheActivityID(activity);

boolean contained = isParentResource(locationValue);
IlvSimpleResource parentResource = null;

if (!contained) {
    String parentResourceId = locationValue + "_resource" + "_" + i;
    parentResource = new IlvSimpleResource(parentResourceId,
locationValue,
    1);
    ganttModel.addResource(parentResource, rootResource);
    addParentResource(locationValue, parentResource);

    cacheResourceID(parentResource);
} else {
    parentResource = parentResourceMap.get(locationValue);
}

IlvSimpleResource resource=null;

Object obj=activityNameResourcesTable.get(nameValue);

if (obj==null){
    resource = new IlvSimpleResource(resourceid, nameValue,
    1);
    ganttModel.addResource(resource, parentResource);
    cacheResourceID(resource);
    activityNameResourcesTable.put(nameValue, resource);
}else{
    resource=(IlvSimpleResource) obj;
}

IlvActivity cachedActivity = getActivity(actityid);

```

```

        IlvResource cachedResource = getResource(resourceid);

        IlvSimpleReservation reservation = new IlvSimpleReservation(
            resource, cachedActivity);

        ganttModel.addReservation(reservation);
    }

    gantt.setGanttModel(ganttModel);

    IlvActivityTileLayout layout = new IlvActivityTileLayout(gantt);
    gantt.setActivityLayout(layout);

    /*
     * Remove unnecessary columns
     */
    IlvJTable table = gantt.getTable();
    TableColumn name = table.getColumn("Name");
    TableColumn ID = table.getColumn("ID");
    TableColumn Qty = table.getColumn("Qty");
    name.setPreferredWidth(200);
    table.removeColumn(ID);
    table.removeColumn(Qty);
    return ganttModel;
}

```

Important: The code described in Example 3-4 on page 107 is not the full project source but only integration-related code snippets. Before you move on to the next section and test your application, you must use the full project code. See Appendix A, “Additional material” on page 287 for details on how to download the complete code.

3.3.4 Testing the application

We have finished the development of the IBM JViews Human Resources sample application. Now we need to publish it to an application server. Perform the following steps to deploy the EAR project file:

1. In the Servers view, right-click your **WebSphere Application Server 7.0** and select **Add and Remove**.
2. In the Available projects list, select the **IBM Human ResourcesEAR** project, and click **Add** and then **Finish**.
3. After the **Publish** step completes, your service will be deployed and started.

4. Point your preferred browser to the following URL address:
http://localhost:8080/IBM_Human_Resources/faces/testTable.jsp

Figure 3-22 shows the IBM ILOG JViews Gantt chart developed in this section.

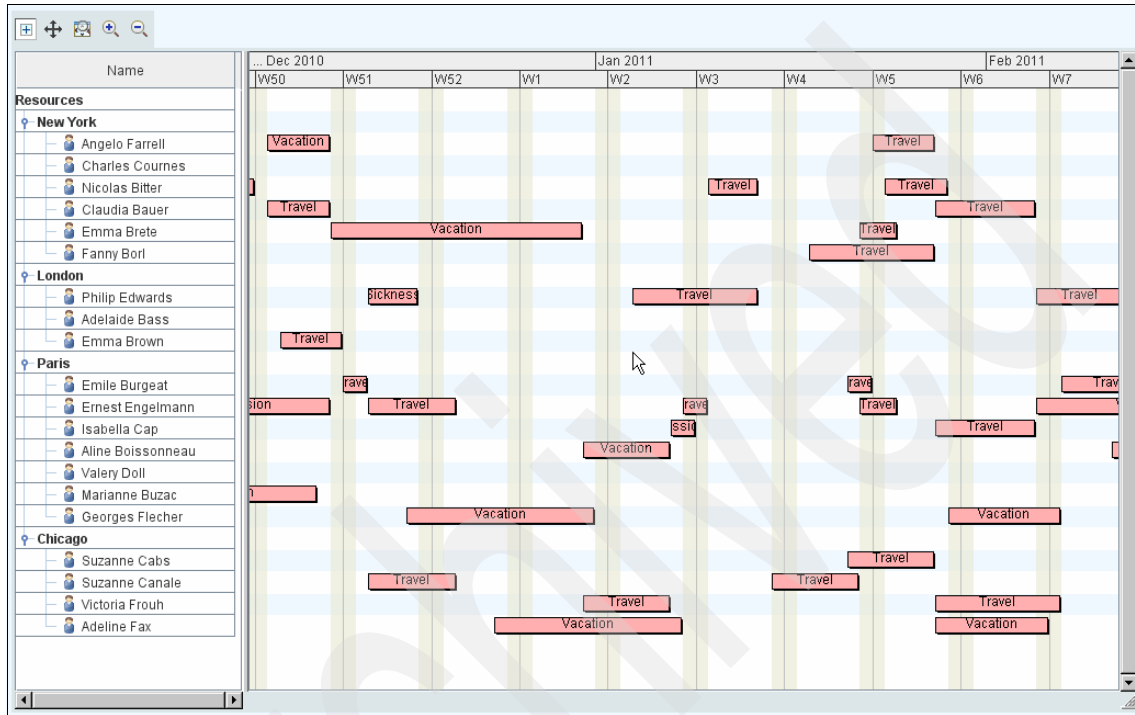


Figure 3-22 The IBM Human Resource Gantt chart

3.4 Integrating the ILOG Elixir Gantt component with IBM Cognos report

This section covers the integration of ILOG Elixir and IBM Cognos using a Cognos list report as a data source for the Elixir Gantt chart visualization.

3.4.1 Software requirements

In order to accomplish these tasks, you will need to have the following applications installed:

- IBM Cognos 10

- ▶ Adobe Flash Builder 4.0
- ▶ IBM ILOG Elixir Enterprise 3.0

3.4.2 Introduction to ILOG Elixir Gantt resource chart

A Gantt resource chart is a Gantt chart used to display the allocation or scheduling of resources. It is typically found in planning and production scheduling applications.

In IBM ILOG Elixir Enterprise, the ResourceChart control is a list of resources that display their properties in columns and their associated tasks in a timeline. Resources can be organized as a hierarchy, either to match a hierarchy of resources or to arrange them into groups or categories.

The resource chart control is a composite control composed of:

1. A data grid that displays the hierarchy of resources and their properties.
2. A Gantt sheet that displays the tasks arranged in time. Tasks are laid out in rows corresponding to the resources they are assigned to. The Gantt sheet header displays a time scale that supports navigating and zooming.

Figure 3-23 shows an example of a IBM ILOG Elixir Resource Chart.

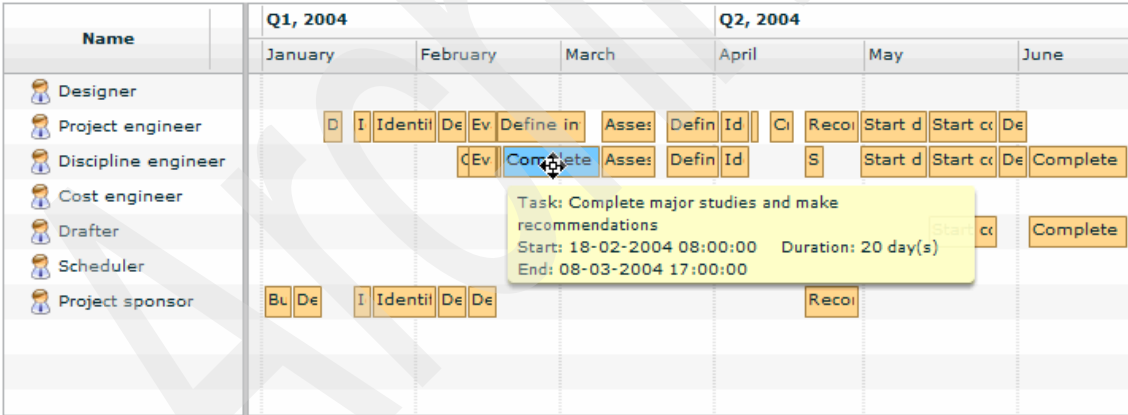


Figure 3-23 Resource chart

In our scenario, the resources are persons and the tasks are absence periods, such as vacation, travel, and illness periods.

3.4.3 Creating the IBM Human Resources Elixir Gantt chart project

In this topic we are going to create a new project called IBM Human Resources and customize it to display the Absences IBM Cognos report data.

In the Adobe Flash Builder application create a new Flex project:

1. Click **File** → **New** → **Flex Project**.
2. Type the project name as IBM Human Resources and click **Next**.
3. Accept bin-debug as the project output folder and click **Next**.
4. Setup the Framework Linkage field as **Merged into code**.
5. Click **Add SWC**.
6. Browse to the *installation path*/IBM/ILOG/Elixir Enterprise 3.0/frameworks/libs/ and select ilog-elixir-enterprise.swc.
7. Click **Open**, then **OK**.
8. Click **Add SWC** again and browse to *installation path*/IBM/ILOG/Elixir Enterprise 3.0/frameworks/locale/en_US/.
9. Select ilog-elixir-enterprise_rb.swc. Click **Open**, then **OK**.
10. In the **Main application file** field, replace the default Main.mxml file with humanresources.mxml. See Figure 3-24 on page 114.

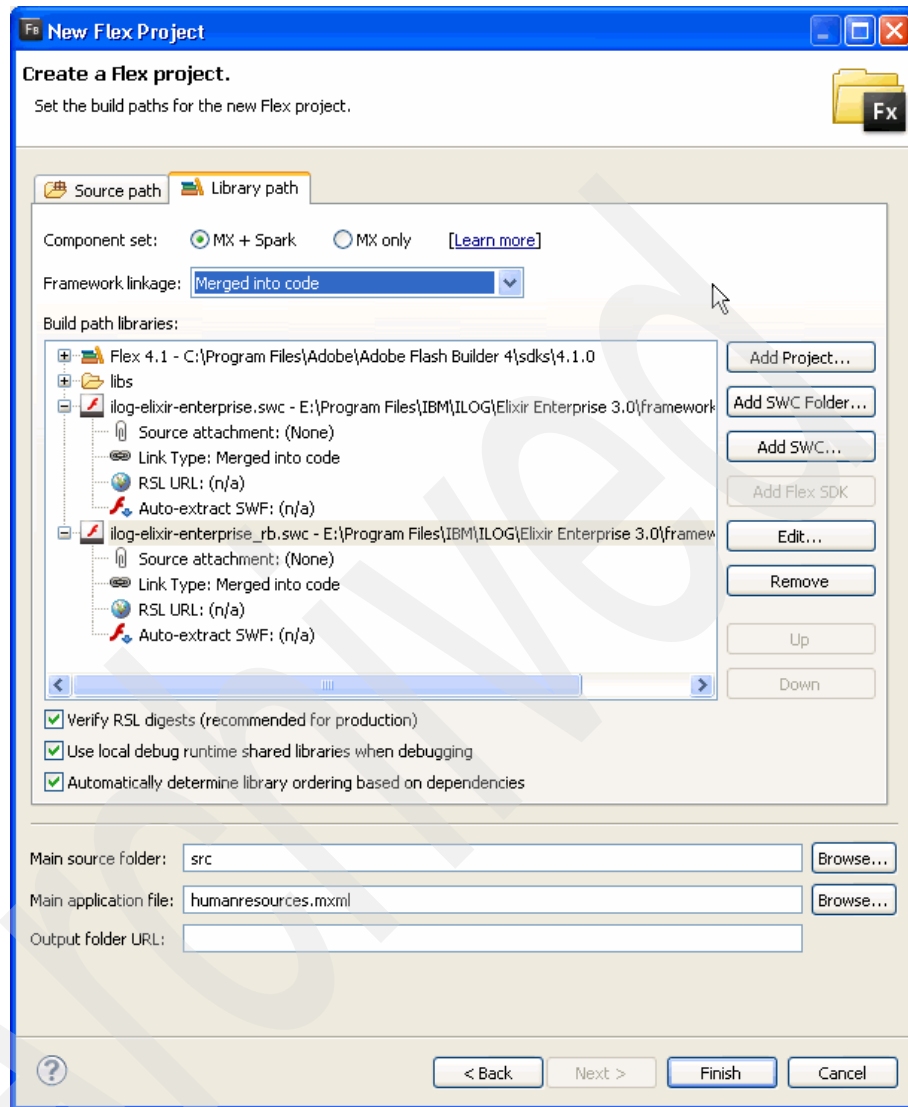


Figure 3-24 Project properties window

11. Click **Finish**.

Note: The source code for the Adobe Flash Builder project that is created in this section is available on the web. See Appendix A, “Additional material” on page 287 for details on how to download it.

Creating the application user interface

At this point, the `humanresources.mxml` file is created. This will be the main file of our application. We will now change the content of the file by the initial implementation of the application shown in Example 3-5.

Example 3-5 Initial code for `humanresources.mxml`

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:gantt="http://www.ibm.com/xmlns/prod/ilog/elixir/gantt/2010"
  xmlns:local="*"
  layout="absolute"
  backgroundColor="0xFFFFFFFF"
  applicationComplete="init()">
  <mx:Script>
    <![CDATA[
      private function init():void {}
    ]]>
  </mx:Script>
  <mx:VBox width="100%" height="100%" verticalGap="2"
    left="5" right="5" top="5" bottom="5" id="globalBox">
    <mx:HBox>
      <mx:Button label="Zoom In"
        click="{ganttSheet.zoom(.5,null,true);}"/>
      <mx:Button label="Zoom Out"
        click="{ganttSheet.zoom(2,null,true);}"/>
      <mx:Button label="Show All"
        click="{ganttSheet.showAll(-1, true);}"/>
    </mx:HBox>
    <mx:HBox width="100%" height="100%"
      horizontalGap="0" verticalAlign="top">
      <gantt:ResourceChart id="resourceChart"
        width="100%" height="100%" left="3" right="3">
        <gantt:dataGrid>
          <gantt:GanttDataGrid id="dataGrid" headerHeight="50"
            rowHeight="20" width="200"
            alternatingItemColors="{[0xFFFFFFFF, 0xDEDEDE]}">
            <gantt:columns>
              <mx:AdvancedDataGridColumn
                dataField="Name" headerText="Name"/>
            </gantt:columns>
          </gantt:GanttDataGrid>
        </gantt:dataGrid>
      <gantt:ganttSheet>
        <gantt:GanttSheet id="ganttSheet"
```

```

        alternatingItemColors="{[0xFFFFFFFF, 0xDEDEDE]}"
        moveEnabled="false" resizeEnabled="false"
        reassignEnabled="false"
        workingAlpha="0" nonWorkingAlpha="0"
        minVisibleTime="{new Date(2006,0,0)}"
        maxVisibleTime="{new Date(2013,0,0)}" >
    </gantt:GanttSheet>
</gantt:ganttSheet>
</gantt:ResourceChart>
</mx:HBox>
</mx:VBox>
</mx:Application>

```

This initial implementation is composed of an empty Elixir Enterprise Gantt component specified through the `<ResourceChart>` tag and three buttons: two buttons to zoom the Gantt time line in and out, and a third one to zoom the timeline to display all the data available.

Figure 3-25 shows the outline of the application in the Outline view of Flash Builder.

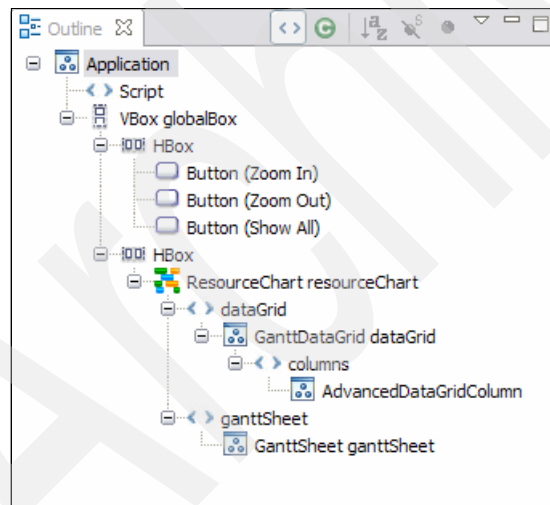


Figure 3-25 Outline of the application

The Gantt component itself is composed of two elements:

1. A `<dataGrid>` element that defines the grid on the left part of the Gantt
2. A `<GanttSheet>` section that defines the right part of the Gantt where the tasks are displayed along a timeline

In addition to configuration for the color, we have set configuration for the Gantt component:

- ▶ The <dataGrid> defines a single column with 'Name' as the header.
- ▶ The <ganttsheet> component has moveEnabled, resizeEnabled, and reassignEnabled properties set to false so that the Gantt is now in View mode only because we do not want to be able to edit tasks with the mouse.

Running the initial application

There is no data in the Gantt component yet, but we can run the application to see the Gantt chart component in action.

To run the application:

- ▶ Right-click the humanresources.mxm1 file in the project explorer panel and select **Run Application**.

Figure 3-26 shows the initial application with the three buttons and the Gantt component.

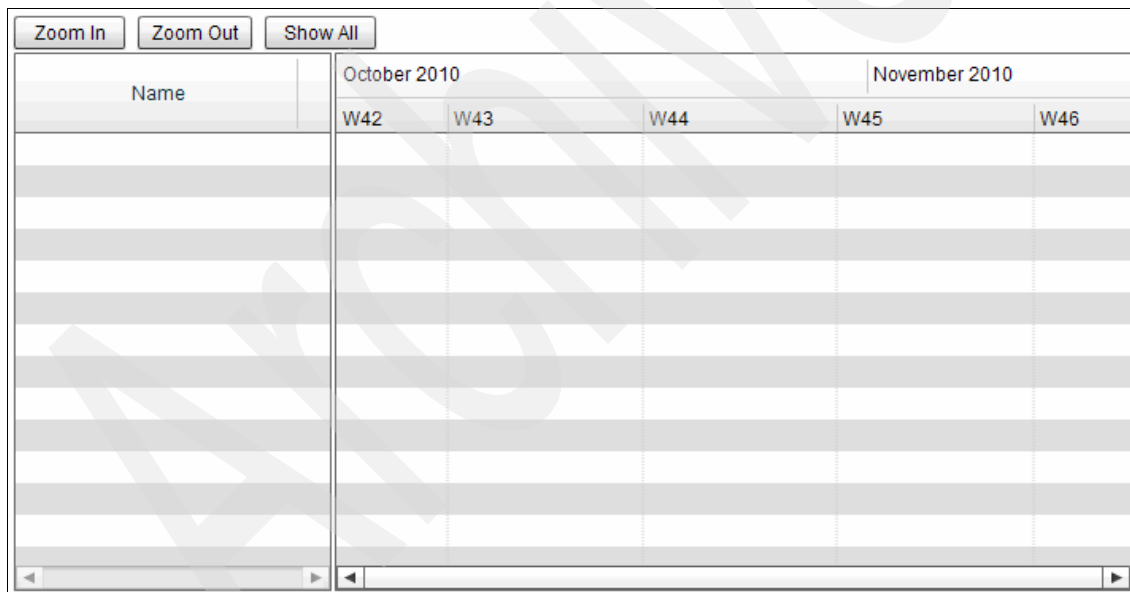


Figure 3-26 Initial empty Gantt chart

3.4.4 Calling Cognos Mashup Services (CMS) from Flex

The IBM Human Resource initial application has been created. We now need to see how we can populate the Gantt chart component with the data coming from

the Cognos report that we created in 3.2, “Creating the IBM Cognos sample report” on page 87.

For this purpose we create a new CMSConnector class that will be in charge of connecting to the Cognos data.

Go to **File** → **New** → **ActionScript File**. The file name must be CMSConnector. See Figure 3-27.

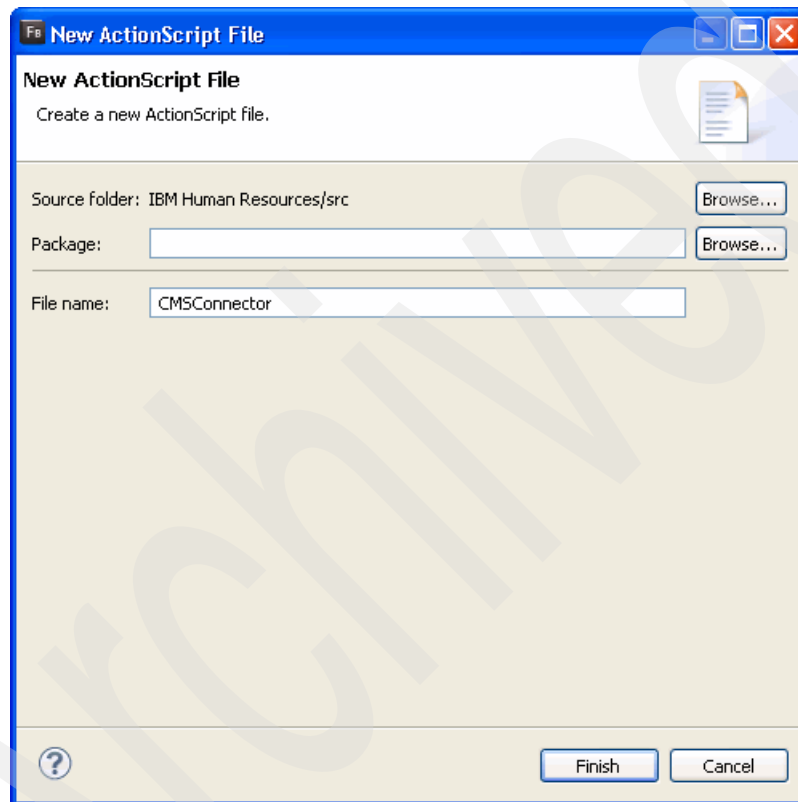


Figure 3-27 New ActionScript file - CMSConnector

The role of this ActionScript class is to do the REST call to the Cognos Mashup Services and prepare the result of the call so that we will be able to display the resulting data in our Gantt chart component.

Use the code in Example 3-6 on page 119 to fill the CMSConnector.as file. Note that this class only handles the CMS DataSet output and not all the CMS output type. In our scenario, we are only interested in the data that constitutes the report, not the report itself, and that is exactly the purpose of the DataSet output format that was introduced in Cognos version 10.

```
package
{
    import flash.events.EventDispatcher;
    import flash.events.IEventDispatcher;

    import mx.collections.ArrayCollection;
    import mx.collections.ArrayList;
    import mx.collections.ICollectionView;
    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;
    import mx.rpc.http.HTTPService;

    [Event(name="result", type="mx.rpc.events.ResultEvent")]
    [Event(name="fault", type="mx.rpc.events.FaultEvent")]
    public class CMSConnector extends EventDispatcher
    {
        private var service:HTTPService;
        private var dataSetURI:String;
        private var dataSet:Object;
        private var tableIds:ArrayList;

        public function connect(serverURI:String, reportPath:String):void
        {
            dataSetURI = serverURI+
                "/ibmcognos/cgi-bin/cognos.cgi/rds/reportData/path"+
                escape(reportPath)+
                "?async=OFF&fmt=DataSet";
            service = new HTTPService();
            service.addEventListener(ResultEvent.RESULT, onResult);
            service.addEventListener(FaultEvent.FAULT,
                function(e:FaultEvent) : void {
                    dispatchEvent(e);
                });
            service.url = dataSetURI;
            service.resultFormat = "e4x";
            service.send();
        }

        private function onResult(e:ResultEvent):void
        {
            var doc:XML = e.result as XML;
            dataSet = {};
            tableIds = new ArrayList();
        }
    }
}
```

```

        default xml namespace
        = new
Namespace("http://www.ibm.com/xmlns/prod/cognos/dataSet/201006");

        for each (var table:XML in doc..dataTable) {
            tableIds.addItem(table.id.toString());
            var rows:ArrayCollection = new ArrayCollection();
            for each (var row:XML in table.row) {
                var rowObject:Object = new Object();
                for each (var cell:XML in row.children())
                    rowObject[cell.name().localName] = cell.toString();
                rows.addItem(rowObject);
            }
            dataSet[table.id] = rows;
        }
        dispatchEvent(e);
    }

    public function getDataSet():Object
    {
        return dataSet;
    }

    public function getFirstTable():ICollectionView
    {
        return dataSet[tableIds.getItemAt(0)] as ICollectionView;
    }
}
}

```

The connect method is used to do the connection to the REST services. The first parameter is the URI to the server and the second parameter the path to the Cognos report.

To get data from a report, the default URI to the IBM Cognos Mashup Service REST interface is:

/ibmcognos/cgi-bin/cognos.cgi/rds/reportData/path/<report path>

where the *ibmcognos* folder name can change depending on your server configuration.

The IBM Cognos report path in this scenario will be:

► /Public Folders/ILOG Elixir HR/Absences

The parameters for the IBM Cognos Mashup Service used are:

- ▶ `fmt=DataSet`

We use this option to get the result data in the DataSet format.

You can render an IBM Cognos resource in the DataSet format when you are only interested in the data contained in the report output, and not in any formatting data (IBM Cognos 10 version only).

- ▶ `async=OFF`

This option specifies whether the report will run synchronously or asynchronously. In this case the option was to setup as OFF because we want the server to send us the actual data, not a redirection for the data.

After the URI is created, the method uses the Flex HTTPService class to call the REST service. The result format of the HTTPService is set to “e4x” so that the resulting data will be already parsed as XML.

The XML document that we receive is in the XML DataSet format defined by the namespace `http://www.ibm.com/xmlns/prod/cognos/dataSet/201006`. A DataSet document consists of a `dataSet` root element containing one or more `dataTable` elements. Each `dataTable` element corresponds to a report part, or to a page of report part output if the report is requested with page breaks.

In our scenario, the DataSet output will contain a single table because our report is a simple list report. Example 3-7 shows a fragment of the DataSet XML result.

Example 3-7 CMS DataSet result

```
<?xml version="1.0" ?>
<dataSet xmlns="http://www.ibm.com/xmlns/prod/cognos/dataSet/201006">
  <dataTable>
    <id>List1</id>
    <row>
      <Name>Angelo Farrell</Name>
      <Location>New York</Location>
      <Genre>male</Genre>
      <StartDate>2010-02-15T00:00:00.000</StartDate>
      <EndDate>2010-02-21T00:00:00.000</EndDate>
      <Reason>Travel</Reason>
    </row>
    <row>
      <Name>Angelo Farrell</Name>
      <Location>New York</Location>
      <Genre>male</Genre>
      <StartDate>2010-03-27T00:00:00.000</StartDate>
      <EndDate>2010-04-04T00:00:00.000</EndDate>
```

```
        <Reason>Travel</Reason>
    </row>
    ...
</dataTable>
</dataSet>
```

When the data is received from the Cognos CMS services, the `onResult` method is called. This method loops over each table and each row of the tables to build and store the resulting data as a collection of row objects: an instance of `Flex ArrayCollection` containing objects with properties corresponding to the name of the columns of the Cognos list report. Each row object has a `Name`, `Location`, `Genre`, `StartDate`, `EndDate`, and `Reason` property.

Finally the `onResult` method sends a `ResultEvent` to indicate that the data is ready.

In case of an error, a `FaultEvent` is sent to indicate the error when loading the data.

As indicated previously, in our scenario, we will only have a single table. The `getFirstTable` function in the `CMSConnector` will allow us to get this single table in the data set and populate our Gantt component.

3.4.5 Displaying CMS Data in the Gantt component

In this section we will perform the following tasks:

- ▶ Call the Cognos REST services through the `CMSConnector` class.
- ▶ Use the resulting data as data provider for the Gantt.
- ▶ Configure the necessary settings so that the Gantt chart can display the data.

Using the `CMSConnector` class

The `CMSConnector` class will allow us to call the Cognos REST service.

Let's go back to our main MXML file, `humanresources.mxml`. The initial implementation defines a block of Script code with an empty `init` method. This method is called when the application initialization is complete. We will use our `CMSConnector` class in this method.

Replace the `init` method with the new code shown in Example 3-8.

Example 3-8 initialization method using the `CMSConnector`

```
import mx.controls.Alert;
import mx.core.FlexGlobals;
```



```

import mx.rpc.events.FaultEvent;
import mx.rpc.events.ResultEvent;
import mx.utils.URLUtil;

private var cmsConnector:CMSConnector;

private function init():void
{
    cmsConnector = new CMSConnector();
    cmsConnector.addEventListener(ResultEvent.RESULT, dataLoaded);
    cmsConnector.addEventListener(FaultEvent.FAULT, onError );

    var loaderUrl:String =
        FlexGlobals.topLevelApplication.loaderInfo.url;
    var protocol:String = URLUtil.getProtocol(loaderUrl);
    var serverName:String = URLUtil.getServerNameWithPort(loaderUrl);

    if (protocol == "file") {
        protocol ="http";
        serverName = "MyServer:80";
    }

    cmsConnector.connect(protocol+"://" + serverName ,
        "/Public Folders/ILOG Elixir HR/Absences");
}

public function onError(evtObj:Event):void
{
    Alert.show("Error Loading data");
}

public function dataLoaded(evtObj:Event):void
{
}

```

Note: In this code, you will want to change the serverName from MyServer:80 to the actual name and port of the server where Cognos is installed before you run the code.

This new implementation of the init method now creates a CMSConnector instance and calls the CMSConnector.connect method to perform the REST call for our Cognos list report located at /Public Folders/ILOG Elixir HR/Absences, as shown in Example 3-9 on page 124.

Example 3-9 Connection to CMS

```
cmsConnector.connect(protocol+"://"+ serverName ,  
    "/Public Folders/ILOG Elixir HR/Absences");
```

The connect method requires the name of the server and the port (serverName variable) where Cognos is installed to be able to perform the CMS call. Obviously, we do not want to have a hard-coded URL in the code because it will force the code to be recompiled when the server changes. This is the reason why the code uses `FlexGlobals.topLevelApplication.loaderInfo.url`. This URL is the URL from which the Flex application is loaded. If we deploy the Flex application on the same server that is running Cognos, then the application can use this server name and port to do the CMS call. This is a simple way to avoid having a hard-coded server name in the code.

Note: You can see that we keep a hard-coded server name here when the protocol that loaded the Flex application is "file." The reason is that we still want to be able to debug our application using the Flash Builder debugger while the application we are testing is not yet deployed to the Cognos server. These lines can be removed when debugging is finished.

On the CMSConnector, two event listeners are registered:

- ▶ The `dataLoaded` method is called when the data from CMS is available (when the CMSConnector sends the `RESULT` event). This is how we will feed our Gantt component with data.
- ▶ The `onError` method is called in case of an error.

Filling the Gantt chart component with data

An Elixir resource chart component gets its data from two data sources: the resources displayed in the grid and the tasks assigned to the resources displayed in the timeline.

- ▶ The resources can be a flat list or can contain hierarchical or grouped resource data. It is set using the `resourceDataProvider` property of the `ResourceChart` object.
- ▶ The task data provider is a list-based data provider. Specify the task data for the `ResourceChart` control by using the `taskDataProvider` property.

For the list of tasks, we can simply use our data table coming from the CMS call like shown in Example 3-10 on page 125.

Example 3-10 Setting the taskDataProvider

```
resourceChart.taskDataProvider = cmsConnector.getFirstTable();
```

For the resources, we do not already have the list of persons. Our data table has a 'Name' column. Using this 'Name' property we can compute the list of unique resources by extracting from the task list a list of elements with unique names, as shown in Example 3-11.

Example 3-11 Setting the resourceDataProvider

```
var resources:ArrayCollection = new ArrayCollection();

var cache:Object = {};
for each (var row:Object in cmsConnector.getFirstTable()) {
    if (!cache[row['Name']]){
        resources.addItem(row);
        cache[row['Name']] = true;
    }
}

resourceChart.resourceDataProvider = resources;
```

Now the Gantt component has a data provider defined for the resources and one for the tasks.

We can improve this a little using the grouping and sorting features of the collection system in Flex. We want to group the persons by the city they live in (indicated by the Location field in the data) and also sort the names in alphabetic order.

For this we use the GroupingCollection2 class in Flex that is used to group data, and the Sort class.

Example 3-12 shows the implementation of the dataLoaded method with sorting and grouping.

Example 3-12 dataLoaded method

```
public function dataLoaded(evtObj:Event):void
{
    var resources:ArrayCollection = new ArrayCollection();

    var cache:Object = {};
    for each (var row:Object in cmsConnector.getFirstTable()) {
        if (!cache[row['Name']]) {
```

```

        resources.addItem(row);
        cache[row['Name']] = true;
    }
}

// grouping resources by Location
var groupedResources:GroupingCollection2
    = new GroupingCollection2();
groupedResources.source = resources;
var grouping:Grouping = new Grouping();
grouping.fields = [new GroupingField('Location')];
groupedResources.grouping = grouping;
groupedResources.refresh();

// Sorting by Name
var hierarchicalResources:HierarchicalCollectionView
    = new HierarchicalCollectionView(groupedResources);

var sort:Sort = new Sort();
sort.fields = [new SortField("Name")];
hierarchicalResources.sort = sort;

resourceChart.resourceDataProvider = hierarchicalResources;
resourceChart.taskDataProvider = cmsConnector.getFirstTable();

dataGrid.expandAll();
ganttSheet.showAll();
}

```

When data is loaded, the `dataGrid.expandAll()` method will expand all rows of the grid. Otherwise the locations are collapsed by default.

The `ganttSheet.showAll()` call adjusts the zoom level of the timeline so that all tasks are visible.

Configuration of the Gantt chart component

Because the Gantt component has two separate collections, one for the resources and one for the tasks, you might wonder how the component knows which task is associated to which resource. For the moment, the component cannot make this association. If you run the application at this step, you will not see any tasks in the timeline.

In order to make this association, we must specify two properties on the `ResourceChart` component:

- ▶ The `taskResourceIdField` property of the `ResourceChart` specifies the name of a field of the task object that gives the ID of the resource associated with the task.
- ▶ The `resourceIdField` property of the `ResourceChart` specifies the name of a field of the resource object that gives the ID of this resource.

The resource chart is able to make the association between task and resources thanks to this referencing by IDs. In our scenario, we identify a resource by its name. Both properties should be set to the value "Name." Example 3-13 on page 127 shows the modified `ResourceChart` tag in the `mxm1` file.

Example 3-13 ResourceChart tag

```
<gantt:ResourceChart id="resourceChart"
    width="100%" height="100%" left="3" right="3"
    taskResourceIdField="Name"
    resourceIdField="Name">
```

The last step that is required to be able to display the tasks is to tell the component how to find the start and end dates of a task.

The `taskStartTimeFunction` and `taskEndTimeFunction` properties of the `ResourceChart` are there to specify methods to retrieve the dates from the task object.

In our Cognos data, the dates are stored in the `StartDate` and `EndDate` columns of the data table. The values we have retrieved are string values, we will have to parse the string to create a Flex Date object as shown in Example 3-14.

Example 3-14 getStartTime and getEndTime

```
private static function parseDate(value:String):Date {

    var dateStr:String = String(value)
    dateStr = dateStr.replace(/\-/g, "/");
    dateStr = dateStr.replace("T", " ");
    dateStr = dateStr.replace(/\.[0-9]+\./, "");
    dateStr = dateStr + " GMT-0000";

    var date : Date = new Date(Date.parse(dateStr));

    date.hours = date.minutes = date.seconds = date.milliseconds = 0;
    return date;
}

private function getStartTime(data:Object):Date
```

```

{
    return parseDate(data["StartDate"]);
}

private function getEndTime(data:Object):Date
{
    var date:Date = parseDate(data["EndDate"])
    date.setHours(23,59,59,999);
    return date;
}

```

The `parseDate` method converts the date string in the format returned by Cognos (YYYY-MM-DDTHH:MM:SS.mmm) to a format that Flex can parse (YYYY/MM/DD HH:MM:SS GMT-0000).

Note that in our scenario, we are only working with date and not time information. The Gantt component uses the date and time information to display the task. In order to have the last day being part of the displayed task, we set the time to the end of the day in the `getEndTime` method.

Example 3-15 shows the modified *ResourceChart* tag in the `mxm1` file.

Example 3-15 ResourceChart tag

```

<gantt:ResourceChart id="resourceChart"
    width="100%" height="100%" left="3" right="3"
    taskResourceIdField="Name"
    resourceIdField="Name"
    taskEndTimeFunction="getEndTime"
    taskStartTimeFunction="getStartTime">

```

After this step, all the elements necessary to display the data in the Gantt component are in place.

Additional graphic improvements

Although it is not the subject of this book to describe all the possible customizations of the appearance of the Gantt chart component, we will show now how to create a simple custom rendering for the tasks in the Gantt. We want to display the bars of the Gantt in different colors depending on the type of absence the task represents. Our data set defines four types of absences: Travel, Sickness, Vacation, and Mission.

To have a different color for each type of absence, we create a new renderer for the task:

1. Right-click the project in Flash Builder and choose **New** → **Action Script File**
2. In the window, type CustomTaskItemRenderer for the **File name**.
3. Replace the code of CustomTaskItemRenderer.as with the code in Example 3-16.

Example 3-16 CustomTaskItemRenderer.as

```
package
{
    import com.ibm.ilog.elixir.gantt.TaskItemRenderer;

    public class CustomTaskItemRenderer extends TaskItemRenderer
    {
        private static const reasonToColor:Object = {
            Travel: 0x65844d,
            Mission: 0xb19f80,
            Sickness: 0x965151,
            Vacation: 0x6a90a4
        };

        override protected
            function updateDisplayList(w:Number, h:Number):void
        {
            this.setStyle("backgroundColor" ,
                reasonToColor[this.data.data['Reason']]);
            super.updateDisplayList(w,h);
        }
    }
}
```

This simple renderer derives from the standard renderer (TaskItemRenderer) and changes the backgroundColor of the bar according to the type of absence.

In order to use this new renderer, set the taskItemRenderer property to CustomTaskItemRenderer in the <GanttSheet> tag.

Note: Many possible customizations are possible in the ResourceChart component. For more information about customization, examine the examples that are provided with ILOG Elixir Enterprise and in particular the Human Resources Example that are close to our scenario and show various possibilities such as bar renderer using gradients with a layout depending on the zoom levels.

Testing the application

Now that our application is complete, we can test it from Flash Builder:

- **Right-click** the `humanresource.mxml` file in the project explorer panel and select **Run application**.

Figure 3-28 shows the ILOG Gantt chart using the IBM Cognos list report data.

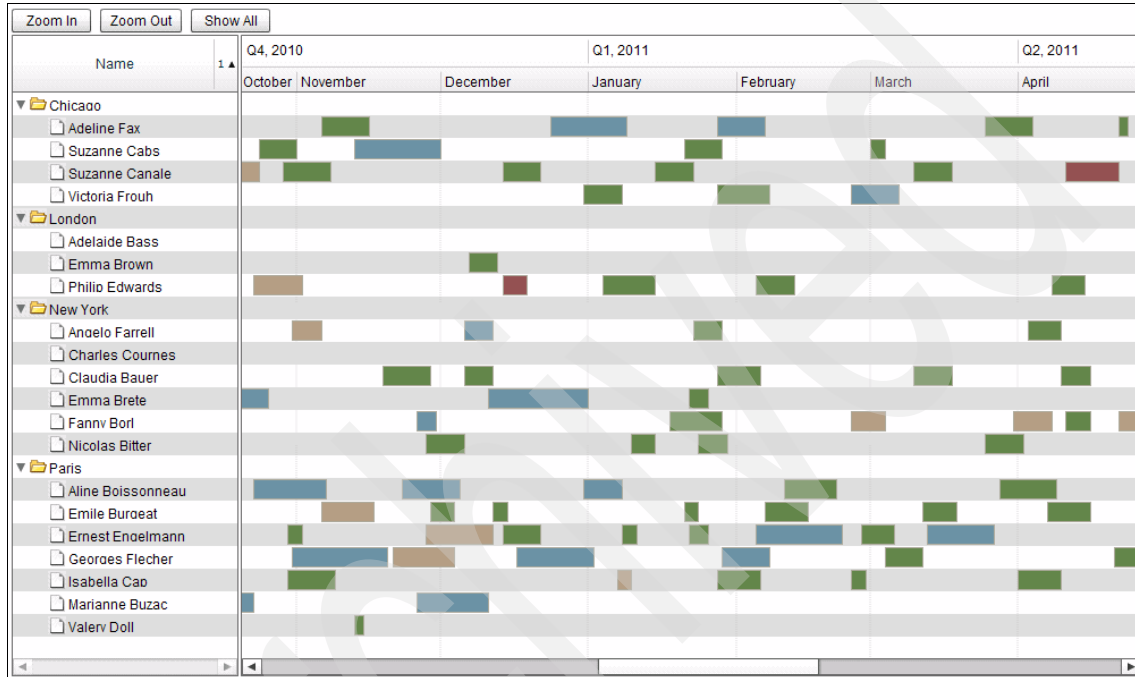


Figure 3-28 The Human Resource application

3.4.6 Deploying the code into the IBM Cognos IBM Cognos server

The following are the steps for deploying the compiled code to the IBM Cognos server that will be used as part of the Business Insight portal when we cover the publishing process of the ILOG visualizations.

1. Browse to locate the Flex project bin-debug folder:
`C:\Documents and Settings\<user_name>\Adobe Flash Builder 4\IBM Human Resources\bin-debug`
2. Copy the following files to the `IBM Cognos_installation\webcontent` folder:
 - `humanresources.html`

The html file that opens the Flex application.

- `humanresources.swf`
The compiled Flex application.
- `playerProductInstall.swf`
A Flex application for upgrading the Flash player.
- `swfobject.js`
JavaScript library to embed the Flex application in the HTML file.

Test the visualization now by pointing your browser to:

`http://IBM_Cognos_server/<IBM_Cognos_installation>/humanresources.html`

Note: This chapter does not cover the IBM Cognos Authentication process because we will be publishing the code on the IBM Cognos Business Insight page. We assume that the user must be authenticated before visualizing the IBM Cognos content.

Authentication might be required to report the content if you are deploying your ILOG visualizations outside of an IBM Cognos server. In this case, you will need to use the REST services of Cognos to authenticate with the Cognos server.

3.5 Publishing the ILOG Visualizations to IBM Cognos Business Insight

It is time to publish our ILOG Visualizations to the IBM Cognos Business Insight dashboard to accomplish our integration scenario.

The next steps will show how to create a Business Insight dashboard presenting the ILOG visualization:

1. Open your browser and go to your IBM Cognos application welcome page.
2. In the **My Actions** section, click **Create my dashboards**.
3. In the IBM Cognos Business Insight welcome page, click **Create New** as shown in Figure 3-29 on page 132.



Figure 3-29 The IBM Cognos Business Insight welcome page

4. In the bottom of the right section, click **Toolbox**.
5. Select and drag a new Web Page item to the right section. See Figure 3-30.

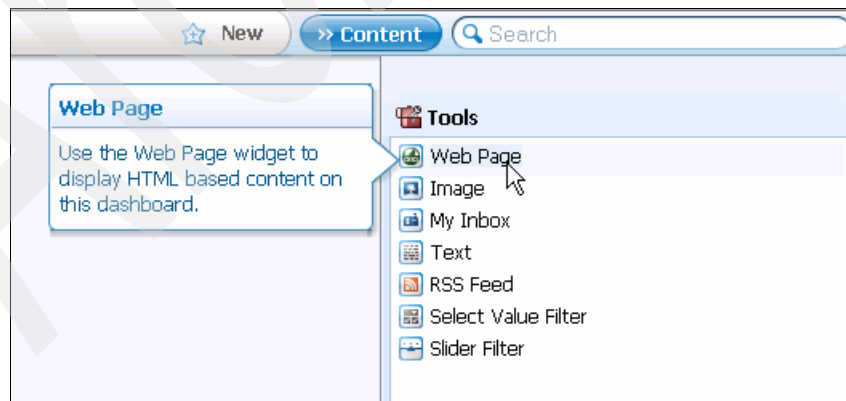


Figure 3-30 The Web Page widget item

6. You will be prompted to fill the link of the visualization content. The address of the sample page will be similar to the following:

http://cognos_server:80/cognos_installation/humanresources.html

Important: The publishing process for an ILOG JViews visualization in the Business Insight dashboard is the same. The difference is that the JViews visualization must be deployed in an *application server* instead being deployed to the IBM Cognos web content folder.

The address used by the ILOG JViews Gantt chart must be:

http://server_address:8080/IBM_Human_Resources/faces/testTable.jsp

Figure 3-31 shows the IBM ILOG JViews Gantt chart dashboard.

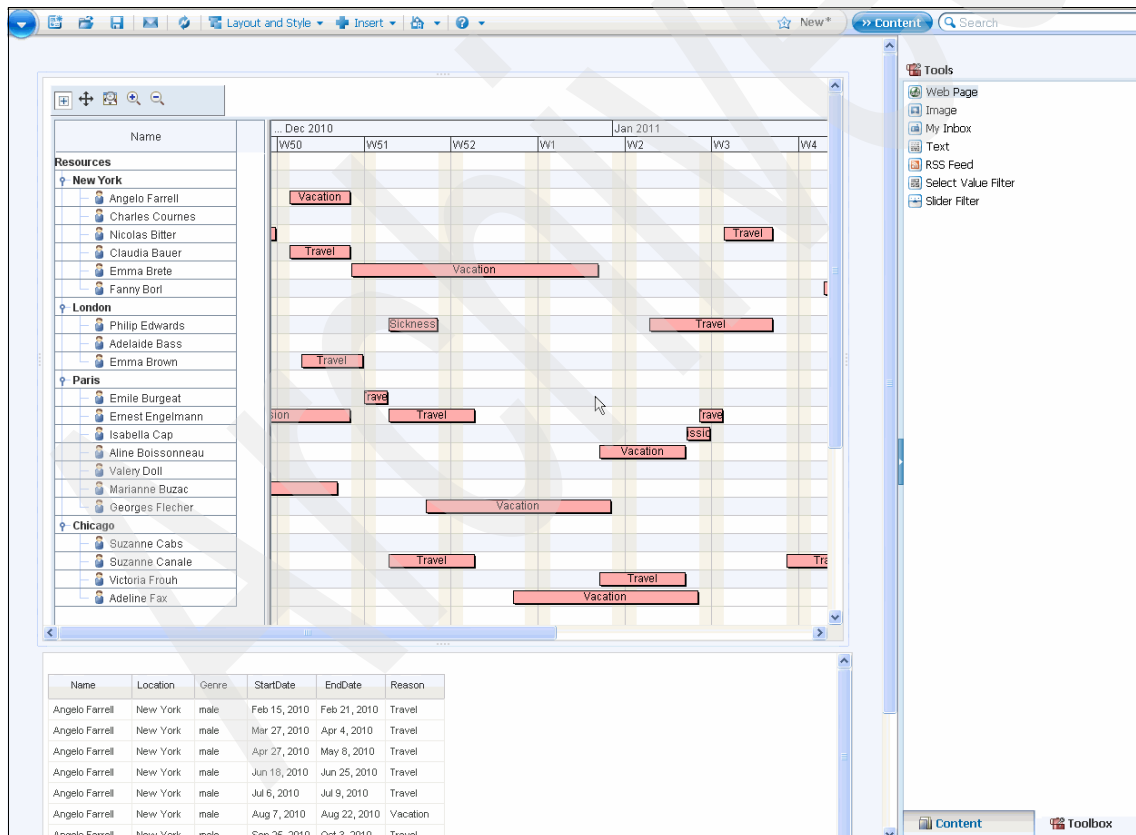


Figure 3-31 The ILOG JViews Gantt chart visualization in the IBM Cognos Business Insight dashboard

Figure 3-32 shows the IBM ILOG Elixir Gantt chart dashboard.

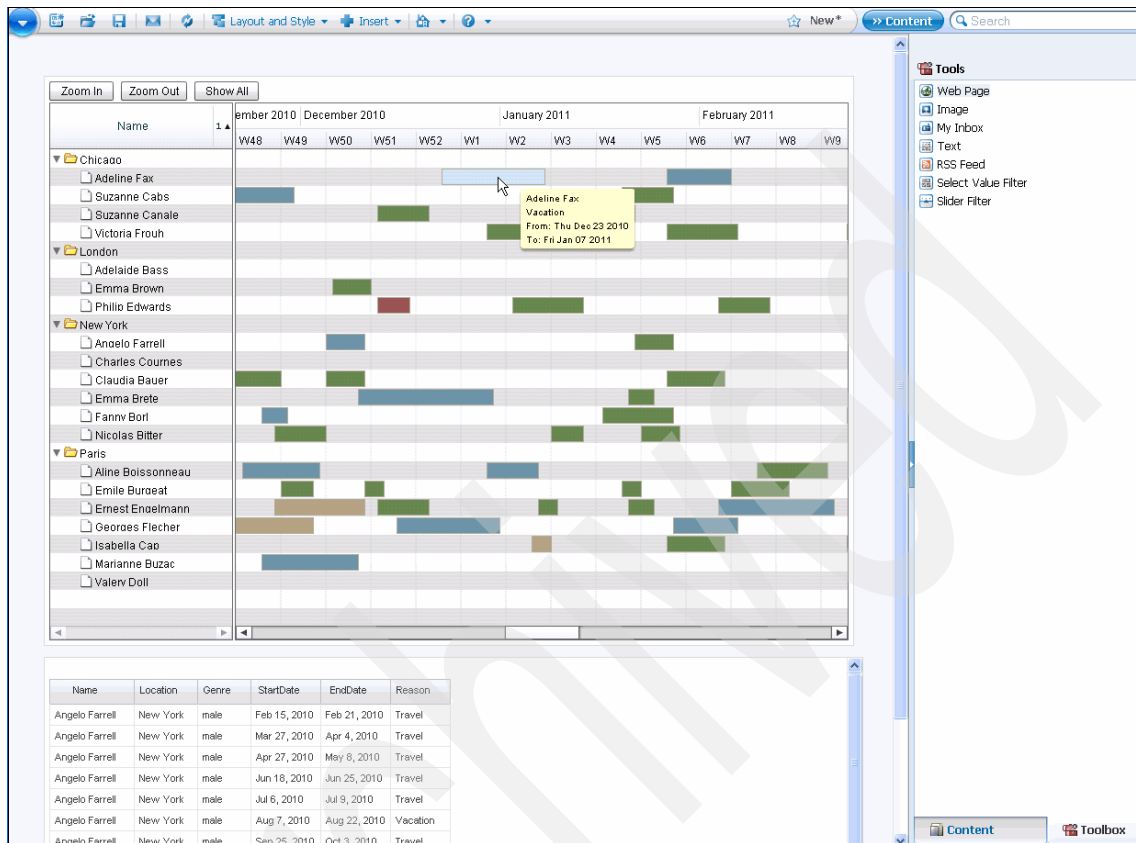


Figure 3-32 The ILOG Elixir Gantt chart visualization in the IBM Cognos Business Insight dashboard

Note: We have included the Absences sample report in the same dashboard to illustrate the differences of displaying the same information in a simple listing report and using the IBM ILOG Visualization products. Refer to the official IBM Cognos documentation for more information about how to add other widgets to a dashboard.

Our proposed scenario for this chapter is now complete. We created an IBM Cognos report and loaded its data into the IBM ILOG JViews and Elixir visualization products. Finally, we integrated them as part of the IBM Cognos Insight portal. The information provided by this chapter should give you a starting point if you are planning to integrate the IBM Cognos content with the IBM ILOG Visualization modules for your project needs.

Creating IBM ILOG Elixir iWidgets for Mashup Center

This chapter describes how an IBM ILOG Elixir component can be wrapped as an iWidget so it can be deployed into IBM Mashup Center.

It gives step-by-step instructions for creating an iWidget wrapper for the IBM ILOG Elixir Treemap component. It goes on to describe how to make sure data and selection is synchronized between the component and the mashup context.

Finally, the wrapper is demonstrated in IBM Mashup Center by creating a mashup application using a treemap to analyze complex data.

Note: The source code for the Adobe Flash Builder project and the Rational Application Developer project that are created in this chapter are available on the web. Check Appendix A, “Additional material” on page 287 for details on how to download them.

4.1 Creating an iWidget wrapper for an IBM ILOG Elixir component

As described in the introduction section 1.3, “What is IBM ILOG Elixir Enterprise” on page 21, IBM ILOG Elixir is a set of advanced visualization components based on the Adobe Flex platform. The components target Adobe Flex applications in general, but they can be when building a mashup application, particularly when used in IBM Mashup Center.

To enable this, the IBM ILOG Elixir components must be wrapped into an iWidget component, which is the specified way of creating a widget for IBM Mashup Center. After this is done, the iWidget can be deployed in IBM Mashup Center catalog and the user can use the IBM ILOG Elixir component as part of its mashup applications.

Wrapping the IBM ILOG Elixir component as an iWidget increases its interoperability, allowing you to re-use it in any iWidget container including Business Space powered by WebSphere as shown in Chapter 5, “Business Monitoring with ILOG Elixir and WebSphere Business Space” on page 177. Creating an iWidget is as simple as providing the XML descriptor describing the events and modes of your widget, and optionally providing the JavaScript code that manages it.

This section describes how the IBM ILOG Elixir Treemap component can be wrapped as an iWidget. The treemap component allows the user to easily analyze two dimensions of a data set in a compact display. As shown in Figure 4-1 on page 137, one dimension of the data is conveyed through the size of the treemap cells and the other dimension is conveyed through the color of the cells.

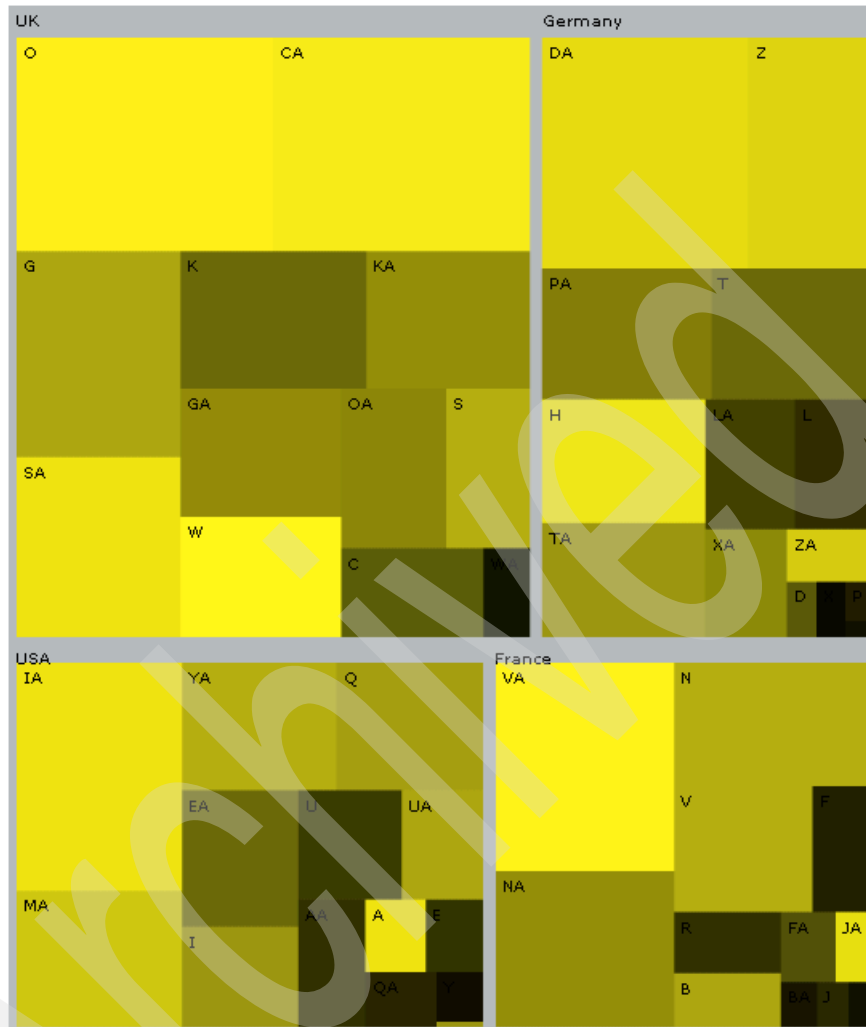


Figure 4-1 Treemap showing revenues and profit of companies clustered by country

If we consider that the first dimension conveys the revenue of a company and the second dimension conveys its profit, we can easily spot companies with large revenues and big profits (big cells with a bright yellow color). Conversely, small revenue companies with little profit are easily ignored (small cells, with a darker color). This is the kind of multi dimensional analysis the treemap can help to perform.

From this example, we see how useful it might be to use such components in a mashup application. The treemap component was chosen simply for illustration purposes; it is important to realize that a similar technique can be used for any

IBM ILOG Elixir component you want to use in IBM Mashup Center. As all IBM ILOG Elixir components can be incorporated similarly, users can benefit from the other advanced visualization components of ILOG Elixir in a mashup context.

4.1.1 Software requirements

This section requires you to install the following products or their trial version:

- ▶ IBM ILOG Elixir 3.0 or IBM ILOG Elixir Enterprise 3.0
- ▶ IBM Rational Application Developer 8.0
- ▶ Adobe Flash Builder 4.0 (optional, but highly recommended)

4.1.2 Creating an iWidget in Rational Application Developer

The first task to achieve is to build the skeleton of an iWidget for the treemap in Rational Application Developer using following steps:

1. Click **File** → **New** → **Dynamic Web Project**
2. A wizard shows up where you can configure the project. See Figure 4-2 on page 139.

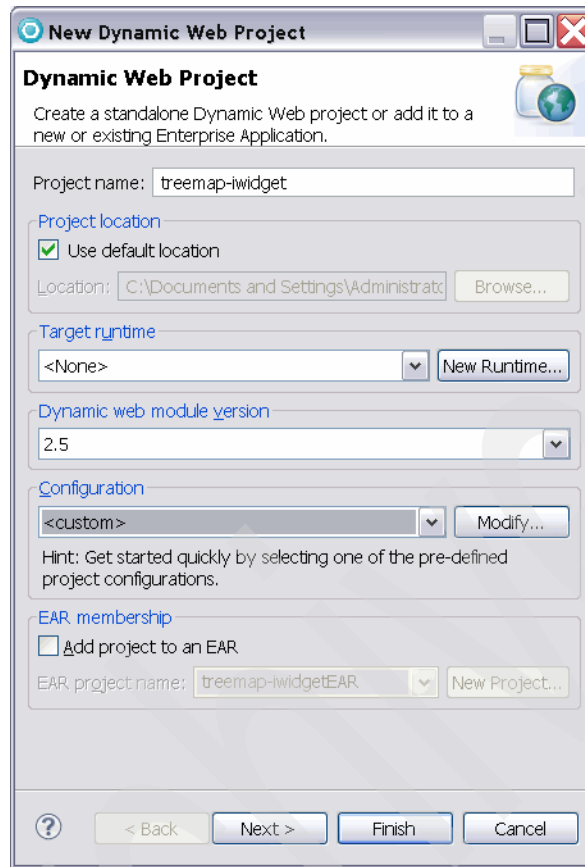


Figure 4-2 New Dynamic Web Project wizard

3. After this is done, a treemap-iwidget project is created in the workspace. Right-click the project name and click **Properties**, search for facets and you will be able to add the **iWidget** and **Web 2.0** → **Dojo** facets as shown in Figure 4-3 on page 140.

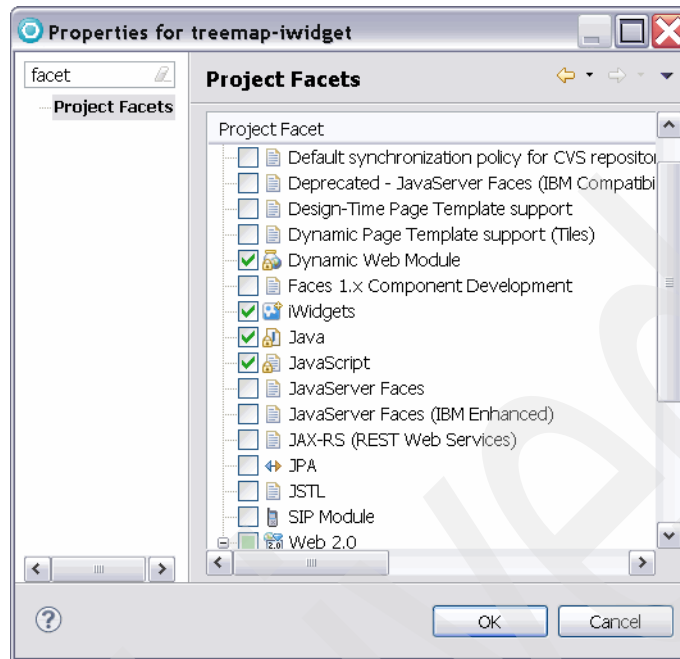


Figure 4-3 Add iWidget and Dojo facets to the project

4. The project is now ready to accept a new iWidget. Right-click the project name and click **New** → **iWidget**.
5. A wizard displays. Type `treemapWidget` for the widget name. Because we want the widget to accept data as an input event and to be editable, select the options **Event Subscriber Widget** and **Edit mode** as shown in Figure 4-4 on page 141.

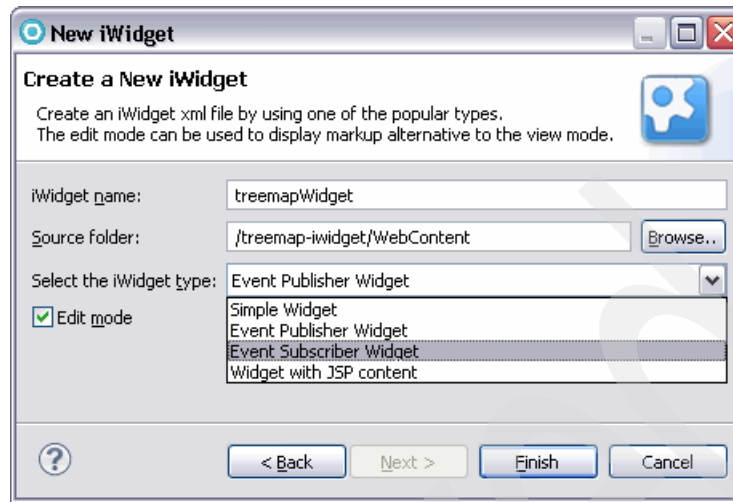


Figure 4-4 New iWidget wizard

6. After clicking **Finish**, the iWidget is created and the design view of the iWidget XML descriptor is displayed. See Figure 4-5.

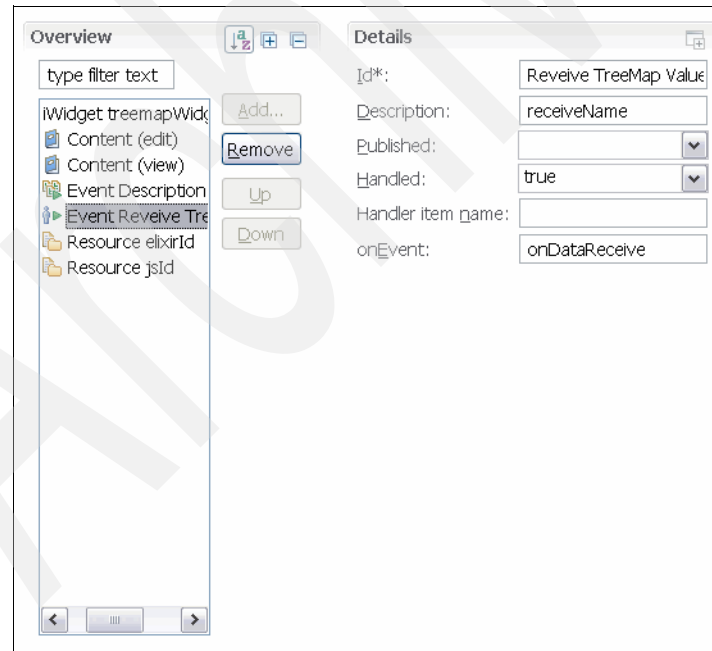


Figure 4-5 iWidget XML descriptor design view

In this editor, you can change several properties of the widget. Another way to proceed is to click the **Source** tab at the bottom of the view and alter the XML source of the descriptor. We need to modify this descriptor to build our iWidget:

- a. Change the iScope of the iWidget to `ibm_ilogvisu_redbook.treemapWidget`. This way, we avoid a potential clash with another JavaScript object called `treemapWidget`.
- b. Change the id of the receiveEvent event to `Receive TreeMap Value` and change its onEvent handler to `onDataReceive`.
- c. Change the content view template to the content of Example 4-1.

Example 4-1 View mode content template

```
<div id="_IWID_treemapWidgetViewHolder"></div>
```

- d. Add a new resource and set the `src` attribute to `elixir.js`. Also, create a JavaScript source file and name it `elixir.js`. The file consist of the code from Example 4-2.

This mediator code will register iWidgets and make their methods available to the Adobe Flex application using the IBM ILOG Elixir component. This will synchronize the iWidget context with events actually happening on the Flex side of the widget. Adding this file as a resource in the XML descriptor will make sure that the iWidget container will load it when the iWidget itself is loaded.

Example 4-2 elixir.js

```
var ibm_ilogvisu_redbook = {};  
  
(function () {  
    var hash = {};  
  
    /**  
     * Call a iWidget function from  
     * the Flex side.  
     */  
    ibm_ilogvisu_redbook.invokeWidget =  
        function(widgetId, fnName, params) {  
            var widget =  
                ibm_ilogvisu_redbook.getWidget(widgetId);  
            if (widget) {  
                var fn = widget[fnName];  
                if (fn) {  
                    try {
```

```

        return fn.call(widget, params);
    } catch (e) {
        console.log("error on function call");
    }
    } else {
        console.log("can't find function");
    }
    } else {
        console.log("can't find widget");
    }
    }
    return undefined;
};

/**
 * Get a handle on iWidget function.
 */
ibm_ilogvisu_redbook.getWidget = function(widgetId) {
    return hash[widgetId];
};

/**
 * Register a iWidget.
 */
ibm_ilogvisu_redbook.addWidget = function(widgetId, widget) {
    hash[widgetId] = widget;
};

/**
 * De-register a iWidget.
 */
ibm_ilogvisu_redbook.removeWidget = function(widgetId) {
    delete hash[widgetId];
};
})();

```

7. After the XML descriptor is ready, open the `treemapWidget.js` file and start coding the widget itself. You need to modify the name of the Dojo class to be identical to the one given in the XML descriptor, that is, `ibm_ilogvisu_redbook.treemapWidget`. You can also add variables that will be useful for the next steps. After this is done you will have something similar to Example 4-3.

Example 4-3 Initial version of `ltreemapWidget.js`

```

dojo.declare("ibm_ilogvisu_redbook.treemapWidget", null,{

```

```

// hold a reference to the data
data : null,
// hold a reference to the id of the Flex object
flexId : null,
// hold a reference to selection
selection : null,
// the widget name
widgetName : "treemapWidget",
// available fields
fields : null,

onDataReceive:function(/*com.ibm.mashups.iwidget.IEvent*/ event){
},

onLoad:function() {
},

onUnload:function() {
},

onView:function() {
},

onEdit:function() {
},

onSizeChanged:function(/*com.ibm.mashups.iwidget.IEvent*/ event) {
    var newWidth = event.payload.newWidth;
    var newHeight = event.payload.newHeight;
},

onNavStateChanged:function(/*com.ibm.mashups.iwidget.IEvent*/
event) {
}
});

```

4.1.3 Creating the Adobe Flex application for the iWidget

Now that the skeleton of the iWidget has been created, we need to actually prepare an IBM ILOG Elxir treemap to be used in the iWidget. For that, we will use the Adobe Flex IDE named Adobe Flash Builder.

1. In Flash Builder, click **File** → **New** → **Flex Project** and give it the name `treemapWidget`. This creates a blank Adobe Flex project.

2. Right-click the project name and click **Add IBM ILOG Elixir Libraries**. This adds the IBM ILOG Elixir SWC files to the project. These SWC files are the files that contain the actual components, including the treemap.

Note: In order to get the right menu, IBM ILOG Elixir must have been installed after Adobe Flash Builder and the option to install IBM ILOG Elixir on top of Adobe Flash Builder must have been chosen. If that is not the case, you will have to add the IBM ILOG Elixir libraries to the project manually. For that, check the IBM ILOG Elixir documentation.

3. Type the contents of Example 4-4 in the `tremapWidget.mxml` file, which is the main file of the Flex application.

Example 4-4 Adobe Flex application

```
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
               xmlns:s="library://ns.adobe.com/flex/spark"
               xmlns:ilog="http://www.ilog.com/2007/ilog/flex"
               preinitialize="preinitializeHandler(event)">

<fx:Script>
<![CDATA[
    import mx.collections.Grouping;
    import mx.collections.GroupingCollection2;
    import mx.collections.GroupingField;
    import mx.core.FlexGlobals;
    import mx.events.FlexEvent;

    private var _data:Array;
    private var _labelField:String;

    /**
     * Initialize the link between Flex & the browser
     * iWidget context for the handleData method to be called
     */
    private function preinitializeHandler(event:FlexEvent):void
    {
        ExternalInterface.addCallback("onDataReceive",
            onDataReceive);
    }

    /**
     * Method called when data to be displayed changed. We
     * expect to get data as:
     * { value : [ table of data objects ],
     *   areaField : "data field to use for area",
```

```

*   colorField : "data field to use for color",
*   labelField : "data field to use for labels",
*   groupingField : "data field to group data"
* }
*/
private function onDataReceive(data:*):void
{
    _data = data.value as Array;
    _labelField = data.labelField;
    var gc:GroupingCollection2 = new GroupingCollection2();
    gc.source = data.value;
    var gr:Grouping = new Grouping();
    var field:GroupingField = new
GroupingField(data.groupingField);
    field.groupingObjectFunction = groupingFunction;
    gr.fields = [ field ];
    gc.grouping = gr;
    gc.refresh();
    treemap.areaField = data.areaField;
    treemap.colorField = data.colorField;
    treemap.labelField = data.labelField;
    treemap.dataProvider = gc;
}

private function groupingFunction(label:String):Object
{
    var obj:Object = {};
    obj[_labelField] = label;
    return obj;
}
]]>
</fx:Script>
<ilog:TreeMap id="treemap" width="100%" height="100%"
    labelThreshold="2" />
</s:Application>

```

The `<ilog:TreeMap>` tag is there to instantiate the treemap itself. Right now it is empty of data. The `onDataReceive` method receives the data from the `iWidget` side and feeds the IBM ILOG Elixir treemap with it. The `preinitilazeHandler` is there to coordinate the process and make sure that after the `onDataReceive` method is called in JavaScript, the same method gets called on the Flex application.

The data received by the `onDataReceive` method contains five properties:

- `value`: An array containing the actual data in a flat form (each slot of the array contains a piece of data).
 - `groupingField`: The name of the property to look at in the data to create a hierarchy of cells from the initial flat data. For example, if you have data for several countries, you can use the “country” value as `groupingField` to make sure data will be grouped by country.
 - `areaField`: The name of the property to look at in the data to compute the size of a given treemap cell.
 - `colorField`: The name of the property to look at in the data to compute the color of a given treemap cell.
 - `labelField`: Contains the name of the property to look at in the data to display the label of a given cell.
4. Because Adobe Flash Builder automatically compiles the project, you should now have a `bin-debug` folder in your project that contains the debug version of the treemap application. If you launch it, nothing will appear as no data has been transmitted to the treemap.

However in the next section we will use the application we just created. In particular, the generated Flash `treemapWidget.swf` file will be added to the HTML content by the `iWidget`.

Note: The steps in the section done in Adobe Flash Builder could have been instead performed with a text editor and the Adobe Flex SDK command line compiler. Refer to Adobe Flex documentation and IBM ILOG Elixir documentation to learn how to use command line compilation.

4.1.4 Bringing the Flex application into the `iWidget`

In order to include the Flex application in the `iWidget`, we need to go back to the Rational Application Developer project and perform the following steps:

1. Right-click the `WebContent` folder of the project and click **Import** → **File System**. The **File System** option might not be present; in this case, select the **Import** option and type `File System` in the **Filter** box.
2. A wizard similar to Figure 4-6 on page 148 displays. Browse to your Flex project to import its `bin-debug` directory. Make sure to click the **Create links in workspace** option so that when a modification is done on the Flex project, it is reflected in your `iWidget` project.

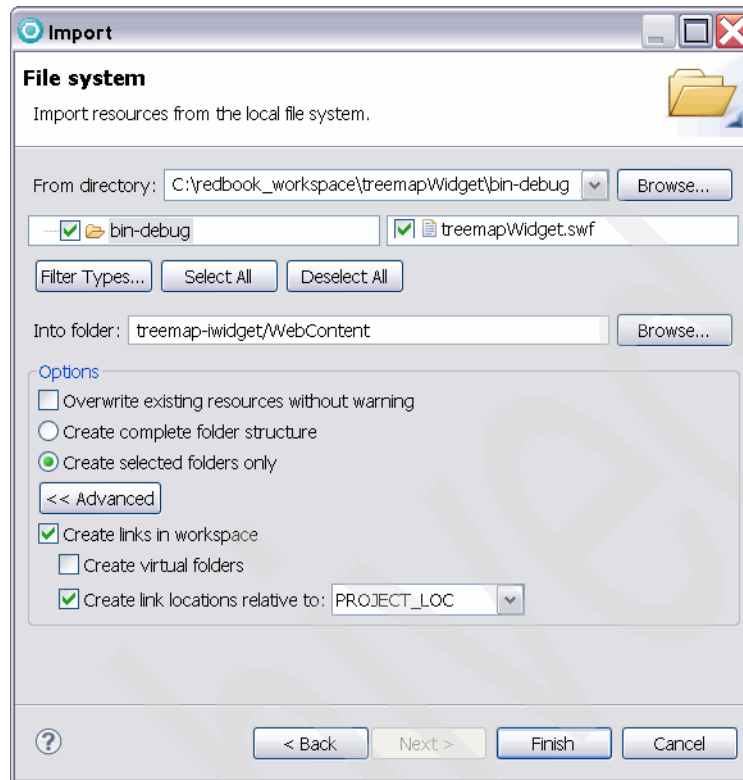


Figure 4-6 Import File system wizard

3. Open the **treemapWidget.js** file. Complete the code to load the SWF file in the bin-debug folder when the iWidget is loaded by implementing the onView method as in Example 4-5.

Example 4-5 onView method implementation

```
onView:function() {
    // replaces _IWID_ in the xml file with the widget's id to find
    // the node
    var widgetId = this.iContext.widgetId;
    var holderNodeId = "_" + widgetId + "_" + this.widgetName +
    "ViewHolder";
    var holderNode = this.iContext.getElementById(holderNodeId);

    // create a div to hold the Flash content
    var flashContent = document.createElement("div");
    dojo.style(flashContent, { "height" : "100%" });
    this.flexId = this.iContext.widgetId+"_Flex"+this.widgetName;
```

```

// get the protocol
var protocolForCodebase = "http";
if (this.iContext.io.widgetBaseUri.indexOf("https") >-1) {
    protocolForCodebase = "https";
}
var flashCodebase = protocolForCodebase+

"://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#ver
sion=10,0,0,0";
// url points to the Flash swf file;
var url =
this.iContext.io.widgetBaseUri+"bin-debug/"+this.widgetName+".swf";
// we need to rewrite url in case of proxy usage
url = this.iContext.io.rewriteURI(url);
// place the Flash content in the div
holderNode.appendChild(flashContent);
flashContent.innerHTML = "<object id=\""+this.flexId
+ "\" classid=\"clsid:D27CDB6E-AE6D-11cf-96B8-444553540000\"
+ \"codebase=\""+ flashCodebase + "\"
+ \"height=\"100%\" width=\"100%\">\"
+ "<param name=\"wmode\" value=\"opaque\">\"
+ "<param name=\"src\" value=\""+url+"\"/>\"
+ "<param name=\"flashVars\" value=\"\"widgetId=\"+widgetId+\"\"/>\"
+ "<embed id=\""+this.flexId+\" name=\""+this.flexId
+ "\" src=\""+url
+ "\"
pluginspage=\"http://www.macromedia.com/shockwave/download/index.cgi
?P1_Prod_Version=ShockwaveFlash\"
+ \"height=\"100%\" width=\"100%\" flashVars=\"\"widgetId=\"+widgetId
+ \"\" wmode=\"opaque\"/></object>";
}

```

The `onView` method is called when the widget is displayed. The code uses the Flash plug-in to load the `treemapWidget.swf` available in the `bin-debug` directory and displays it as part of the HTML content.

4. In addition to loading the Flex application, we must also register the `iWidget` with the mediator available in the `elixir.js` file so that communication between the Flex application and the widget can occur. This can be done by implementing the `onLoad` and `onUnload` methods to register and un-register the `iWidget` in the mediator as shown in Example 4-6.

Example 4-6 onLoad and onUnload methods implementation

```

onLoad:function() {
    // when loading register ourselves

```

```

        ibm_ilogvisu_redbook.addWidget(this.iContext.widgetId, this);
    },

    onUnload:function() {
        // on unload un-register ourselves
        ibm_ilogvisu_redbook.removeWidget(this.iContext.widgetId);
    }
}

```

5. Finally, you can resize the div element that contains the Flash application when resizing the iWidget. For that, you can implement the iWidget onSizeChanged method as in Example 4-7.

Example 4-7 onSizeChanged method implementation

```

onSizeChanged:function(event) {
    var widgetId = this.iContext.widgetId;
    var holderNodeId = "_" + widgetId + "_" + this.widgetName +
    "ViewHolder";
    var holderNode = this.iContext.getElementById(holderNodeId);
    var newWidth = event.payload.newWidth;
    var newHeight = event.payload.newHeight;
    dojo.style(holderNode, {
        "height" : (newHeight-3)+"px"
    });
}

```

We now have a workable iWidget that loads a Flex application and allows communication between the two. We now need to make sure that data can be plugged into the iWidget.

4.1.5 Sending iWidget data to the Flex application

In the XML descriptor of the iWidget, we have defined a Receive TreeMap Value event. This event allows the treemap to get its data value from another iWidget in a mashup application using standard wiring between the two widgets.

The contract of this event is that the treemap expects to receive JavaScript Object Notation (JSON) data in the format use in Example 4-8, where each item of the array represents a data item that will be displayed in a treemap cell.

Example 4-8 Example of JSON expected as data input

```

[
    {'revenue':1000,'name':'N1','profit':100,'country':'France'},
    {'revenue':2000,'name':'N2','profit':20,'country':'France'},

```

```

{'revenue':3500,'name':'N3','profit':130,'country':'USA'},
{'revenue':4500,'name':'N4','profit':170,'country':'USA'},
{'revenue':1500,'name':'N5','profit':150,'country':'UK'},
{'revenue':2100,'name':'N6','profit':50,'country':'UK'},
{'revenue':1800,'name':'N7','profit':180,'country':'Germany'},
{'revenue':2700,'name':'N8','profit':200,'country':'Germany'}
]

```

When such event is received, the iWidget container makes sure that the `onDataReceive` method of the iWidget is called with an event containing the data in Example 4-8 on page 150 in its `payload` property. The iWidget wrapper can then send that data to the Flex application.

A first implementation of `onDataReceive` is shown in Example 4-9.

Example 4-9 `onDataReceive` method implementation

```

onDataReceive:function(/*com.ibm.mashups.iwidget.IEvent*/ event) {
    this.data = event.payload;
    // if we receive some JSON here, to avoid using JSON
    // decode API in Flex decode it here
    if (event.type == "json" || event.type == null)
        this.data = dojo.fromJson(this.data);
    var payload = { };
    payload.value = this.data;
    // we build a object that contains everything to
    // configure the treemap
    payload.areaField = "revenue";
    payload.labelField = "name";
    payload.colorField = "profit";
    payload.groupingField = "country";
    // forward data to Flex
    if (dojo.isIE) {
        window[this.flexId].onDataReceive(payload);
    } else {
        document[this.flexId].onDataReceive(payload);
    }
}

```

Note: Be cautious because the way to call the Flex `onDataReceive` method is slightly different on Internet Explorer than other browsers.

For this first version, the data actually comes from the event, but the other configuration properties such as `areaField` and `colorField` are hard-coded. That

means that data that is passed to the iWidget *must* provide the data fields listed here: revenue, name, profit, and country. Of course, in a real mashup, the data might be different and thus contain different fields. For this reason, options will be created for that in a later section when exploring the iWidget edit mode.

4.1.6 Testing the iWidget in Rational Application Developer

Now that the widget correctly loads the treemap and sends data to it, it should be tested. Starting IBM Mashup Center server and deploying the iWidget takes some time. However, Rational Application Developer provides a J2EE Preview server that allows us to quickly test an iWidget in the IDE. For this, you need to follow these steps:

1. Click **File** → **New** → **Other...**, click **Server** and then click **Next**.
2. The wizard in Figure 4-7 displays. Choose the J2EE Preview server in the Basic section.

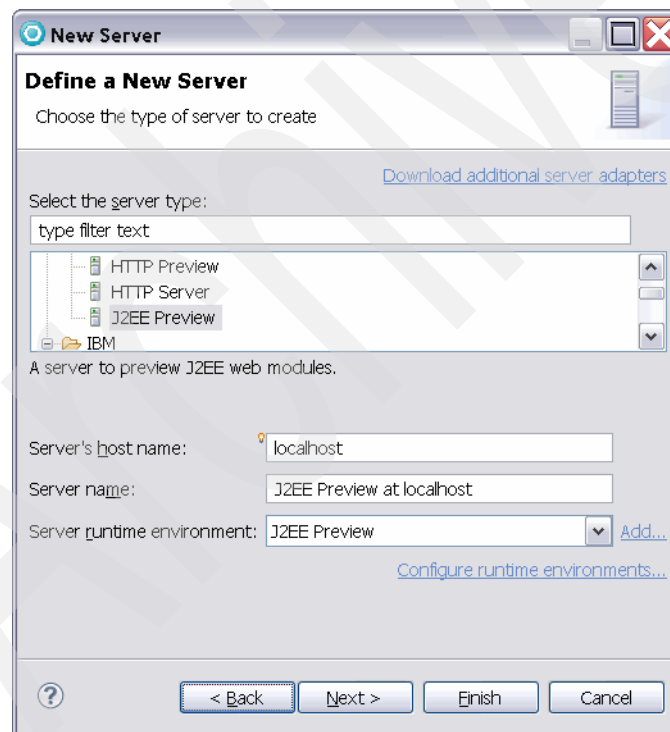


Figure 4-7 New Server wizard

3. Click **Next**, add your project to the list of the configured projects for this server, and click **Finish**.

4. Right-click the `treemapWidget.xml` file and select **Run As** → **Run on Server**. The iWidget in the Universal Test Client for iWidget runs as shown in Figure 4-8.

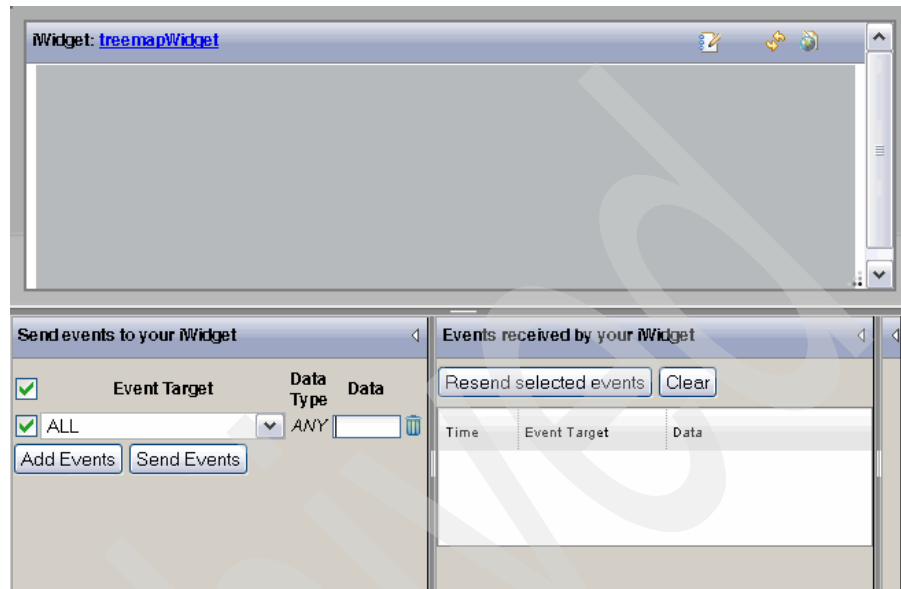


Figure 4-8 *treemapWidget* in Universal Test Client for iWidget

Because we have not provided any data, the treemap is empty (greyed).

5. To test, send the iWidget an event that contains some test data:
 - a. Select **Receive TreeMap Value** in the Event Target drop down list to select this event to be sent.
 - b. Paste the data from Example 4-8 on page 150 into the Data box.
 - c. Click the **Send Events** button. The data is sent to the treemap iWidget, which forwards it to the Flex application, which displays it as in Figure 4-9 on page 154.

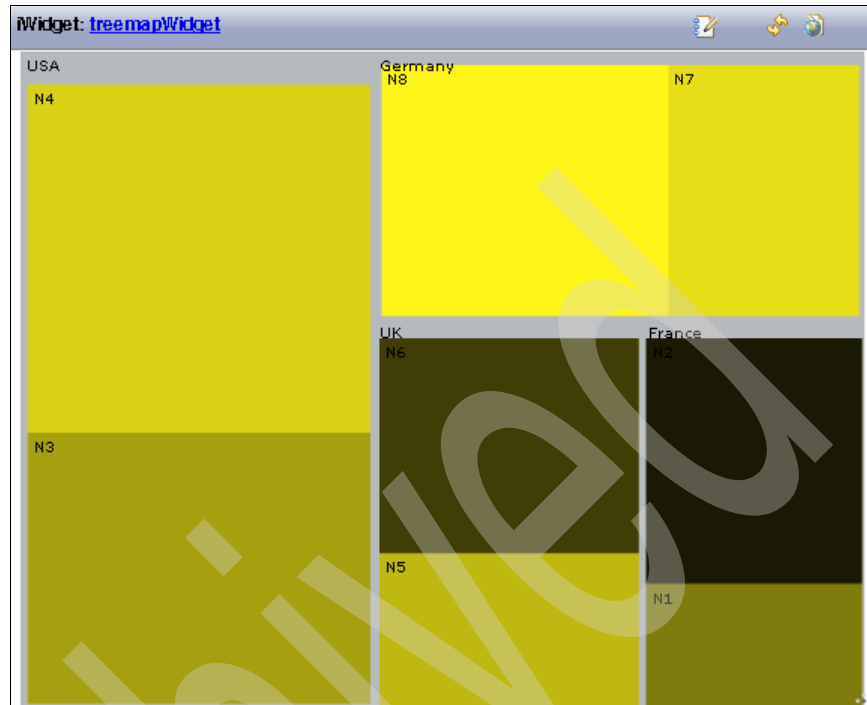


Figure 4-9 Test of the Treemap iWidget

You can see that the USA is the first country in terms of companies revenues, then Germany, the United Kingdom, and France. The company with the highest profit (lighter yellow) is the N8 company in Germany.

4.1.7 Adding more events to the iWidget

In addition to the data received event, the iWidget can expose any event that exists on the IBM ILOG Elixir Treemap component. The most interesting one is the selection event, which allows us to synchronize the selection between the treemap and, for example, an iWidget table displaying the same data.

In order to achieve this, we need to perform the following steps:

1. In Rational Application Developer, add the additional events to the XML descriptor of the iWidget. For the selection event, we do not just want to receive the selection, we want it to be bidirectional: We want to be receiving and sending the event. This can be done as in Example 4-10 on page 155 by defining two events: one with `handled="true"` (the one to be received) and the other one with `published="true"` (the one to be sent).

Example 4-10 iWidget XML descriptor excerpt

```
<iw:eventDescription id="handleSelectionDesc"
  payloadType="json" description="Receives and displays selection
value in JSON format"
  lang="en" />

<iw:eventDescription id="fireSelectionDesc"
  payloadType="json" description="Send selection value in JSON
format"
  lang="en" />

<iw:event id="Receive Selection Value" handled="true"
  onEvent="onSelectionReceive" description="handleSelectionDesc" />

<iw:event id="Send Selection Value" published="true"
  description="fireSelectionDesc" />
```

2. Add the event handler defined in Example 4-10 in the `treemapWidget.js` to receive the selection and define the method that will be called by the Flex side to notify that the treemap has changed its selection. See Example 4-11.

Example 4-11 Selection methods implementation

```
dispatchSelection : function(payload) {
  this.iContext.iEvents.fireEvent("Send Selection Value",
    null, payload);
},

onSelectionReceive : function(event) {
  // we receive some JSON here, to avoid using JSON decode
  // API in Flex decode it here
  var payload = dojo.fromJson(event.payload);
  // forward data to Flex
  if (dojo.isIE) {
    window[this.flexId].onSelectionReceive(payload);
  } else {
    document[this.flexId].onSelectionReceive(payload);
  }
}
```

3. Provide a similar selection method on the Flex side by editing `treemapWidget.mxml` in Adobe Flash Builder as in Example 4-12.

Example 4-12 treemapWidget.mxml additional methods to selection

```
<fx:Script>
```

```

<![CDATA[
  // [...]

  /**
   * Initialize the link between Flex & the browser
   * iWidget context for the handleData method to be called
   */
  private function preinitializeHandler(event:FlexEvent):void
  {
    ExternalInterface.addCallback("onDataReceive",
      onDataReceive);
    ExternalInterface.addCallback("onSelectionReceive",
      onSelectionReceive);
  }

  // [...]

  /**
   * Method called when selection to be displayed changed
   */
  private function onSelectionReceive(data:*):void
  {
    var index:int = parseInt(data.index);
    if (index !=- 1)
      treemap.selectedItems = [ _data[index] ];
    else
      treemap.selectedItems = [ ];
  }

  /**
   * Method called when selection is changed on treemap and must
   * be send back to iWidget.
   */
  private function treemap_changeHandler(event:Event):void
  {
    // selection has changed, let's forward it
    ExternalInterface.call("ibm_ilogvisu_redbook.invokeWidget",
      FlexGlobals.topLevelApplication.parameters.widgetId,
      "dispatchSelection",
      { "index" : treemap.selectedItems.length != 0 ?
        _data.indexOf(treemap.selectedItems[0]) : -1 });
  }
]]>
</fx:Script>
<ilog:TreeMap id="treemap" width="100%" height="100%"

```

```
labelThreshold="2"  
change="treemap_changeHandler(event)"/>
```

In the `preinitializeHandler` method, we register the fact that the `Flex.onSelectionReceive` method must be called when the `iWidget` is calling the method with the same name. That method is implemented such that the selected item is recovered from its index and then set on the treemap. If selection index is -1, the selection is cleared.

Conversely, when the user clicks on the treemap to select a cell, the `treemap_changeHandler` method is called and invokes the `dispatchSelection` method on the `iWidget` to let the `iWidget` send a selection event that can be caught by other `iWidgets` in the mashup.

4. To test, relaunch the `iWidget` in the Universal Test Client from Rational Application Developer. Reconfigure the data as in step 5 on page 153. Then, in the Event Target drop down, click the **Receive Selection Value** event and type the index of the data to be selected in the **Data** box. As the payload of the selection event is JSON, we need to type the index as a piece of JSON. for example, for index 3, we type { 'index' : 3}.
5. Click **Send Events**. The event will be dispatched to the treemap `iWidget`, which sends it to the Flex side, which makes sure the third item in the treemap is selected. You should now see a blueish rectangle around the N4 cell in the treemap (the one with index 3).
6. Conversely, if you click one of the treemap cells to select it, you should see in the Event sent by your widget window a new event appearing as in Figure 4-10.



The screenshot shows a window titled "Events sent by your iWidget" with a "Clear" button. Below the button is a table with two columns: "Time" and "Data". The table contains two rows of event data.

Time	Data
16:4:48	[object Object]
16:4:59	[object Object]

Figure 4-10 Log of events sent by the treemap `iWidget`

This proves the selection correctly fired an event.

4.1.8 Creating an edit mode for the iWidget

In Example 4-9 on page 151, the data fields used for color, area, label, and grouping parameters were hard-coded. Of course, depending on the data input, the user of the iWidget might want to be able to change these values. To allow for that, we need to implement an edit mode for the widget. In the edit mode, a window displays that will ask the user to specify the fields to use for each of the widget parameters.

This section describes the steps needed to add such an edit mode to the treemap widget:

1. The iWidget XML descriptor must be amended to contain the template for the new mode as in Example 4-13. This template is made of two sections: the first one contains four drop downs to select the data field for each parameter, and the second one contains a **Save** and a **Cancel** button. When the user clicks **Save** or **Cancel**, a method on the iWidget is called to decide whether or not to take the modifications into account.

Example 4-13 iWidget XML descriptor edit mode template

```
<div id="_IWID_treemapWidgetEditHolder">
  <div id="_IWID_contentNode"
    style="padding:10px;width:450px;overflow:auto;">
    <div>
      <span style="font-size:1.2em">Color field:</span>
      <select style="width:150px;width: 200px;"
        id="_IWID_colorField">
      </select>
    </div>
    <br/>
    <div>
      <span style="width:150px;font-size:1.2em">Area field:</span>
      <select style="width: 200px;"
        id="_IWID_areaField">
      </select>
    </div>
    <br/>
    <div>
      <span style="width:150px;font-size:1.2em">Label field:</span>
      <select style="width: 200px;"
        id="_IWID_labelField">
      </select>
    </div>
    <br/>
  </div>
```

```

        <span style="width:150px;font-size:1.2em">Grouping
field:</span>
        <select style="width: 200px;"
            id="_IWID_groupingField">
        </select>
    </div>
</div>
<div id="_IWID_footer">
    <input id="_IWID_save_text"
        type="button" value="Save"
        onclick="iContext.iScope().save()"
        style="margin-right: 13px;"/>
    <a id="_IWID_Cancel_Link"
        onclick="iContext.iScope().cancel();"
        onkeypress="iContext.iScope().onCancelKeyPressed(event);"
        href="javascript:void(0);">Cancel</a>
</div>
</div>

```

2. Four items must be defined in the XML descriptor to hold the values chosen in the edit mode for them to be used during the iWidget life cycle. This is done by creating an item set and adding the items to the set as shown in Example 4-14.

Example 4-14 iWidget XML descriptor items set

```

<iw:itemSet id="attributes">
    <iw:item id="colorField" />
    <iw:item id="areaField" />
    <iw:item id="labelField" />
    <iw:item id="groupingField" />
</iw:itemSet>

```

3. In order to propose the various possible values for the parameters in the drop downs created in Example 4-13 on page 158, we need to query the data input and look for data fields available in the values. For that, as shown in Example 4-15, we query all the fields of the first data item and fill the fields array with the result. This simple heuristic works fine if all data fields can be found on the first data item. However, this method will not work for more complex use-cases.

Example 4-15 Get the available fields in the data

```

this.fields = [];
for (var k in this.data[0]) {
    this.fields[i++] = k;
}

```

```
}
```

4. Add the `onEdit` method that will be called when switching to edit mode to the `treemapWidget.js` file. This method uses the field values computed in Example 4-15 on page 159 to fill the drop down list. It also initializes the `selectedIndex` property of the various drop downs with the values already available in the attributes, as shown in Example 4-16.

Example 4-16 onEdit method implementation

```
onEdit:function() {
    var widgetId = this.iContext.widgetId;

    var att = this.iContext.getiWidgetAttributes();

    var colorField = att.getItemValue("colorField");
    var areaField = att.getItemValue("areaField");
    var labelField = att.getItemValue("labelField");
    var groupingField = att.getItemValue("groupingField");

    var colorNodeId = "_" + widgetId + "_colorField";
    var areaNodeId = "_" + widgetId + "_areaField";
    var labelNodeId = "_" + widgetId + "_labelField";
    var groupingNodeId = "_" + widgetId + "_groupingField";

    var colorNode = this.iContext.getElementById(colorNodeId);
    var areaNode = this.iContext.getElementById(areaNodeId);
    var labelNode = this.iContext.getElementById(labelNodeId);
    var groupingNode = this.iContext.getElementById(groupingNodeId);

    var innerHTML = "";
    for (var i = 0; i < this.fields.length; i++) {
        innerHTML += "<option value='"+this.fields[i]+"'>"+
            this.fields[i]+"</option>";
    }

    colorNode.innerHTML = innerHTML;
    areaNode.innerHTML = innerHTML;
    labelNode.innerHTML = innerHTML;
    groupingNode.innerHTML = innerHTML;

    colorNode.selectedIndex = dojo.indexOf(this.fields, colorField);
    areaNode.selectedIndex = dojo.indexOf(this.fields, areaField);
    labelNode.selectedIndex = dojo.indexOf(this.fields, labelField);
    groupingNode.selectedIndex = dojo.indexOf(this.fields,
```

```
        groupingField);  
    }  
}
```

5. After choosing the values in the drop downs configured in Example 4-16 on page 160, the user will click either **Save** or **Cancel**. An implementation must be provided for these methods. In Example 4-17, the save method saves the values chosen by the user to the iWidget attributes before switching back to view mode, while the cancel method just switches back, ignoring the modifications of the user.

Example 4-17 cancel and save methods implementation

```
cancel:function() {  
    this.iContext.iEvents.fireEvent("onModeChanged", null,  
        {newMode : "view"});  
},  
  
save:function() {  
    var widgetId = this.iContext.widgetId;  
  
    var colorNodeId = "_" + widgetId + "_colorField";  
    var areaNodeId = "_" + widgetId + "_areaField";  
    var labelNodeId = "_" + widgetId + "_labelField";  
    var groupingNodeId = "_" + widgetId + "_groupingField";  
  
    var colorNode = this.iContext.getElementById(colorNodeId);  
    var areaNode = this.iContext.getElementById(areaNodeId);  
    var labelNode = this.iContext.getElementById(labelNodeId);  
    var groupingNode = this.iContext.getElementById(groupingNodeId);  
  
    var att = this.iContext.getiWidgetAttributes();  
  
    att.setItemValue("colorField",  
        this.fields[colorNode.selectedIndex]);  
    att.setItemValue("areaField",  
        this.fields[areaNode.selectedIndex]);  
    att.setItemValue("labelField",  
        this.fields[labelNode.selectedIndex]);  
    att.setItemValue("groupingField",  
        this.fields[groupingNode.selectedIndex]);  
  
    att.save();  
  
    this.iContext.iEvents.fireEvent("onModeChanged", null,  
        {newMode : "view"});  
}
```

```
}
```

6. Now that the configuration is saved, in the attributes, we must use these values when sending the data to the Flex side instead of using the hard-coded values. In Example 4-18, the `onDataReceive` method now calls a `sendData` method and in this method, the `colorField`, `areaField`, `labelField` and `groupingField` values are queried from the attributes to send them to the Flex application.

Example 4-18 sendData implementation

```
onDataReceive:function(event) {
    this.data = event.payload;
    // if we receive some JSON here, to avoid using JSON
    // decode API in Flex decode it here
    if (event.type == "json")
        this.data = dojo.fromJson(this.data);
    // we need this to know available fields
    var i = 0;
    this.fields = [];
    for (var k in this.data[0]) {
        this.fields[i++] = k;
    }
    this.sendData();
},

sendData:function() {
    if (this.data != null) {
        var payload = { };
        payload.value = this.data;
        // we build a object that contains everything to
        // configure the treemap
        var att = this.iContext.getWidgetAttributes();
        payload.colorField = att.getItemValue("colorField");
        payload.areaField = att.getItemValue("areaField");
        payload.labelField = att.getItemValue("labelField");
        payload.groupingField = att.getItemValue("groupingField");
        // forward data to Flex
        if (dojo.isIE) {
            window[this.flexId].onDataReceive(payload);
        } else {
            document[this.flexId].onDataReceive(payload);
        }
    }
}
```

7. When switching modes, the Flex application is reloaded by the web browser, so we have to actually re-send the data and selection events to the Flex application as soon as we re-enter the view mode. This could be done by calling the appropriate methods on the document or window object as in Example 4-9 on page 151. However, this is not possible because there is a delay between the instant the mode is switched and the moment the Flex application is actually reloaded. That means we need to wait for the Flex application to be available in order for the data and selection to be sent back. To coordinate these two events, perform the following steps:
 - a. As in Example 4-19, create a method on the iWidget that will be called when the Flex application will be ready and that will be in charge of sending back the data.

Example 4-19 commitPendingEvents implementation

```
commitPendingEvents:function() {  
    this.sendData();  
    this.sendSelection();  
}
```

- b. On the Flex side, call this method when the application has been fully loaded, as in Example 4-20.

Example 4-20 treemapWidget.mxml additional code

```
<fx:Script>  
    <![CDATA[  
        // [...]  
  
        protected function  
        treemap_creationCompleteHandler(event:FlexEvent):void  
        {  
            ExternalInterface.call("ibm_ilogvisu_redbook.invokeWidget",  
                FlexGlobals.topLevelApplication.parameters.widgetId,  
                "commitPendingEvents", null);  
        }  
  
    ]]>  
</fx:Script>  
<ilog:TreeMap id="treemap" width="100%" height="100%"  
    labelThreshold="2"  
    creationComplete="treemap_creationCompleteHandler(event)"  
    change="treemap_changeHandler(event)"/>
```

8. The iWidget is ready for testing in the Universal Test Client. Set the test data and then switch to edit mode by clicking **Modes** → **edit** (the icon at the top right corner of the widget). The Edit mode shows up as shown in Figure 4-11.

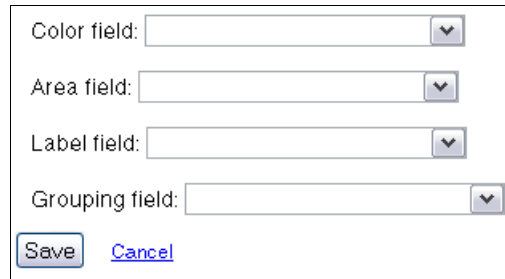


Figure 4-11 iWidget Edit Mode in Universal Test Client

You can change the various fields values and click **Save**. The treemap will then be rebuilt to comply with your new configuration. If you come back to the edit mode, you will see the values have been preserved thanks to the iWidget item set mechanism. Conversely, if you modify them and click **Cancel**, no effect will be taken and coming back to the edit mode should revert back to the previous settings.

4.2 Deploying the IBM ILOG Elixir iWidget in IBM Mashup Center

We now have a functional treemap iWidget. This section describes how to package that iWidget and deploy it to IBM Mashup Center catalog so it can be used in a mashup application.

IBM Mashup Center is an enterprise mashup platform: it allows us to dynamically assemble mashup applications in a secure and reliable manner. It provides a set of components including:

- ▶ The catalog where you can manage widgets, feeds, and mashups.
- ▶ The mashup builder that allows you to drag and drop widgets, and create a mashup application by connecting them to data feeds and wiring the widgets to exchange events between them.
- ▶ The feed generator that allows you to create a data feed from several types of data sources including databases, XML, and Microsoft Excel documents.

- ▶ The data mashup builder that allows you to combine and transform several feeds into a new one. This is particularly useful if you have several sources of data you need to aggregate.

4.2.1 Software requirements

This section requires you to have the following products or their trial version installed:

- ▶ IBM Rational Application Developer 8.0
- ▶ IBM Mashup Center 2.0

To complete the steps in this section, you will need to complete the steps documented in 4.1, “Creating an iWidget wrapper for an IBM ILOG Elixir component” on page 136 first.

4.2.2 Deploying the iWidget to the catalog

Before being able to use the iWidget in the mashup builder, we need to deploy it to the catalog following these steps:

1. In Rational Application Developer, right-click your project name and click **Export** → **WAR**. The wizard shown on Figure 4-12 displays.

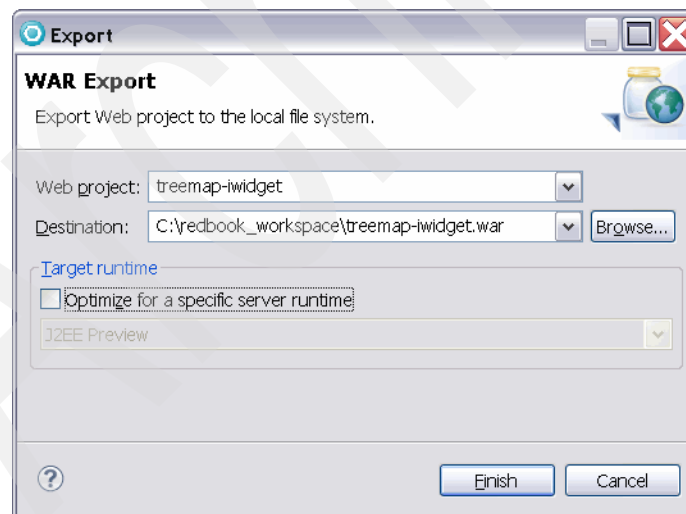


Figure 4-12 WAR Export wizard

2. You can click to clear Optimize for the J2EE preview server because the WAR will be run on the Mashup Center server. After you are done, click **Finish** and

a WAR file is exported. This WAR file contains everything needed by Mashup Center to load your iWidget.

Note: Before exporting a WAR, if the `catalog.xml` and `mashup.properties` files are in the `WebContent` of your project but not in the `WEB-INF` sub-directory, move them to that directory because Mashup Center will expect them to be there when importing the WAR file. You can also edit the `catalog.xml` to associate your iWidget with an icon and a description. This is highly recommended to facilitate the use of the iWidget in the Mashup Center environment.

3. After starting the Mashup Center server, navigate to the catalog that resides in InfoSphere™ MashupHub at a URL similar to `https://yourserver:9444/mashuphub`. After logging in, on the catalog page click **Upload** → **Upload Widget**. In the window that displays, select the iWidget source type. In our case, select **iWidget Package (.war)** and then click **Next**.
4. A new window displays that allows you to upload the WAR created in step 1 on page 165. Browse to the WAR file location and then click **Next**.
5. On the next window, you can specify some detailed information for the widget including its name and version as shown on Figure 4-13. Click **Finish** when done.



Specify the following information

Required fields marked with *

* Title:

Description:

* Version:

Tags:

Permissions:

- ☒ Public (All users can view the Widget)
- ☐ Private (Widget is invisible to all other users)
- ☐ Custom (Custom permission settings)

Figure 4-13 iWidget details form

After performing the previous steps, the widget has now been deployed to the catalog. In order for it to be used into the mashup builder, add it to a palette of the mashup builder:

1. Click the **Add to Mashup Builder** link on the page that appeared after step 5 on page 166.
2. Enter your login details in the mashup builder login window that displays and click **Next**.
3. The following window asks you to select the palette or category you want the iWidget to be added to. Select the Demo category and click **Finish**.

The treemap iWidget should now be available in mashup builder in the Demo category.

4.2.3 Prepare your data with the feed generator

Before using the treemap iWidget previously deployed to create a mashup application, we need to have some data ready in mashup builder. Either use the sample data already available in Mashup Center, or make sure your actual data are available. In this example we will use a dummy data file and make that data available as a data feed through the feed generator.

Note: To get this sample Microsoft Excel data file, see Appendix A, “Additional material” on page 287 for additional material related to chapter 4.

1. Log onto InfoSphere MashupHub as in step 3 on page 166. Click **Create** → **New Feed** to create a new feed. A window asking you which source of data you want to use shows up. For this example, select the **Excel Workbook** source and click **Next**.
2. Browse to the Microsoft Excel file and upload it. If you use the example file, remember to specify the header row as being row number 1 as in Figure 4-14 on page 168.

*Input type: ☒ Upload File ☐ Upload URL

*Filepath:
Specify the location of the file

Header row:
The row number that contains the column headings

Range:
Specify the range to include (ex: A1:F5 default: include entire worksheet)

Worksheet number:
The worksheet (tab) number from the Excel workbook to use

Figure 4-14 Microsoft Excel data file upload

3. Click **Next** and you will arrive at a window to enter the details of your feed. Type name such as My Treemap Data and click **Finish**. The data should now be available as a data feed to be used in mashup builder. If you look at the details of the feed, it gives you the URL of the feed which will be in the following form:

`https://myserver:9444/mashuphub/client/plugin/generate/entryid/61/pluginid/3.`

Copy this URL so you will have it available later.

Note: In this section, we simply directly translated a Microsoft Excel file to a data feed. However, data mashup builder is much more powerful than this. By clicking **Create** → **Data Mashup** in InfoSphere MashupHub, you are able to mashup various feeds and create new sources from them.

4.2.4 Using the iWidget in mashup builder

This section describes how to create a treemap iWidget instance in mashup builder and connect it with the data feed created in the previous section.

Creating a mashup space and page

The first task is to create a mashup page that will be the container of our application.

1. Launch the builder by going to Lotus® Mashups URL (this is something like `http://myserver:9081/mum/enabler`). After you log in, the Welcome Space of IBM Mashup Center is shown.
2. Click **Welcome Space** → **Create a New Space** to create a new space to avoid cluttering the default space. Type Redbook Space in the Space name field and click **Save** as shown in Figure 4-15.

Figure 4-15 Create Space window

3. Click **Create a New Page** to add a page to the Redbook Space and type Redbook Page for its name.

Creating a Treemap and connecting it to data

The page is now created and is by default in edit mode, which means you will be able to build your mashup by adding new widgets to the page.

1. To create a treemap widget, click the **Demo** category and drop a Treemap Widget onto the page.

2. Click the **Tools** category and drop a Data Editor onto the page to connect that widget to the data feed we defined in section 4.2.3, “Prepare your data with the feed generator” on page 167.
3. Click the **Click to wire widgets** menu of the treemap to display the wiring window. As shown in Figure 4-16, click the **Receive** tab and choose to receive as TreeMap value the content of the Data Editor.

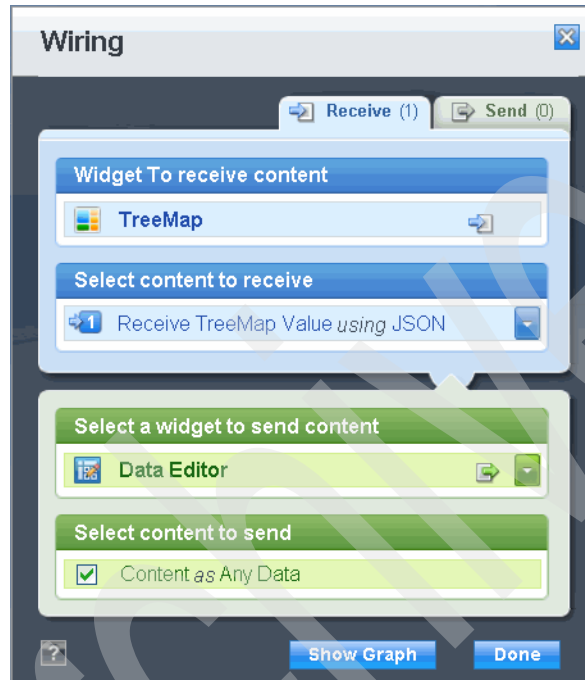


Figure 4-16 iWidget Wiring

4. Click the **Edit Settings** menu as shown in Figure 4-17 on page 171. In the Settings window, click the **Get Data From Feed** button and enter the feed URL determined in step 3 on page 168. After you have entered this, click **OK**, then **Save**. This loads the data into the editor and sends it to the treemap.

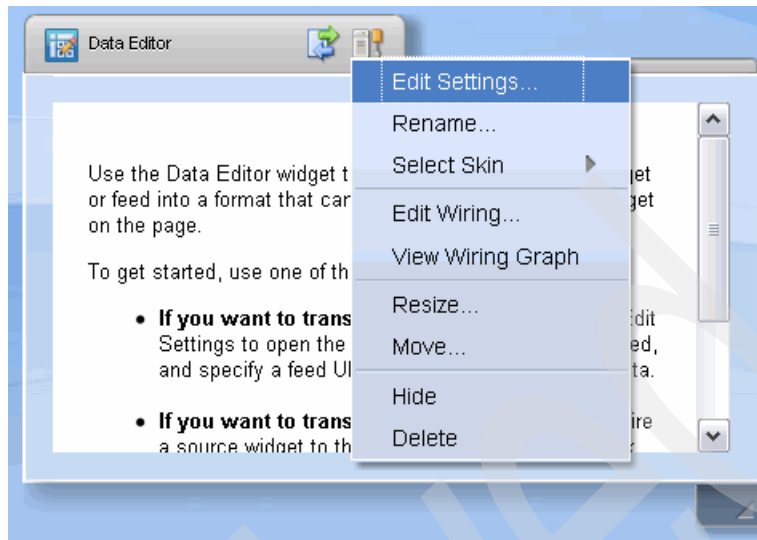


Figure 4-17 Edit Settings menu

Note: In this example, we have just loaded the data and forwarded it to the treemap. However, the Data Editor also allows you to sort or filter data before sending it to the target iWidget.

5. Click the **Edit Settings** menu. A window displays that allows you to specify the fields as when testing in the Universal Test Client in step 8 on page 164. Click **Save** and you will get the treemap showing your data as in Figure 4-18.

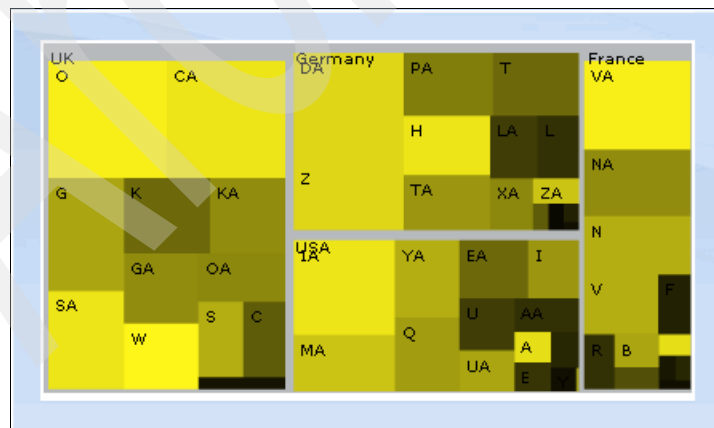


Figure 4-18 Treemap iWidget after data connection

Wiring the treemap selection with other widgets

Now that you have the correct data, display that data in a table widget and synchronize the selection with the treemap:

1. Click the **Favorites** category and drop a Data Viewer on the page.
2. Click the **Click to wire widgets** menu of the data viewer to display the wiring window. Click **Receive** and wire the data from the Data Editor as shown in Figure 4-19. This is similar to wiring the treemap to accept data from the Data Editor. The Widget to receive content should be Data Viewer, and the widget to send content should be Data Editor.

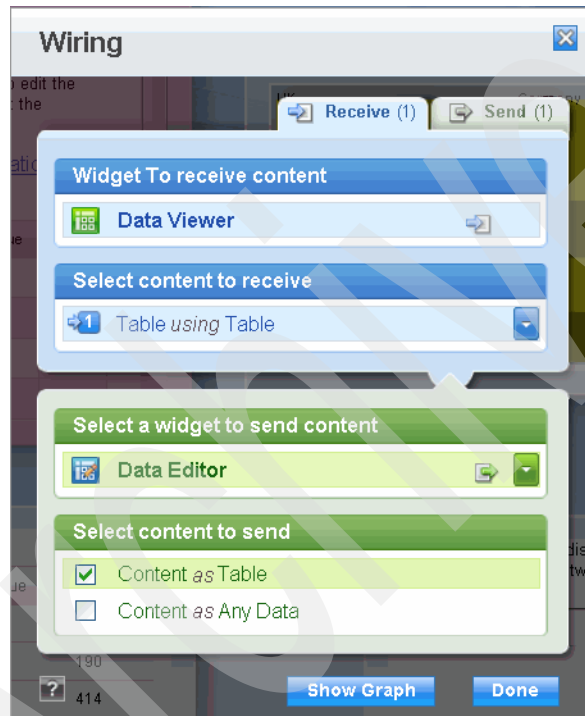


Figure 4-19 Data Viewer wiring

3. The data now shows up in the table. However, the first columns of data (the index) are shown. To remove this column, click the **Edit Settings** menu and select **Table Content**. When the list of columns is shown, click to clear the Index column and click **Save**.
4. The data now appears in the table without the index column as shown in Figure 4-20 on page 173.

Company	Country	Revenue	Profit
A	USA	8927	144
B	France	6528	190
C	UK	3645	414
D	Germany	3605	54
E	USA	2115	137

Page 1 | [2](#) | [3](#) | [4](#) | [5](#) | ...

[Previous](#) | [Next](#)

Figure 4-20 Data Viewer iWidget

5. Make sure that after a Data Viewer row is selected, the corresponding cell is selected in the TreeMap by wiring the index of the table selected row to the selection event of the treemap. However, although the index of the Data Viewer selected row is exposed as a raw index like '3', the TreeMap is expecting a piece of JSON of the form { index : 3 }. Therefore, we need to use a JavaScript Adapter between the two widgets:
 - a. Click the **Tools** category and drop a JavaScript Adapter onto the page.
 - b. Wire the JavaScript Adapter Receive event to the Data Viewer Index event.
 - c. Click the JavaScript Adapter **Edit Settings** menu and select **Table Content**. When the window is shown, type `return dojo.toJson({index : payload});` in the text field as the returned value for the adapter. This line of code takes the raw index and create a piece of JSON that can be consumed by the treemap iWidget as a selection event input. Click **Save**.
 - d. Click the **Click to wire widgets** menu of the treemap and select the content of the JavaScript Adapter as the input value for the Receive Selection Value event as shown in Figure 4-21 on page 174.

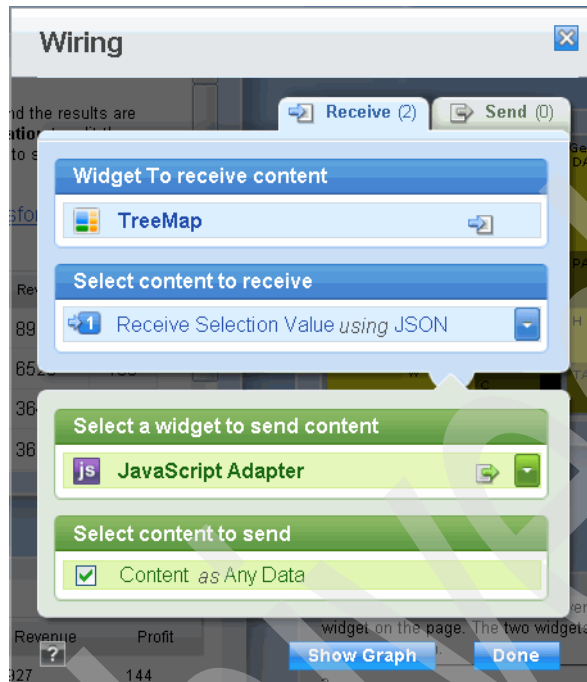


Figure 4-21 Wiring selection window

After you have clicked **Done**, you should be able to select a row in the Data Viewer and see the corresponding cell selected (blue halo rectangle) in the TreeMap.

6. Finally, hide the intermediary widgets that are just there to connect and modify data, namely the Data Editor and the JavaScript Adaptor. This gives you a final mashup application that looks like Figure 4-22 on page 175.

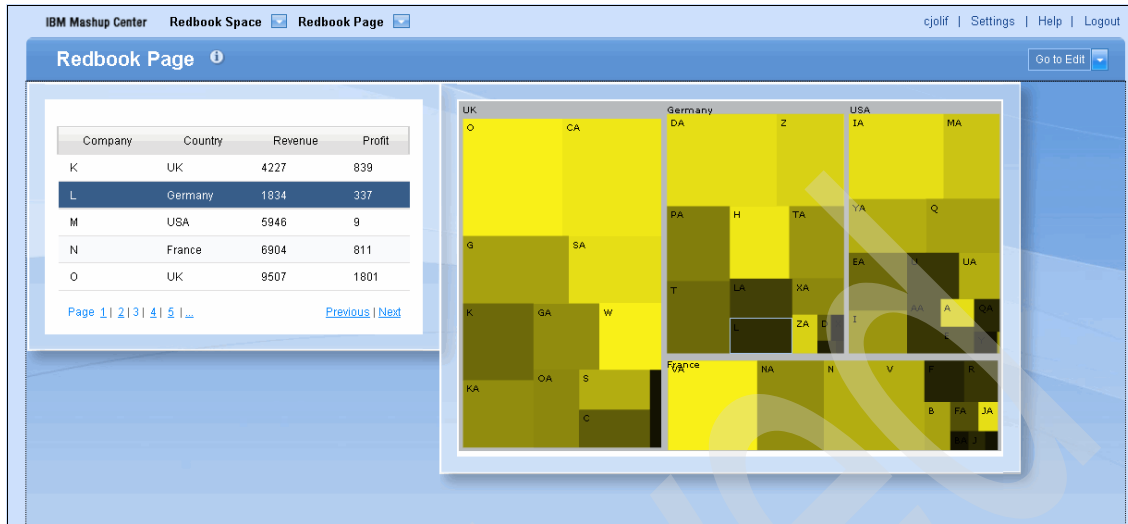


Figure 4-22 Final mashup application

Now that the mashup application is built, you can save your work and go back to the page view mode.

You can go even further and, for example, have a detailed form displaying the details of a company revenue and profit that changes its data based on the selected cell in the treemap.

Any IBM ILOG Elixir component such as pivot charts, gauges, dashboard maps, and timelines can be wrapped as an iWidget in a similar way and re-used in any mashup application. You can customize the iWidget even further by allowing the mashup builder to customize more advanced settings than the ones explored in this example, such as colors and other styles used by the component.

Business Monitoring with ILOG Elixir and WebSphere Business Space

This chapter describes how IBM ILOG Elixir Enterprise components can be used inside the Business Space powered by WebSphere environment to provide new ways to visualize and monitor your business data.

In this chapter, we will see how to take advantage of the REST services of WebSphere Business Monitor to create a dashboard that will represent the key performance indicators of our business using IBM ILOG Elixir Enterprise.

Finally, we will explore two ways to deploy this dashboard in the Business Space environment: deployment of the application in the Web Site widget of Business Space for a simple integration, and creation of a new iWidget in the Business Space catalog for a deeper integration with more customization capabilities for the user.

5.1 Introduction to IBM Business Space powered by WebSphere

IBM Business Space powered by WebSphere is a common or general user interface framework for aggregating content and delivering it using a browser. In this book, *Business Space* will be used interchangeably with IBM Business Space powered by WebSphere.

A business space is a collection of related web content that provides you with insight into your business.

You can have many business spaces present on a Business Space run time, and each space has within it a number of tabbed pages. Each space is set up to provide pages relevant to a particular role or business function. Currently, the primary use of Business Space is within the Business Process Management (BPM) suite. So, for example, you can have a business space for monitoring business processes, and a further space for working on the human tasks within those processes.

Figure 5-1 shows the welcome page of the Business Space environment.

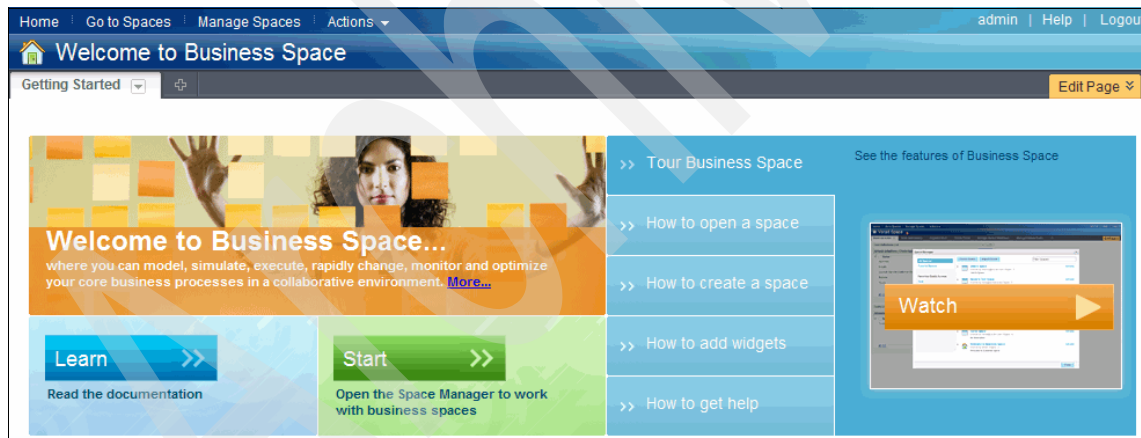


Figure 5-1 Welcome to Business Space

Business Space supplies the user interface for the Business Process Management (BPM) set of services and tools that support process management.

The activities that you perform to develop and manage your business processes are termed the BPM life cycle. This involves documenting the processes themselves, developing and testing services, and monitoring and improving the results.

A number of products can be used for developing and monitoring artifacts across the BPM life cycle, including:

- ▶ IBM WebSphere Business Compass
- ▶ IBM WebSphere Business Modeler
- ▶ IBM WebSphere Business Services Fabric
- ▶ IBM WebSphere Business Monitor
- ▶ IBM WebSphere Integration Developer
- ▶ IBM Business Space powered by WebSphere

The business spaces you create are made up of pages that contain widgets. *Widgets* are user interface components that are combined to provide the functionality of the pages in your business spaces.

To develop widgets, you need an environment for coding, running, testing, and viewing them. In this section, we will use WebSphere Integration Developer. Business Space supplies several common widgets, and other IBM WebSphere products also provide widgets.

To assemble business spaces, you need a browser linked to an application server running IBM Business Space powered by WebSphere.

5.2 Business monitoring using IBM ILOG Elixir components

In our scenario, we have chosen to show how to use IBM ILOG Elixir, IBM Business Space and WebSphere Business Monitor together to create a monitoring dashboard. However, many other integration scenarios of Business Space and ILOG Elixir are possible. Even If you are looking for another type of scenario, you might learn interesting techniques of integration of Elixir and Business Space from this scenario.

In this section, we will discuss the following sections:

- ▶ Business Monitoring
- ▶ The Integration Scenario

5.2.1 Business monitoring

Business monitoring solutions are used to give an indication of the overall health of an organization with regards to various key performance indicators and business measures. These could be the number of items delivered on time, the average turn over rate, the average time to handle a call, and other data.

In order for organizations to change quickly and to dynamically account for a new market strategy, detect potential problems before they occur, and identify opportunities, access to both real-time and historical data is of the utmost importance.

Business monitoring collects relevant events and aggregates the event data to present real-time, role-based information. This information is then shown in dashboards.

By monitoring business events, existing business processes can be improved. The continual process of measuring and analyzing business processes is crucial to optimizing the efficiency of an organization and defining potential areas for cost-cutting. This can be the case, for example, in identifying areas of the business that require large amounts of manpower but account for only a small percentage of overall revenue.

IBM has a large portfolio of products that can be used for business monitoring based on specific requirements. These products are:

- ▶ Cognos

This product delivers the complete range of Business Intelligence (BI) capabilities using a single service-oriented architecture (SOA). Users can capture and manage multiple hierarchies, structures, and definitions in a centralized store for use across performance management applications.

- ▶ WebSphere Business Monitor

This product is a comprehensive Business Activity Monitoring system that offers real-time insight into business processes. It provides users with real-time, end-to-end views of business process performance using customizable dashboards, portals, and mobile devices. It also provides users with historical analysis and prediction capabilities, and allows them to create their own KPIs and alerts.

- ▶ WebSphere Lombardi Edition

WebSphere Lombardi Edition provides simple web-based authoring for business users, allowing them to define their processes. Lombardi Teamworks provides extensive levels of caching to optimize runtime performance. Process execution is dynamic, evaluating each step and transition within the process diagram in real time.

- ▶ WebSphere Business Events

This product allows users to define and manage business events. It enables business users to detect, evaluate, and respond effectively to events or patterns of events. It provides powerful correlation capabilities and can effectively process high volumes of events generated by disparate systems.

For this chapter, we will focus on using WebSphere Business Monitor as our business activity monitoring platform. Refer to Chapter 3, “Using IBM ILOG Visualization and IBM Cognos for web applications” on page 85 for an integration of ILOG Visualization inside the Cognos product.

5.2.2 The integration scenario

For our scenario, we will use the ITSO Movies company. This is a company that distributes films to theaters in several countries. The company uses Business Space and WebSphere Business Monitor to monitor the number of tickets sold in its various locations.

The ITSO Movies company has developed a monitoring model in WebSphere Business Monitor, and has defined several KPIs (Key Performance Indicators) in the model so that those KPIs can be displayed inside Business Space powered by WebSphere.

The model developed by the ITSO Movies company allows users to monitor the number of tickets that are sold in each theater, contains KPIs to monitor the monthly sales (for example, the KPI named Monthly Sales for China), and also contains KPIs to monitor the daily sales (for example, the KPI named Daily Sales for UK).

Figure 5-2 shows the Sales KPI in Business Monitor KPIs Widget.



Figure 5-2 The KPI widget

In our scenario, we will query the KPI's value through the WebSphere Business Monitor REST services and create a Flex application with IBM ILOG Elixir. This Flex application will allow the ITSO Movies company to monitor the monthly

sales and the daily sales values in an interactive dashboard, mixing gauges, maps, and charts.

5.2.3 Software requirements

In order to be able to follow this scenario, you need to have the following software installed:

- ▶ WebSphere Business Monitor version 7.0
- ▶ Business Space powered by WebSphere version 7.0
- ▶ WebSphere Integration Developer 7.0
- ▶ IBM ILOG Elixir Enterprise 3.0
- ▶ Adobe Flash Builder 4.0, to develop the Flex application

Note: We will not discuss the actual monitor model development, as it is outside the scope of this book. We assume that the monitor model was previously developed and we will only discuss the related Business Space aspects and tasks.

The monitor models are available for download from the additional material provided with this book. You need to deploy the monitor models to your WebSphere Business Monitor instance before you can follow our example.

To follow the example, you also need to generate ticket sales using the modules provided.

The following items need to be deployed to generate the ticket sales events:

- ▶ ITSOTicketSalesApp: a mediation module
- ▶ ITSOMovieTicketSalesMMApplication: the monitor model
- ▶ CreateTicketSalesEventsApp: a Ticket Sales generator

After you have deployed these items, log into the Business Process Choreographer and perform the following steps:

1. Click **Currently Valid**.
2. Click **TicketSalesGenerator**.
3. Click **Start Instance**.
4. Enter the number of tickets you want to generate and click **Submit**.

5.3 WebSphere Business Monitor REST Services

WebSphere Business Monitor offers several services to query the monitored information. The services offered as REST services are the following:

- ▶ Model services
These services return metadata about deployed monitor models.
- ▶ KPI services
These services support the retrieval and management of Key Performance Indicators.
- ▶ Instance data services
These services return monitored data that has been collected by Business Monitor.
- ▶ Diagram services
These services return Scalable Vector Graphics (SVG) diagrams for deployed monitor models.
- ▶ Alert services
These services support the retrieval and management of alert subscriptions and alerts.
- ▶ Security services
These services support security.

For our use scenario, we will mainly use the KPI services, which allow us to query the KPI information and the KPI values.

5.4 Creating the Elixir dashboard

Start by creating an Adobe Flex application that can display key performance indicators of the ITSO Movies company and deploying it in the Business Space. To display the key performance indicators, we will use several IBM ILOG Elixir components: the radar chart, the map, and gauge components. These components allow us to monitor several key performance indicators at the same time and compare the values at a glance.

Note: The source code for the Adobe Flash Builder project that is created in this section is available for download on the web. Check Appendix A, “Additional material” on page 287 for details on how to download it.

In the following sections we will perform the following steps:

1. Create the monitoring dashboard with ILOG Elixir
2. Create a new KPI from the Business Space Dashboard
3. Call the Business Monitor REST Services in Flex
4. Display the KPI in the Elixir components

5.4.1 Creating the dashboard in MXML

To build an Adobe Flex application, we will use the Flash Builder IDE from Adobe. You will need to have installed ILOG Elixir Enterprise version 3.0. Create a new Flex project with Flash Builder using the following steps:

1. Click **File** → **New** → **Flex Project** in the IDE.
2. In the New Flex Project window, type `ITSOMoviesDashboard` and then click **Next**.
3. In the Configure Output window click **Next** again.
4. In the Library path tab window, select “Merge into code” for the Framework linkage. With this Flex linking option, all the required Flex libraries will be merged in our application and it will be easier to deploy.
5. Click **Finish**

A new, empty Flex project is created.

We will now add the IBM ILOG Elixir libraries into this project. To do so, in the Package Explorer view, right-click in the newly created project and choose **Add IBM ILOG Elixir Libraries**.

At this point, a new file named `ITSOMoviesDashboard.mxml` is created. This will be our main file.

We can now replace the content of the `ITSOMoviesDashboard.mxml` with the code shown in Example 5-1.

Example 5-1 ITSOMoviesDashboard.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:Application xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark"
    xmlns:mx="library://ns.adobe.com/flex/mx"
    xmlns:ibm="http://www.ibm.com/xmlns/prod/ilog/elixir/2010"
    xmlns:ilog="http://www.ilog.com/2007/ilog/flex"
    width="100%" height="100%"
    applicationComplete="initializeDashboard()">
```

```

<fx:Script>
  <![CDATA[

      protected function initializeDashboard():void
      {

      }

  ]]>
</fx:Script>

<s:HGroup width="100%" height="100%"
  paddingBottom="8" paddingLeft="8"
  paddingRight="8" paddingTop="8" >
  <s:VGroup width="80%" height="100%" >
    <s:Panel height="100%" width="100%"
      title="ITSO Movies Daily Ticket Sales" >
      <s:VGroup height="100%" width="100%"
        paddingBottom="5" paddingLeft="5"
        paddingRight="5" paddingTop="5" >
        <ibm:SemiCircularLinearGauge
          id="dailySalesGauge"

skinClass="com.ibm.ilog.elixir.gauges.skins.spark.SemiCircularGaugeSkin
"

          editable="false"
          title="Total Daily Sales"
          height="100%" width="100%"/>
        <ilog:RadarChart id="radarChart"
          width="100%"
          height="100%"/>
        </s:VGroup>
      </s:Panel>
    </s:VGroup>
    <s:VGroup width="100%" height="100%">
      <s:Panel title="Monthly Sales Map Dashboard"
        width="100%" height="100%">
        <ibm:WorldCountriesMap id="map"
          width="100%" height="100%"/>
        </s:Panel>
      <s:Panel title="Monthly Sales Country Information"
        width="100%" height="100%">
        <s:HGroup paddingBottom="5" paddingLeft="5"
          paddingRight="5" paddingTop="5"

```

```

        width="100%" height="100%">
        <ibm:SemiCircularLinearGauge id="monthlySalesGauge"
            width="100%" height="100%"
            editable="false"/>

    </s:HGroup>
</s:Panel>
</s:VGroup>
</s:HGroup>
</s:Application>

```

The MXML code is a mix of Flex components and IBM ILOG Elixir components. The Elixir components are the following:

- ▶ The `<ilog:RadarChart>` tag represents the IBM ILOG Elixir Radar Chart component.
- ▶ The `<ibm:SemiCircularLinearGauge>` tag represents an Elixir linear gauge.
- ▶ The `<ibm:WorldCountriesMap>` tag represents an Elixir map of the world on which we will place KPIs.

Figure 5-3 shows the outline of the dashboard in the Flash Builder Outline window.

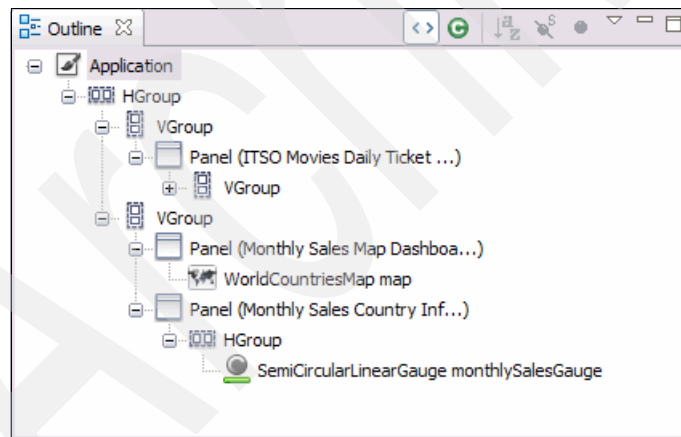


Figure 5-3 Outline of components in the dashboard

When the mxml file is created, click the **Design** button in the ITSOMoviesDashboard.mxml buffer in Flash Builder. You should get a similar result as in Figure 5-4 on page 187.

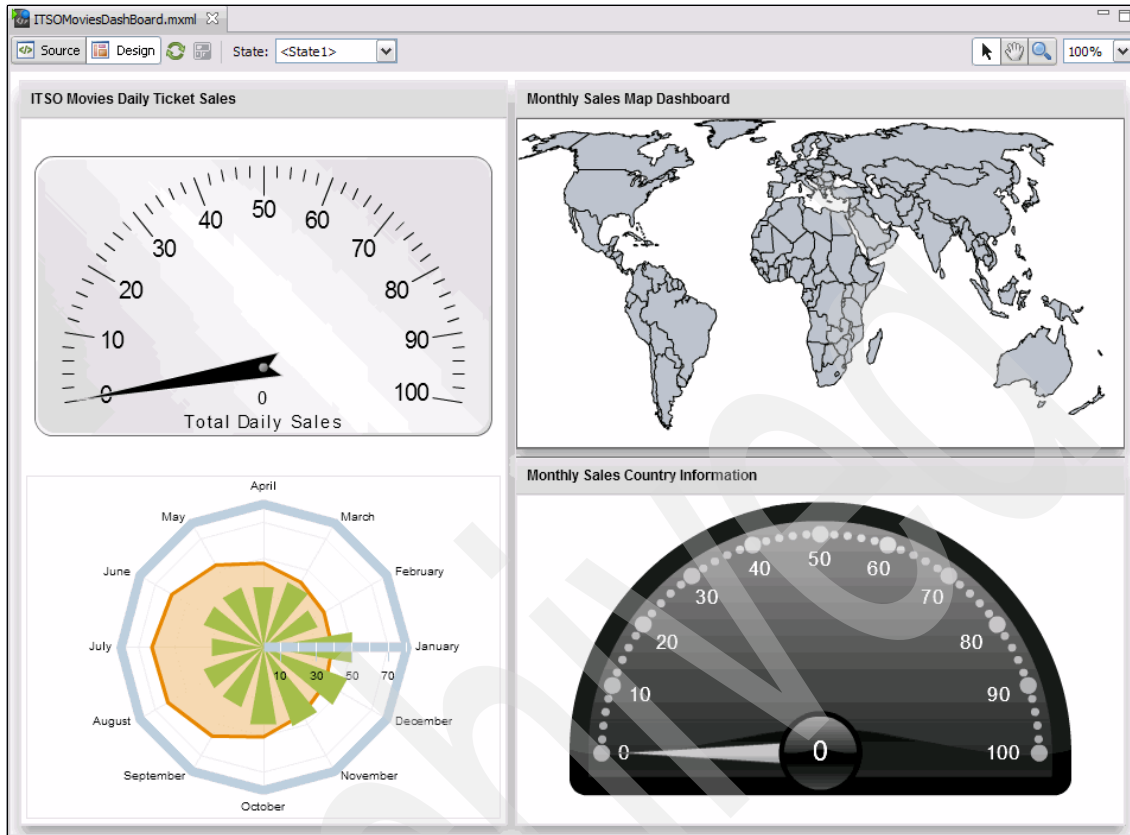


Figure 5-4 The dashboard in design view

Note: In the current example we only use standard gauges from IBM ILOG Elixir. The product provides an Eclipse plugin that enables the creation of custom gauges. You can read about this at the following URL:

http://publib.boulder.ibm.com/infocenter/elixent/v3r0/index.jsp?topic=/com.ibm.ilog.elixir.doc/Content/Visualization/Documentation/Flex/Elixir_Enterprise/_pubskel/ps_elixir_corecomponents101.html

5.4.2 Creating a new KPI in WebSphere Business Monitor

In our scenario, we want to display the total daily sales of tickets in an Elixir gauge. The ITSOMovies Business Monitor model does not provide such a KPI; the model only defines a KPI for the daily sales in each country. One way to solve this problem would be to add the KPI in the monitoring model or to do the

computation of the total daily sales through code in the Flex dashboard, but it is also possible to create a new KPI directly from the Business Space dashboard. WebSphere Business Monitor allows creating new KPIs directly from the Business Space environment. For this, you need to have a Business Space that displays the KPI Manager Widget.

1. In the KPI Manager widget, in the model field, select `ITSOMovieTicketSalesMM`. The KPIs for our monitoring model appear as shown in Figure 5-5.

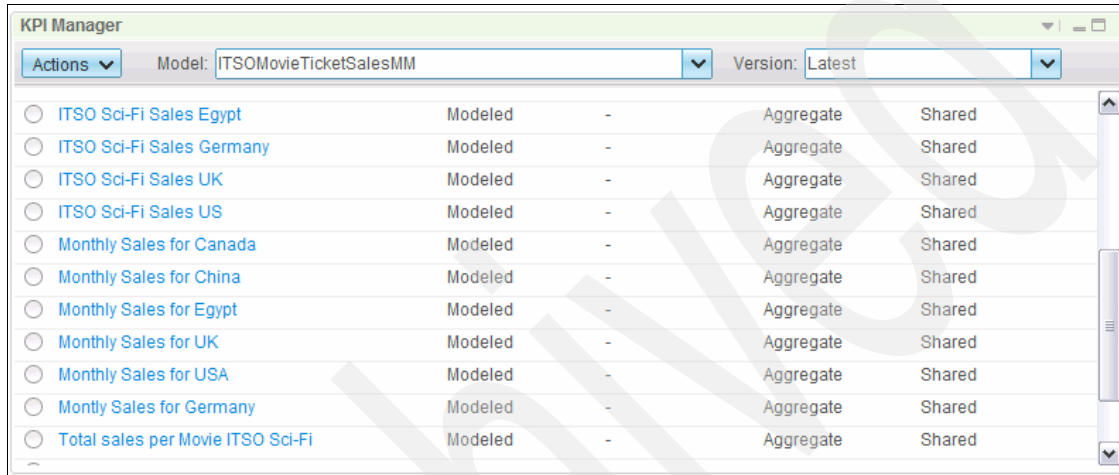


Figure 5-5 KPI Manager widget showing the `ITSOMovieTicketSalesMM`

2. Click **Actions** → **New Expression KPI**.
3. In the New Expression KPI Properties window, type `TotalDailySales` in the KPI name field.
4. Select **Shared** in the Access field.
5. Click the **Definition** tab
6. In the expression field, type:
`Daily_Sales_for_Canada+Daily_Sales_for_China+Daily_Sales_for_Egypt+Daily_Sales_for_Germany+Daily_Sales_for_UK+Daily_sales_for_USA`
7. Click the **Range** tab.
8. Choose 20000000 for the Target.

We have created a new KPI named `TotalDailySales` that corresponds to the total daily sales of the ITSOMovieTicketSales company. This new KPI is accessible from the WebSphere Business Monitor REST services like the other KPIs that have been defined initially in the monitoring model.

Check the new KPI in the KPIs Widget as shown in Figure 5-6.



Figure 5-6 The new TotalDailySales KPI in the KPIs widget

Note: In the KPI Manager, instead of using an expression, you can also create this KPI by aggregation of the sales events sent by the monitor model.

5.4.3 Calling the WebSphere Business Monitor REST Service

Now that the dashboard skeleton is created, we need to see how to call the WebSphere Business Monitor REST Services.

The REST services of WebSphere Business Monitor can all be invoked with a URI that has the following syntax:

`http://monitor_server:monitor_server_port/rest/bpm/monitor/REST_URI`

The REST_URI part of the request depends on the type of service you are interested in. The KPI REST services return the value along with the definition of the KPI in a single REST call. They are among the services provided by Business Monitor. The result of the REST calls are always in the JSON format.

In order to get the value and the definition of a KPI, you must refer to the Business Monitor model and version and the id of the KPI. The full URI of a request is:

`http://monitor_server:monitor_server_port/rest/bpm/monitor/model_id/model_version/kpis/value/kpi_id`

This service returns a lot of information about the KPI: not only the KPI value but also the description of the KPI. Table 5-1 on page 190 shows output parameters for this REST call:

Table 5-1 Notable KPI value service output parameters

Parameter Name	Type	Description
KPI Display Name	string	The key performance indicator (KPI) display name.
Target	number	The key performance indicator (KPI) target value. If the data type is “duration”, the target is in milliseconds.
KPI Range Array	array	Array to contain any KPI ranges.
Target Localized	string	The key performance indicator (KPI) target value in localized format.
KPI Value	string	The key performance indicator (KPI) value.
KPI Value Localized	string	The key performance indicator (KPI) value in localized format.

Note: You can find the full description of this REST service in the IBM info center at the following URL:

<http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.bspace.help.api.rest.doc/rest/monitor/index.htm>

In order to be able to call the KPI Service, we need to create two ActionScript classes in our project:

- The `KPIRequest` class that holds the information about a KPI request.
- The `KPIService` class that does the REST call for a particular `KPIRequest`.

To add a new class on the project:

1. Right-click the **ITSOMoviesDashboard** project in the Package Explorer window and click **New** → **ActionScript class**.
2. Enter `KPIRequest` in the Name field in the New ActionScript Class window.
3. Click **Finish**.

Replace the code with the code shown in Example 5-2.

Example 5-2 KPIRequest class

```
package {

    public class KPIRequest
    {
        private var modelID:String;
        private var modelVersion:String;
        private var kpiID:String;

        public function KPIRequest(modelID:String,
                                    modelVersion:String,
                                    kpiID:String)
        {
            this.modelID = modelID;
            this.modelVersion = modelVersion;
            this.kpiID = kpiID;
        }

        public function getRequestURI(serviceURI:String):String
        {
            return serviceURI +
                "/models/" +
                this.modelID +
                "/versions/" +
                this.modelVersion +
                "/kpis/value/" +
                this.kpiID;
        }
    }
}
```

As you can see, this class simply holds the model ID, the model version, and ID of the KPI needed for the REST call.

The `getRequestURI` method builds the REST request URI that will be used by the `KPIService` class.

The second class is the `KPIService` class. Follow the same steps to create a new ActionScript class and use the code shown in Example 5-3 on page 192.

```
package
{
    import com.adobe.serialization.json.JSONDecoder;

    import flash.events.Event;
    import flash.events.IOErrorEvent;
    import flash.net.URLLoader;
    import flash.net.URLRequest;

    public class KPIService
    {
        private static var serviceURI:String
        = "http://MyServer:9080/rest/bpm/monitor"

        public static function load(kpiRequest:KPIRequest,
                                    onLoad:Function,
                                    onError:Function=null):void
        {
            var request:URLRequest = new URLRequest();
            request.url = kpiRequest.getRequestURI(serviceURI);
            request.method = "GET";

            var loader:URLLoader = new URLLoader();

            loader.addEventListener(Event.COMPLETE,
                function(event:Event) : void {
                    try {
                        var decoder:JSONDecoder
                            = new JSONDecoder(event.target.data, false);
                        var values:Object = decoder.getValue();
                    } catch (e:JSONParseError) {
                        if (onError != null)
                            onError("JSON Parse Error");
                        return;
                    }
                    // check to see if returned object is an error
                    var statusCode:String = values["Status Code"];
                    if (statusCode != null) {
                        var errorDetail:String = values["errorDetail"];
                        if (errorDetail == null)
                            errorDetail = "";
                        if (onError != null) onError(errorDetail);
                    }
                }
            );
        }
    }
}
```

```

        } else {
            if (onLoad != null) onLoad(values);
        }

    });

    loader.addEventListener(IOErrorEvent.IO_ERROR,
        function (event:IOErrorEvent):void {
            if (onError != null) onError(event);
        });

    loader.load(request);
}
}
}

```

Note: You will have to change the `serviceURI` variable in the code and replace `MyServer:9080` with the host and port where Business Space is running. We will see how this can be changed so that the code does not contain any hardcoded host name.

The `KPIService` class provides a single static `load` method to load a KPI. You will call this method with the `KPIRequest` and two functions: the `onLoad` function that is called when the request is completed, and the `onError` function that is called in case of an error.

We use the two Flex classes to do the REST call: the `URLRequest` class for the definition of the URI, and the `URLLoader` to load the request defined by the `URLRequest`.

The line `loader.load(request)` launches the asynchronous call.

Event listeners are added on the request loader to handle the following events:

- ▶ `Event.COMPLETE`
An event fired when the REST asynchronous request is completed and the resulting downloaded data is ready to be used.
- ▶ `IOErrorEvent.IO_ERROR`
An event fired if the REST call resulted in an error that terminates the download.

The REST request will return a string in the JSON format. In order to be able to handle it, we need a JSON parser. The Flex platform itself does not provide such

a parser but several third party libraries are available for handling the JSON format on the Flex platform. In our example, we use an open source library (under the BSD license) named `as3corelib` that you can find at the following URL:

<http://github.com/mikechambers/as3corelib>

To compile the code that uses the JSON parser, add the `as3corelib.swc` in the libraries of your project using the following steps:

1. Right-click the **ITSOMoviesDashboard** project in the Package Explorer view of Flash Builder and select **Properties** to open the project properties.
2. In the Properties for ITSOMoviesDashboard window, click **Flex Build Path**.
3. On the Library path tab, click **Add SWC** and add the `as3corelib.swc` file.

The problem of the hard-coded URI to the REST service

In the final application, we do not want to have a hardcoded URI to the Business Monitor REST services because this would force the application to be recompiled when the services URI changes. One way to fix this problem is to deploy the Flex application on the same server and port as the Monitor REST services. The Flex application can then rebuild the REST services URI from the URL that was used to load the Flex application.

Example 5-4 shows how to modify the code of the `KPIService` class to achieve this goal.

Example 5-4 Computing serviceURI from loaderInfo

```
private static var serviceURI:String = getServiceURI();

public static function getServiceURI() : String {
    var loaderUrl:String =
        FlexGlobals.topLevelApplication.loaderInfo.url;
    var serverName:String = URLUtil.getServerNameWithPort(loaderUrl);
    var protocol:String = URLUtil.getProtocol(loaderUrl);

    if (protocol == "file")
        return "http://MyServer:9080/rest/bpm/monitor";

    return protocol+"://"+serverName+"/rest/bpm/monitor";
}
```

Note: You can see that we still have a hardcoded URL here when the protocol is “file.” The reason is that we still want to be able to debug the Flex application while running it with the Flash Builder debugger. These two lines can be removed when debugging is finished.

5.4.4 Displaying the KPI in Elixir components

Now we will use our `KPIService` class and display the resulting KPI values in the Elixir components.

We want our dashboard to be refreshed regularly. To do so, we are going to set a timer that will call our `KPIService` regularly.

Example 5-5 shows the new implementation of the `initializeDashboard` method in the `ITSOMoviesDashboard.mxml`.

Example 5-5 initializeDashboard

```
import mx.collections.ArrayCollection;

private var timer:Timer;

protected function initializeDashboard():void
{
    refreshKPIs();

    timer = new Timer(60000);
    timer.addEventListener(TimerEvent.TIMER, function (e:Event) : void {
        refreshKPIs();
    });
    timer.start();
}

protected function refreshKPIs(): void {

    var modelID:String = "ITSOMovieTicketSalesMM";
    var version:String = "20100304131512";

    // Fill the Total Daily Sales Gauge
    KPIService.load(new KPIRequest(modelID, version,
        'TotalDailySales'), setDailyGaugeValue);

    // Fill the Radar Chart
    radarChart.dataProvider = new ArrayCollection();
```

```

KPIService.load(new KPIRequest(modelID, version,
    'Daily_Sales_for_Egypt'), addKPIToRadarChart);
KPIService.load(new KPIRequest(modelID, version,
    'Daily_sales_for_USA'), addKPIToRadarChart);
KPIService.load(new KPIRequest(modelID, version,
    'Daily_Sales_for_Canada'), addKPIToRadarChart);
KPIService.load(new KPIRequest(modelID, version,
    'Daily_Sales_for_China'), addKPIToRadarChart);
KPIService.load(new KPIRequest(modelID, version,
    'Daily_Sales_for_Germany'), this.addKPIToRadarChart);
KPIService.load(new KPIRequest(modelID, version,
    'Daily_Sales_for_UK'), addKPIToRadarChart);

// Fill the Map
map.dataProvider = new ArrayCollection();

KPIService.load(new KPIRequest(modelID, version,
    'Monthly_Sales_for_Egypt'), addKPIToMap);
KPIService.load(new KPIRequest(modelID, version,
    'Monthly_Sales_for_USA'), addKPIToMap);
KPIService.load(new KPIRequest(modelID, version,
    'Monthly_Sales_for_Canada'), addKPIToMap);
KPIService.load(new KPIRequest(modelID, version,
    'Monthly_Sales_for_China'), addKPIToMap);
KPIService.load(new KPIRequest(modelID, version,
    'Montly_Sales_for_Germany'), addKPIToMap);
KPIService.load(new KPIRequest(modelID, version,
    'Monthly_Sales_for_UK'), addKPIToMap);

}

```

The Flex timer will load our KPIs every minute by calling the `refreshKPIs` method. This method uses the `KPIService` class for each KPI we are interested in. The `TotalDailySales` KPI that we have created in 5.4.2, “Creating a new KPI in WebSphere Business Monitor” on page 187 is shown in the top left gauge named `dailySalesGauge`.

Displaying KPI Values in the gauge component

The `setDailyGauge` method shown in Example 5-6 on page 197 is called when the `TotalDailySales` KPI value is available. This method sets the gauge value to the KPI value and the gauge maximum to the target of the KPI.

Example 5-6 setDailyGaugeValue method

```
private function setDailyGaugeValue(kpiValue:Object) : void {
    dailySalesGauge.maximum
        = Math.max(kpiValue["Target"],kpiValue["KPI Value"]);
    dailySalesGauge.value = kpiValue["KPI Value"];
}
```

The SemiCircularLinearGauge tag is modified as shown in Example 5-7 so that the label displayed in the center of the gauge is formatted as a currency and the tick values are formatted as millions of dollars.

Example 5-7 SemiCircularLinearGauge tag

```
<ibm:SemiCircularLinearGauge
    id="dailySalesGauge"

skinClass="com.ibm.ilog.elixir.gauges.skins.spark.SemiCircularGaugeSkin"

    editable="false"
    title="Total Daily Sales"
    maximum="0"
    height="100%" width="100%"
    labelFormatFunction="formatGaugeLabel"
    tickLabelFunction="formatGaugeTick" />

private static var numberFormatter:NumberFormatter
    = new NumberFormatter();

private static var currencyFormatter:CurrencyFormatter
    = new CurrencyFormatter();

private function formatGaugeTick(item:TickItem) : String {
    numberFormatter.precision = 0;
    return numberFormatter.format( (Number(item.value) / 1000000));
}

private function formatGaugeLabel(value:Object) : String {
    currencyFormatter.currencySymbol='$';
    return currencyFormatter.format(value);
}
```

Displaying KPI Values in the radar chart

Daily sales KPIs by country will be displayed in the radar chart component through the `addKPIToRadarChart` method shown in Example 5-8.

Example 5-8 `addKPIToRadarChart`

```
private function addKPIToRadarChart(kpiValue:Object) : void {  
    var dataSource:ArrayCollection  
        = radarChart.dataProvider as ArrayCollection;  
    dataSource.addItem(kpiValue);  
}
```

This method simply adds the KPI to the data displayed by the chart.

In Example 5-9 we define the series for the radar chart and the axis.

Example 5-9 `RadarChart` tag

```
<ilog:RadarChart id="radarChart" width="100%" height="100%"  
    dataTipFunction="radarChartDataTipFunction"  
    showDataTips="true">  
    <ilog:series>  
        <ilog:RadarLineSeries dataField="KPI Value"/>  
    </ilog:series>  
    <ilog:angularAxis>  
        <ilog:AngularAxis categoryField="KPI Display Name"/>  
    </ilog:angularAxis>  
  
    <ilog:radialAxisRenderers>  
        <mx:AxisRenderer labelFunction="radarAxisLabelFunction"  
            horizontal="true" visible="true"/>  
        <mx:AxisRenderer horizontal="false" visible="false"/>  
    </ilog:radialAxisRenderers>  
</ilog:RadarChart>
```

The radar chart displays only one series: the values of the KPIs for each country. The `dataField` of the series is set to `KPI Value`: the field of the KPI object that contains the value of the KPI.

The radar chart has two axes:

- The angular axis

This axis is always a category axis. Here each category is a distinct KPI. We set the `categoryField` property to the `KPI Display Name` property.

► The radial axis

In our case, the radial axis is a linear axis that displays the value of the KPI.

An axis renderer defines how the axes are rendered. The vertical axis is hidden and the horizontal axis uses the `radarAxisLabelFunction` method (shown in Example 5-10) to format the label to a currency:

Example 5-10 radarAxisLabelFunction method

```
private function radarAxisLabelFunction(axisRenderer:IAxisRenderer,
                                       label:String):String {
    return currencyFormatter.format(label);
}
```

The `radarChartDataTipFunction` also uses the values of the KPI object to display a tooltip for each KPI. Example 5-11 shows the tooltip function.

Example 5-11 Function computing tooltip for radar chart

```
private function radarChartDataTipFunction(hitData:HitData):String {
    return hitData.item["KPI Display Name"]+"\n"+
           "Value:" + hitData.item["KPI Value Localized"] + "\n"+
           "Target:" + hitData.item["Target Localized"];
}
```

Displaying KPI Values in the map component

For the monthly sales KPIs, the method is similar to the `RadarChart` component, but they are displayed in the map component though the `addKPIToMap` method shown in Example 5-12.

Example 5-12 addKPIToMap

```
private function addKPIToMap(kpiValue:Object) : void {

    var key:String = kpiToCountryKey[kpiValue['KPI ID']];

    kpiValue['key'] = key;

    var dataSource:ArrayCollection
        = map.dataProvider as ArrayCollection;

    dataSource.addItem(kpiValue);
}

private static var kpiToCountryKey:Object =
```

```
{'Monthly_Sales_for_USA' :    'USA',
'Monthly_Sales_for_UK' :     'GBR',
'Monthly_Sales_for_China' :  'CHN',
'Monthly_Sales_for_Canada' : 'CAN',
'Monthly_Sales_for_Egypt' :  'EGY',
'Monthly_Sales_for_Germany' : 'DEU'};
```

Just like for the RadarChart component, the KPI object retrieved from the REST service is directly added in the data provider for the map component. But for the map component, there is an additional step. The map component needs to know on which country each KPI must be displayed. By default, the Map component looks into each data in its data provider for the value of the key field to know where this data must be displayed on the map. The value of the key property must correspond to the name of a country on the map.

We can now modify the WorldCountriesMap tag as shown in Example 5-13 to add interaction to the map.

Example 5-13 WorldCountriesMap tag

```
<ibm:WorldCountriesMap id="map" width="100%" height="100%"
    mouseDown="mapMouseDownHandler(event)"
    skinClass="MapSkin"/>
```

Note: For the map components of Elixir, an element displayed in the map, i.e. a country, a state, or any kind of polygon displayed by the map, is called a map feature, and is represented in ActionScript by the MapFeature class.

The mapMouseDownHandler method is called when the user clicks the map. When the user clicks a country (a map feature), we will show the KPI value in the monthly sales gauges, as shown in Example 5-14.

Example 5-14 mapMouseDownHandler function

```
protected function mapMouseDownHandler(e:MouseEvent):void
{
    var mf:MapFeature = map.getFeatureAt(e.stageX, e.stageY);
    if(mf) {
        var kpiValue:Object = mf.item;
        if (kpiValue) {
            monthlySalesGauge.value = kpiValue['KPI Value'];
            monthlySalesGauge.maximum
                = Math.max(kpiValue['Target'], kpiValue['KPI Value']);
            monthlySalesGauge.title = kpiValue['KPI Display Name'];
        }
    }
}
```

```
}  
}
```

The last step for displaying the KPI value in the map component is to create a new renderer for the map feature. The `MapFeatureRenderer` class will render the map feature on the screen. In the WebSphere Monitor Model, each KPI can define a set of ranges for the value of the KPI and associate a different color for each range. For example, for a low value of the KPI, we will use red, and for a high value, we will use green. We will create a sub-class of the `MapFeatureRenderer` class so that the polygon displaying the country uses the ranges and range colors defined in the KPI.

In the Flex project, add a new MXML File named `KPIMapFeatureRenderer.mxml` by following these steps:

1. Right-click **ITSOMoviesDashboard** in the Package Explorer window.
2. Click **New** → **MXML Item Renderer**.
3. In the new MXML Item Renderer window, type `KPIMapFeatureRenderer` for the name.
4. Click **Finish**. The new file is created.
5. Replace the code with the code shown in Example 5-15.

Example 5-15 KPIMapFeatureRenderer.mxml

```
<?xml version="1.0" encoding="utf-8"?>  
  
<ibm:MapFeatureRenderer  
    xmlns="http://ns.adobe.com/mxml/2009"  
    xmlns:s="library://ns.adobe.com/flex/spark"  
    xmlns:ilog="http://www.ilog.com/2007/ilog/flex"  
    xmlns:ibm="http://www.ibm.com/xmlns/prod/ilog/elixir/2010"  
    contentBackgroundColor="{getColor(data.item)}">  
    <Script>  
        <![CDATA[  
            import com.ibm.ilog.elixir.maps.MapFeature;  
  
            static private const contentFill:Array = ["fill"];  
  
            override protected function get backgroundItems():Array {  
                return contentFill;  
            }  
  
            static private const borderStroke:Array = ["stroke"];  
  
            override protected function get borderItems():Array {
```

```

        return borderStroke;
    }

    private function getColor(kpiObject:Object):int {

        if(kpiObject == null) return 0xddffff;

        var kpiValue:Number = Number(kpiObject['KPI Value']);
        var ranges:Array = kpiObject['KPI Range Array'] as Array;
        for each (var range:Object in ranges) {
            if (kpiValue < Number(range['KPI Range End Value']) &&
                kpiValue >= Number(range['KPI Range Start Value'])) {
                return uint(String(range['KPI Range
Color']).replace("#","0x"));
            }
        }
        return 0xc0c0c0;
    }
}]>
</Script>
<!-- states -->
<ibm:states>
    <s:State name="normal"/>
    <s:State name="hovered"/>
    <s:State name="selected"/>
    <s:State name="selectedAndHovered"/>
</ibm:states>

<ibm:GraphicPath commands="{mapFeature.graphicsPathCommands}">
    <ibm:fill>
        <s:SolidColor id="fill"/>
    </ibm:fill>
    <ibm:stroke>
        <s:SolidColorStroke id="stroke" weight="0" scaleMode="none"/>
    </ibm:stroke>
</ibm:GraphicPath>
</ibm:MapFeatureRenderer>

```

The interesting part in the code is mainly the `getColor` method. This method computes the color of the polygon (`GraphicPath` in the code) used to render the country. The method uses the colors defined in the value ranges that have been defined in the KPI itself.

The `kpiObject['KPI Range Array']` returns an array or range; each range has a start value ('KPI Range Start Value'), an end value ('KPI Range End Value'), and a color ('KPI Range Color').

To indicate to the map component that it must use our new feature renderer, we must create a new skin for the map component itself. The information about which map feature renderer is used by the map is defined in the skin of the component.

To create the new skin, we will duplicate the default skin of the `WorldCountriesMap` component:

1. Right-click **ITSOMoviesDashboard** in the Package Explorer window.
2. Click **New** → **MXML Skin**.
3. In the new MXML Skin window, type `MapSkin` for the name.
4. Type `com.ibm.ilog.elixir.maps.components.WorldCountriesMap` in the Host component field.
5. Select the “Create as copy of:” check box and type `com.ibm.ilog.elixir.maps.skins.spark.MapSkin` in the field.
6. Click **Finish**.

Figure 5-7 on page 204 shows the new MXML Skin window.

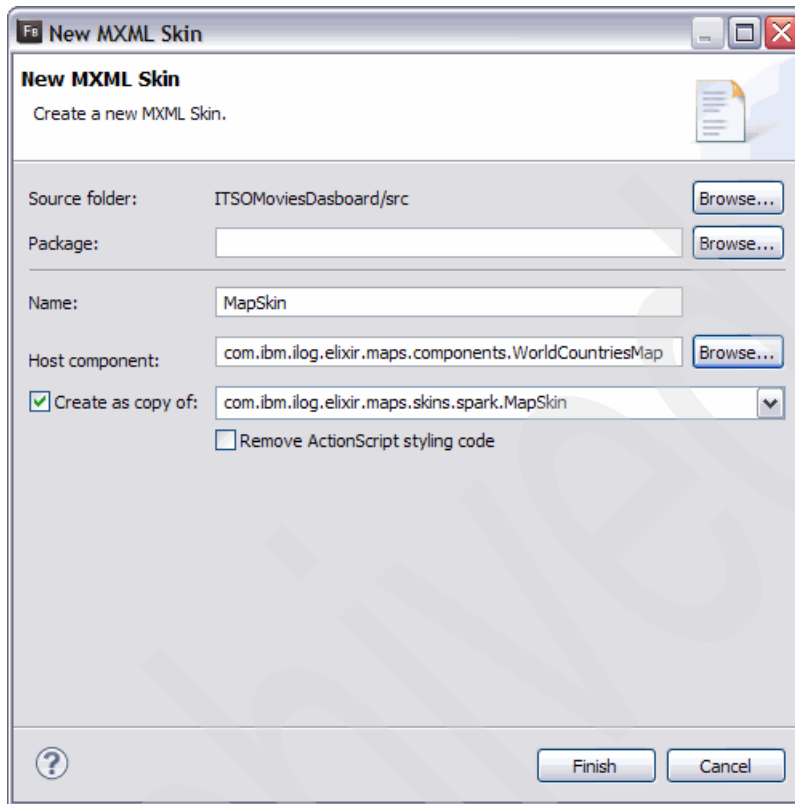


Figure 5-7 New MXML Skin window

This procedure creates a new `MapSkin.mxml` in our project. We will just change the last lines of the file.

- In the `<MapFeatureGroup>` tag, replace the default renderer: `com.ibm.ilog.elixir.maps.skins.spark.DefaultFeatureRenderer` with our renderer, `KPIMapFeatureRenderer`, as shown in Example 5-16.

Example 5-16 MapFeatureGroup tag

```
<ibm:MapFeatureGroup id="featureGroup"
    featureRenderer="KPIMapFeatureRenderer"/>
```

The Elixir Dashboard application is finished and you can now run it directly inside a web browser.

Note: If you are not logged in the Business Space environment, each call to the Business Monitor REST service will require you to enter the login and password. The application needs to be embedded in the Business Space environment; this is the subject of the following sections.

5.5 Deploying the Elixir application using the Web Site widget

In this section, we will we will deploy our Elixir dashboard in a new business space using the Web Site widget.

This will be done in two steps. We will first deploy all the necessary files to the web server that is running Business Space and WebSphere Business Monitor, and then we will create the Web Site widget so that it shows our dashboard.

5.5.1 Deploying the Elixir Dashboard in the Business Space server

To deploy the Elixir dashboard we have created in the previous section, we will create a web project in WebSphere Integration Developer (WID) and deploy this application on the server. The Flex dashboard will thus be deployed on the same server as Business Space.

In WID, ensure that you are showing the web perspective. If you are not in the web perspective, click **Window** → **Open Perspective** → **Web** in the menu bar.

1. Click **File** → **New** → **Dynamic Web Project** to create a new Dynamic Web Project.
2. In the New Dynamic Web Project window, type `ITSOMoviesElixirDashboard` for the Project name.
3. In the EAR Membership area, select **Add project to an EAR** and type `ITSOMoviesElixirDashboardEAR` for the EAR Project Name.
4. Click **Finish**.

Figure 5-8 on page 206 shows the New Dynamic Web Project window.

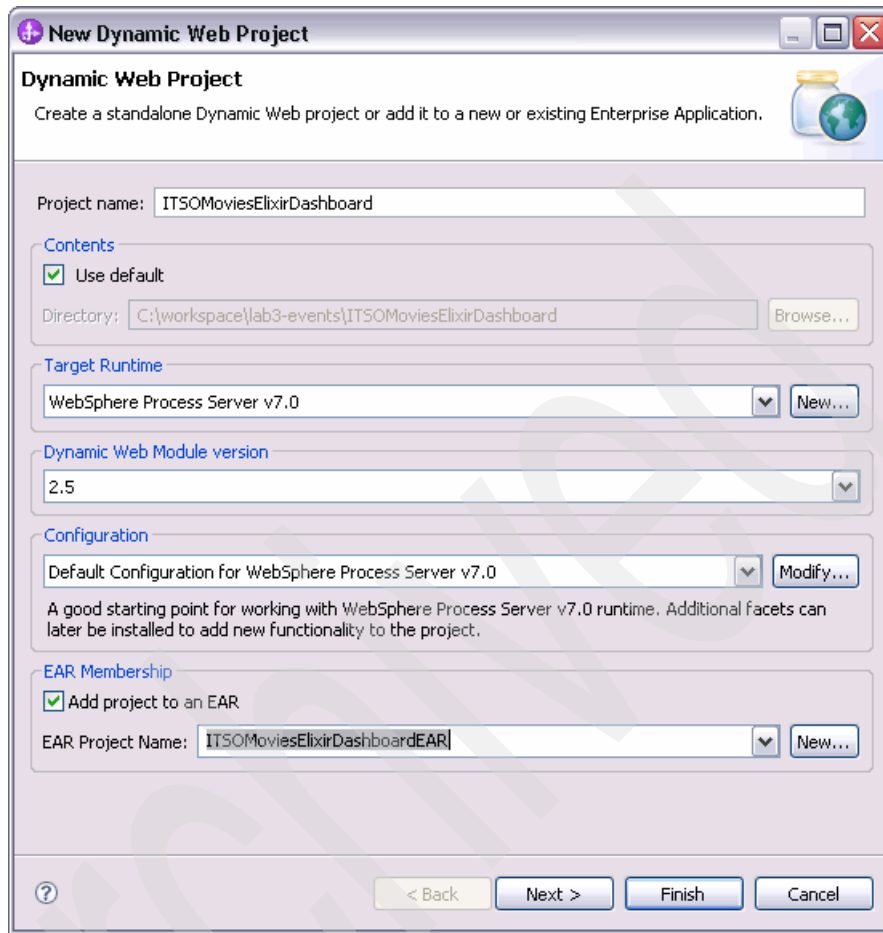


Figure 5-8 New Dynamic Web Project window

In the Enterprise Explorer click **ITSOMoviesElixirDashboard** to expand the content. You should see the WebContent directory. Place the necessary files in this WebContent directory. Browse to the directory where the Flex ITSOMoviesDashboard project was created. In the bin-debug folder you will find the following files:

- ▶ ITSOMoviesDashboard.html
The html file that opens the Flex application
- ▶ ITSOMoviesDashboard.swf
The compiled Flex application

- ▶ `playerProductInstall.swf`
A Flex application for upgrading the Flash player
- ▶ `swfobject.js`
JavaScript library to embed the Flex application in the HTML file

Copy these files to the `WebContent` directory (for example, using drag and drop from a file explorer).

We will now deploy the project to the server:

1. Right-click the **ITSOMoviesElixirDashboardEAR** project in the Enterprise Explorer window, and select **Run As** → **Run On Server**. The Run On Server window appears as in Figure 5-9 on page 208.

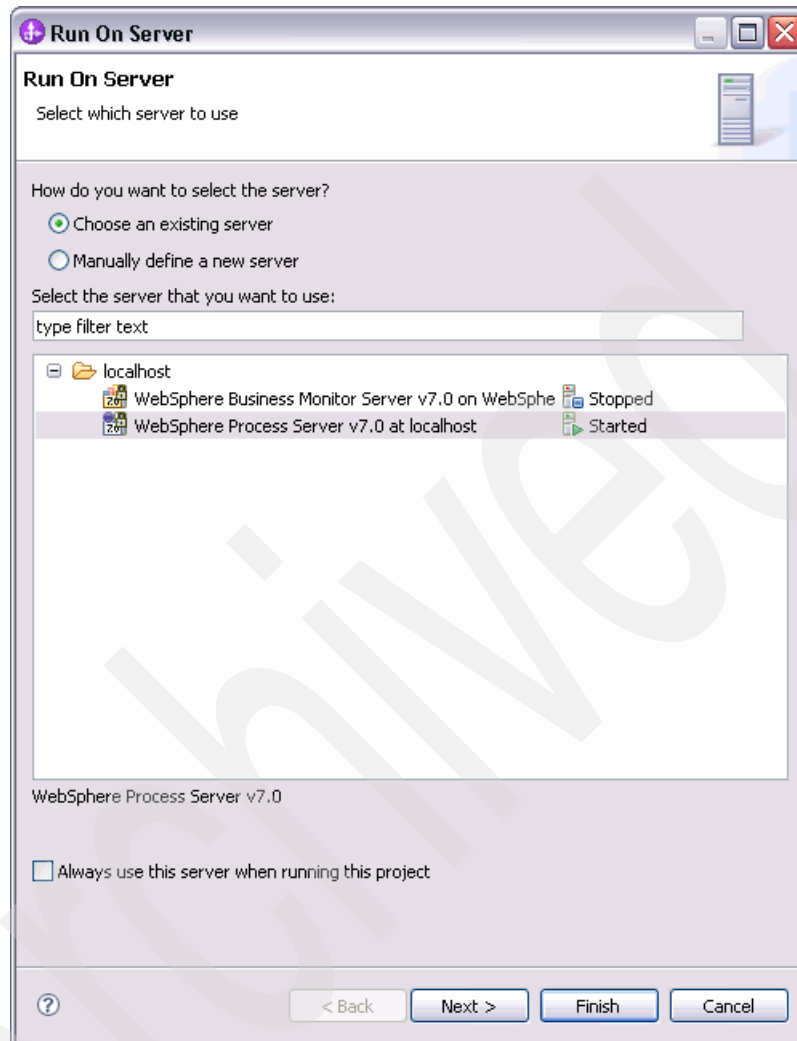


Figure 5-9 Run On Server window

2. Select the server that is running Business Space and click **Finish**.

All the necessary files are now deployed on the server. Open the dashboard html file from its URL on the server:

`http://MyServer:MyPort/ITSOMoviesElixirDashboard/ITSOMoviesDashboard.html`

5.5.2 Creating the Web Site widget

When the necessary files are deployed on the server, we can create the Business Space that will display our dashboard.

Creating the Business Space

To create the business space, perform the following steps:

1. Log in to WebSphere Business Monitor Business Space with the credentials for an administrator.
2. Click **Actions** → **Create Space**.
3. Type Elixir Dashboard Business Space in the Space name field.
4. Select a theme for your business space.
5. Select an icon for the new space.
6. Click **Save**.

Figure 5-10 shows the Create Space window.

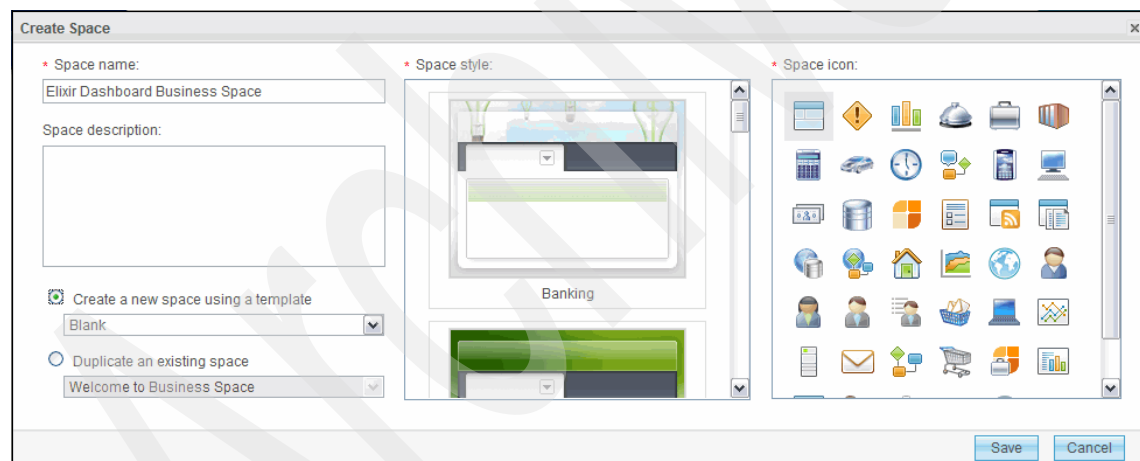


Figure 5-10 Create Space window

Creating pages

After creating the business space, the next step is to create a page for the Elixir Dashboard business space. The creation of the Elixir Dashboard business space has already created the first page (Page 1). We will need to rename this page to host the Web Site widget.

To rename page 1, perform the following steps:

1. Open the page menu by clicking the action button, which is the arrow beside the page name on the page tab.
2. Click **Edit Settings**.
3. Type Elixir Dashboard in the page name field.
4. Click **Save**.

Adding the Web Site widget to the Elixir dashboard page

We will now add the Web Site widget to the page. The Web Site widget allows us to add a web page inside a Business Space page.

Perform the following steps:

1. Navigate to the Elixir Dashboard page and click the **Edit Page** button.
2. Drag and drop the **Web Site** widget from the widget palette. By default the widget shows the IBM website as shown in Figure 5-11.



Figure 5-11 Creation of the Web Site widget

3. Click the little down arrow on the right side of the Web Site widget Header and select **Edit Settings**.

4. In the Web Site Edit window, type the URL of the dashboard, that is, `http://MyServer:MyPort/ITSOMoviesElixirDashboard/ITSOMoviesDashboard.html`. You need to adapt the server name and port to your own server configuration.

You should see the dashboard inside the Web Site widget as shown in Figure 5-12.

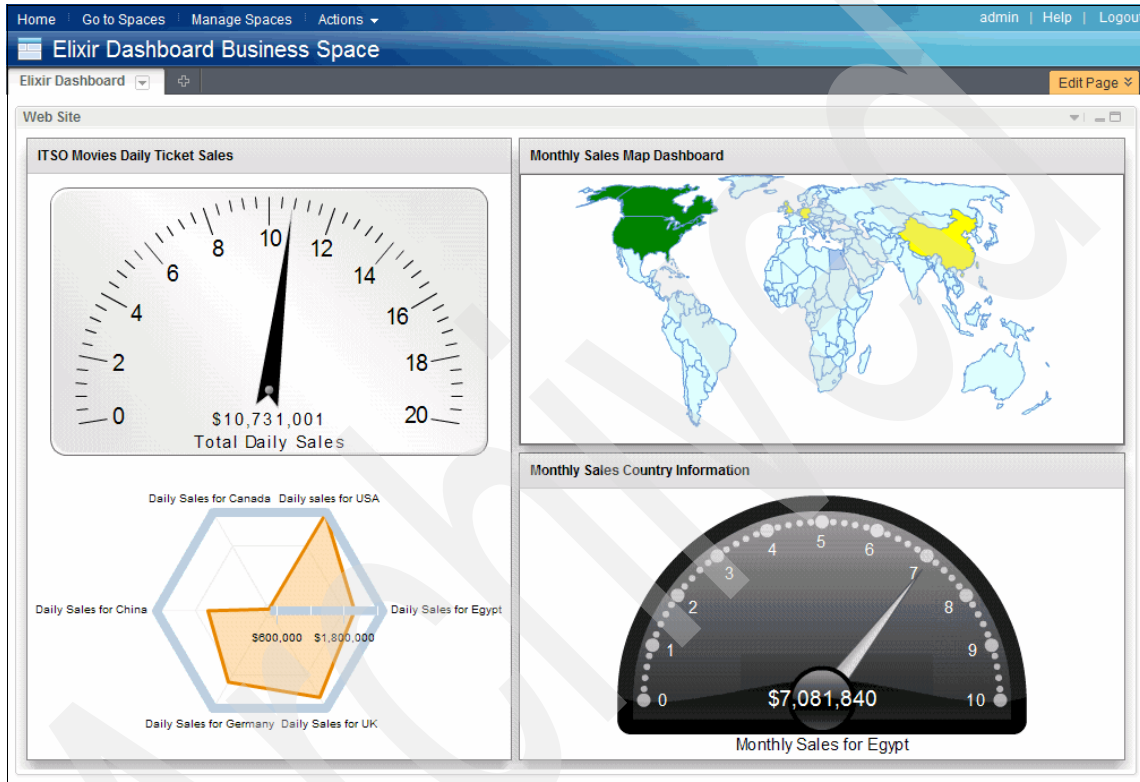


Figure 5-12 The ITSOMovies Elixir Dashboard in the web site widget

5.6 Creating an iWidget to deploy in Business Space

In 5.5, “Deploying the Elixir application using the Web Site widget” on page 205, we used the Web Site widget of Business Space to display our dashboard.

The Web Site widget is easy to use because it only requires a URL for the configuration. However, in our scenario using this widget has many drawbacks. The user who is creating the business space needs to know the URL of the

dashboard to be able to use it. More than that, there is no real possibility to create a configurable dashboard.

In this section, we will create a new iWidget that will display our dashboard. We will also show how to add a new parameter to the dashboard (the refresh rate).

We will then deploy this iWidget in the Business Space environment.

Note: We are going to create a simple iWidget here: our widget will not publish or subscribe to events.

Our dashboard is built as a Flex application that uses several Flex components linked together. Another possibility would be to create a Flex application and an associated iWidget for each Elixir component used in our dashboard. With several iWidgets, it would be possible to create various dashboards by wiring iWidget events.

If you want to learn more about iWidgets and how to create a more complex iWidget with Elixir, refer to Chapter 4, “Creating IBM ILOG Elixir iWidgets for Mashup Center” on page 135.

Although the implementation steps of the iWidget are common for widgets dedicated to Business Space and Mashup Center, the deployment steps are specific for each product.

For the development of iWidget inside the Business Space environment, refer to the documentation available at the following URL:

http://publib.boulder.ibm.com/infocenter/dmndhelp/v7r0mx/index.jsp?topic=/com.ibm.bspace.help.devt.doc/doc/developing_widgets/developingwidgets.html

5.6.1 Creating the iWidget definition

To have tooling support in WebSphere Integration Developer to develop custom widgets, make sure that the Web Development Tools feature is installed. Check the WebSphere Integration Developer package group in the IBM Installation Manager for this feature. If this feature is not installed, you will not be able to use the iWidget wizard and the Universal Test Client for iWidgets.

Note: The source code for the WebSphere Integration Developer project that is created in this section is available on the web. Check Appendix A, “Additional material” on page 287 for details on how to download it.

To create a widget in WebSphere Integration Developer, perform the following steps:

1. Click **File** → **New** → **Other**.
2. Click **Web** → **Dynamic Web Project**, and click **Next**.
3. Type `ITS0MoviesDashboardWidget` into the Project name field.
4. Type `ITS0MoviesDashboardWidgetEAR` in the EAR Project Name field and select **Add project to an EAR**.
5. Leave all the other defaults as they are and click **Finish**. The new project is created. When prompted, switch to the web perspective.
6. Create a new folder under WebContent:
 - a. Right-click **WebContent** and select **New** → **Folder**.
 - b. Type `DashboardWidget` as the folder name.
 - c. Click **Finish**.
7. Create a new widget:
 - a. Right-click **DashboardWidget** and select **New** → **Other**.
 - b. Select **Web** → **iWidget** and click **Next** as shown in Figure 5-13 on page 214.

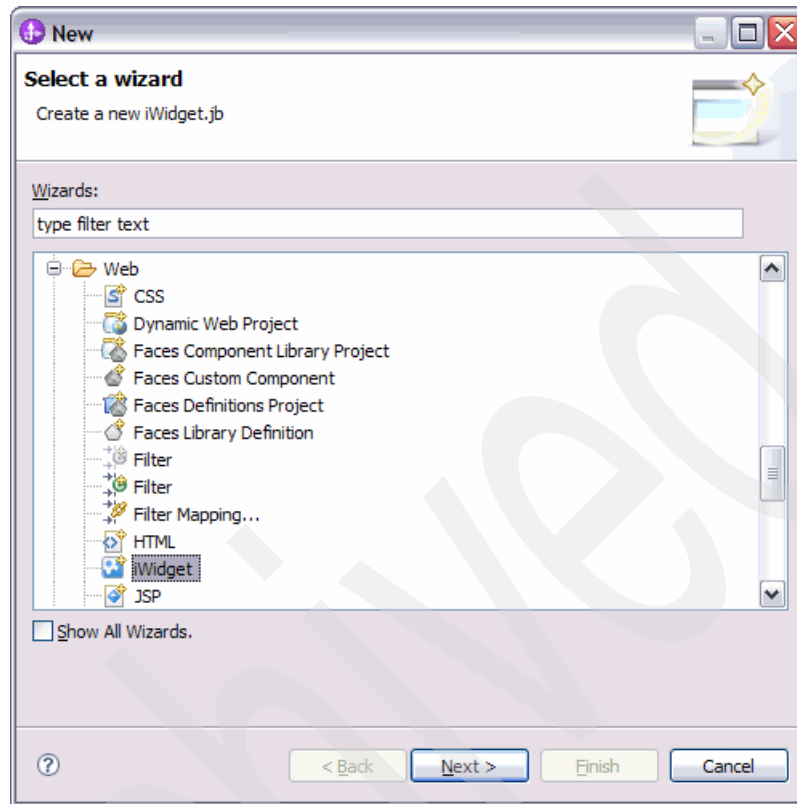


Figure 5-13 Create a new widget

- c. A widget specific wizard page is shown (Figure 5-14 on page 215). Type `ITSOMoviesDashboardWidget` as the `iWidget` name and `/ITSOMoviesDashboardWidget/WebContent/DashboardWidget` as the source folder. This folder will be the folder where we will put all the necessary files for the `iWidget`.
- d. Select **Simple Widget** for the widget type.
- e. Select the **Edit mode** check box as shown in Figure 5-14 on page 215 and click **Finish**.

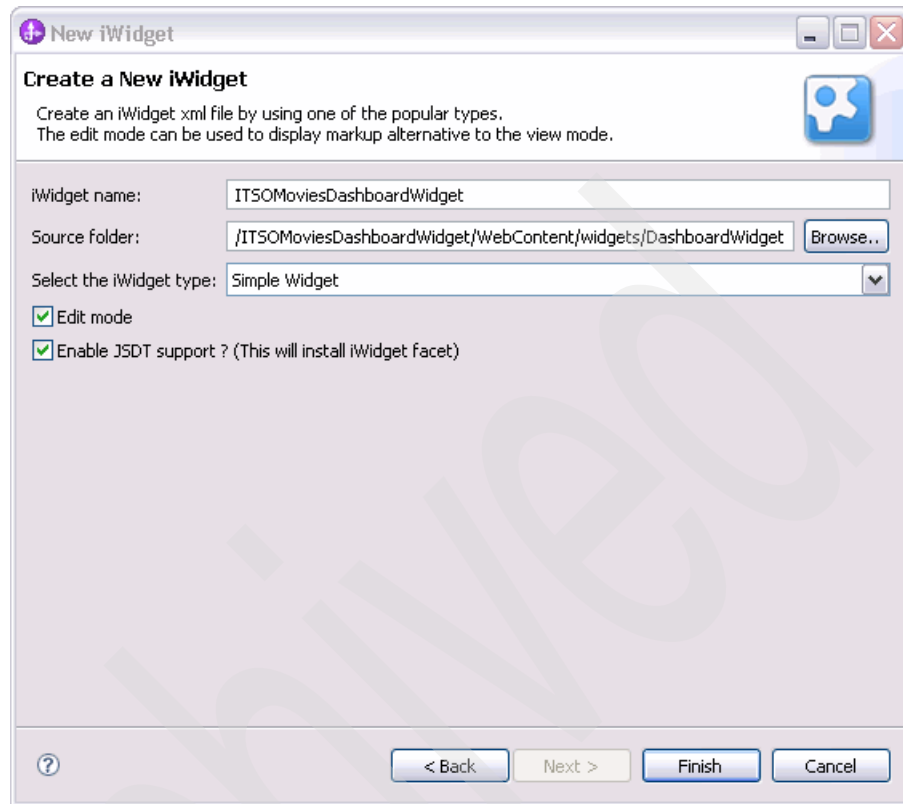


Figure 5-14 The New iWidget window

After this is done, the new widget is created and the iWidget Editor opens as shown in Figure 5-15 on page 216.

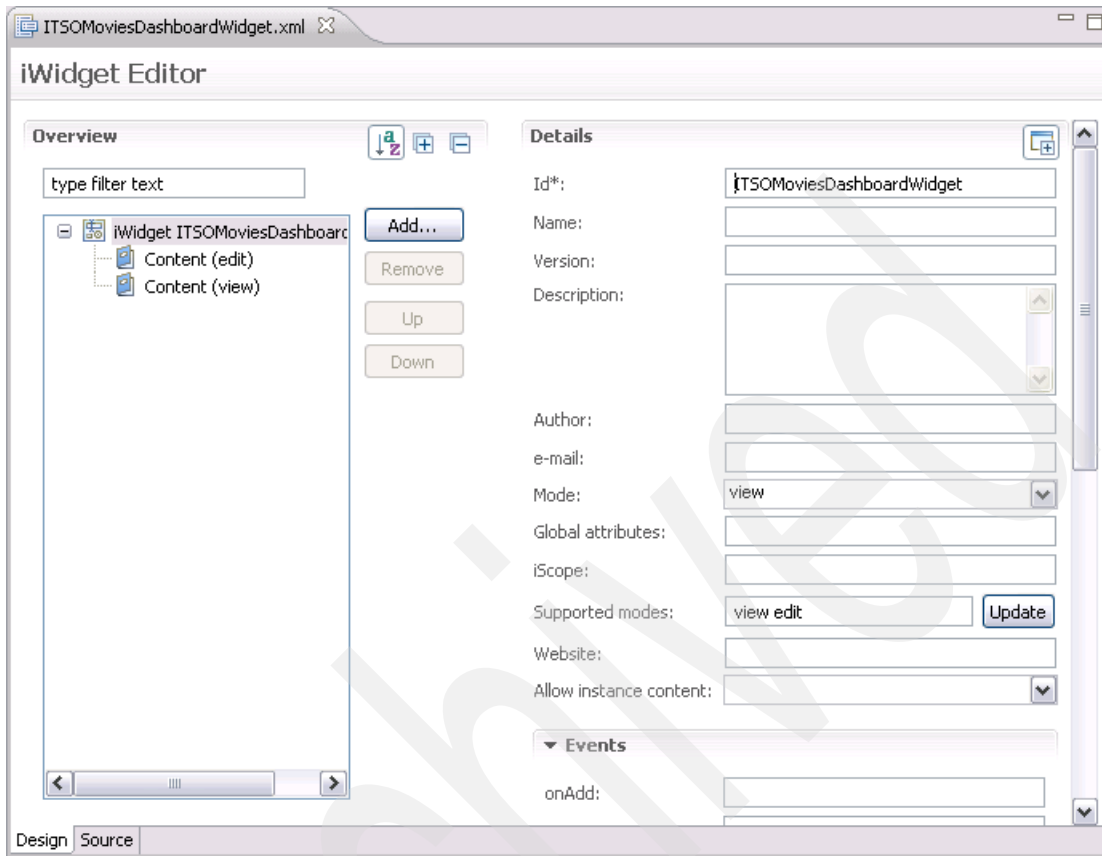


Figure 5-15 The iWidget Editor

This editor will edit the widget definition file, which is an XML file. In our scenario, the file is named `ITSOMoviesDashboardWidget.xml` and is located in the `WebContent/DashboardWidget` folder.

We want to change a few things in this file in preparation for the widget implementation:

1. Type `itsomovies.ITSOMoviesDashboardWidget` for the `iScope`. This is the name of the class implementing the widget in Javascript.
2. On the left side, in the Overview section, click **Content (view)** and in the Details field, set the code shown in Example 5-17.

Example 5-17 Content in view mode

```
<div id="_IWID_ITSOMoviesDashboardWidgetViewHolder"></div>
```

This HTML fragment represents the initial HTML displayed by the iWidget in view mode.

3. Add a new `elixir.js` resource in the definition file. To create the resource, follow these steps:
 - a. In the iWidget Editor click **Add**.
 - b. Select **Resource** and click **OK**.
 - c. Type `elixirId` for the `Id` field and `elixir.js` for the `src` field.
4. Create a JavaScript file named `elixir.js` that contains the code shown in Example 5-18.

The mediator code inside the `elixir.js` file will be in charge of registering iWidgets and making their methods available to the Adobe Flex application that is using the IBM ILOG Elixir component. This will typically allow the application to synchronize the iWidget context in JavaScript with the Flex side of the widget. It is described as a resource in the descriptor file so that it will be loaded at the same time as the widget.

Example 5-18 elixir.js

```
var ibm_ilogvisu_redbook = {};  
  
(function () {  
  
    var hash = {};  
  
    /**  
     * Call a iWidget function from  
     * the Flex side.  
     */  
    ibm_ilogvisu_redbook.invokeWidget = function(widgetId, fnName,  
    params) {  
        var widget = ibm_ilogvisu_redbook.getWidget(widgetId);  
        if (widget) {  
            var fn = widget[fnName];  
            if (fn) {  
                try {  
                    return fn.call(widget, params);  
                } catch (e) {  
                    console.log("error on function call");  
                }  
            } else {  
                console.log("can't find function");  
            }  
        } else {  
            console.log("can't find function");  
        }  
    }  
})
```

```

        console.log("can't find widget");
    }
    return undefined;
};

/**
 * Get a handle on iWidget function.
 */
ibm_ilogvisu_redbook.getWidget = function(widgetId) {
    return hash[widgetId];
};

/**
 * Register a iWidget.
 */
ibm_ilogvisu_redbook.addWidget = function(widgetId, widget) {
    hash[widgetId] = widget;
};

/**
 * De-register a iWidget.
 */
ibm_ilogvisu_redbook.removeWidget = function(widgetId) {
    delete hash[widgetId];
};
})();

```

5.6.2 Adding the Elixir dashboard into the widget

The Flex dashboard contained in the ITSOMoviesDashboard.swf file must be added to the widget project.

Browse to the directory where the Flash Builder ITSOMoviesDashboard project was created. In the bin-debug folder you will find the ITSOMoviesDashboard.swf file. This file must be added in the WebContent/DashboardWidget folder of our enterprise application. To do so, use drag and drop from a file explorer or perform the following steps:

1. Right-click the **DashboardWidget** folder under WebContent in WID and select **Import**.
2. In the Import window, select **General**, then **File System**.
3. Click **Next**.

4. In the Import window, in the From directory field, locate the folder of the Flash Builder ITSOMoviesDashboard project bin-debug directory and click **OK**.
5. Check the ITSOMoviesDashboard.swf file and click **Finish**.

5.6.3 Implementation of the widget

The implementation of an iWidget will be done in JavaScript with the Dojo Toolkit. We are going to create the ITSOMoviesDashboardWidget.js JavaScript file that will contain the code for the iWidget.

Perform the following steps to create the JavaScript implementation file:

1. Right-click **DashboardWidget** folder in the Enterprise Explorer and select **New → File**.
2. Type ITSOMoviesDashboardWidget.js into the File name field and click **Finish**.

This JavaScript file must also be added as a resource in the iWidget definition file. Go back to the iWidget Editor (ITSOMoviesDashboardWidget.xml editor) and perform these steps:

1. In the iWidget Editor click **Add**.
2. Select **Resource** and click **OK**.
3. Type jsId for the Id field and ITSOMoviesDashboardWidget.js for the src field.

For the iWidget, we will define a Dojo class that does not have any super class.

Example 5-19 shows the initial empty implementation.

Example 5-19 iWidget implementation skeleton

```
dojo.declare("itsomovies.ITSOMoviesDashboardWidget", null,{

    // hold a reference to the id of the Flex object
    flexId : null,
    // the widget name
    widgetName : "ITSOMoviesDashboardWidget",

    onLoad:function() {
        // when loading register ourselves
        ibm_ilogvisu_redbook.addWidget(this.iContext.widgetId, this);
    },

    onUnload:function() {
        // on unload un-register ourselves
        ibm_ilogvisu_redbook.removeWidget(this.iContext.widgetId);
    }
});
```

```

    },

    onView:function() {

    },

    onSizeChanged:function(/* com.ibm.mashups.iwidget.IEvent */ event){

    },

    onRefreshNeeded:function(/* com.ibm.mashups.iwidget.IEvent */ event)
    {
    },

    onNavStateChanged:function(/*com.ibm.mashups.iwidget.IEvent*/ event)
    {
    }
  });

```

In this code, the `onLoad` and `onUnload` methods are implemented to register and unregister our widget in the mediator object defined in the `elixir.js` file. This registration stores a reference between the widget ID and the JavaScript instance. With this association, we will be able to call JavaScript methods from the Flex code, and also Flex methods from the JavaScript code.

The onView method

The `onView` method shown in Example 5-20 is called when the widget is displayed. In this method, we add the HTML elements required to load the Flash plugin. The Flash plugin will then load and display the `ITSOMoviesDashboard.swf` application.

Example 5-20 The iwidget onView method

```

onView:function() {

    if (this.flexId != null)
        return;

    // replaces _IWID_ in the xml file with the widget's id to find the
    node
    var widgetId = this.iContext.widgetId;
    var holderNodeId = "_" + widgetId + "_" + this.widgetName +
    "ViewHolder";
    var holderNode = this.iContext.getElementById(holderNodeId);

```

```

// create a div to hold the Flash content
var flashContent = document.createElement("div");
dojo.style(flashContent, { "height" : "100%" });
this.flexId = this.iContext.widgetId+"_Flex"+this.widgetName;
// get the protocol
var protocolForCodebase = "http";
if (this.iContext.io.widgetBaseUri.indexOf("https") >-1) {
    protocolForCodebase = "https";
}
var flashCodebase =
protocolForCodebase+"://download.macromedia.com/pub/shockwave/cabs/flash/swflash.cab#version=10,0,0,0";
// url points to the Flash swf file;
var url = this.iContext.io.widgetBaseUri+"ITSOMoviesDashboard.swf";
// we need to rewrite url in case of proxy usage
url = this.iContext.io.rewriteURI(url);
// Place the Flash content in the div
holderNode.appendChild(flashContent);
flashContent.innerHTML = "<object id=\""+this.flexId
+ "\" classid=\"clsid:D27CDB6E-AE6D-11cf-96B8-444553540000\" \"
+ "codebase=\""+ flashCodebase + "\" \"
+ "height=\"100%\" width=\"100%\">\"
+ "<param name=\"wmode\" value=\"opaque\">\"
+ "<param name=\"src\" value=\""+url+"\"/>\"
+ "<param name=\"flashVars\" value=\"widgetId="+widgetId+"\"/>\"
+ "<embed id=\""+this.flexId+"\" name=\""+this.flexId+"\"
src=\""+url+"\"
pluginspage=\"http://www.macromedia.com/shockwave/download/index.cgi?P1
_Prod_Version=ShockwaveFlash\" \"
+ "height=\"100%\" width=\"100%\" \"
flashVars=\"widgetId="+widgetId+"\" wmode=\"opaque\"/></object>";
},

```

The onSizeChanged method

Example 5-21 shows the implementation of the onSizeChanged method that is called when the widget is resized.

Example 5-21 The iWidget onSizeChanged method

```

onSizeChanged:function(/* com.ibm.mashups.iwidget.IEvent */ event) {

    var widgetId = this.iContext.widgetId;
    var holderNodeId = "_" + widgetId + "_" + this.widgetName +
    "ViewHolder";

```

```

    var holderNode = this.iContext.getElementById(holderNodeId);
    var newWidth = event.payload.newWidth;
    var newHeight = event.payload.newHeight;
    dojo.style(holderNode, {
        "height" : (newHeight-3)+"px"
    });
},

```

With these methods implemented, we have a workable iWidget that can display our Flex dashboard and allow the communication between Flex and JavaScript.

Adding images for the widget

Several images will be displayed in the Business Space environment for our iWidget: an image to represent our widget in the list of available widgets and an image that displays a preview of the widget. Those images can be located anywhere. For our scenario, we will simply add them in the widget enterprise application in the DashboardWidget/images directory.

We created a 28x28 image named ITS0DashboardWidget.gif for the icon and a 160x125 image named ITS0DashboardWidgetPreview.gif for the preview image.

At this step, we have all the files that we need in our iWidget project. Figure 5-16 shows the WebContent folder of the project.

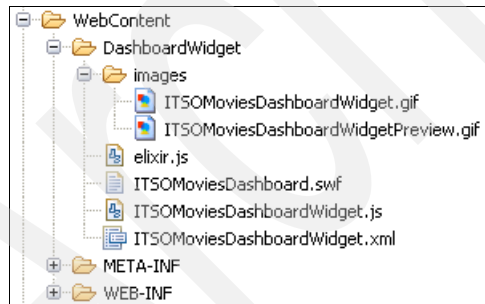


Figure 5-16 WebContent folder

5.6.4 Testing the widget

WebSphere Integration Developer provides a Universal Test Client for iWidgets that allows users to test an iWidget before it is deployed in the Business Space environment.

Before running the Universal Test Client for iWidgets, you might want to check your configuration for the web browser because if you run the Universal Test Client using the internal web browser of WebSphere Integration Developer, you will not be able to use a JavaScript debugger such as Firebug. To check your configuration, perform the following steps:

1. In the WebSphere Integration Developer main menu select **Window** → **Preferences**
2. Click **General** → **Web browser** and choose your browser.

When you use the Universal Test Client for iWidget with Firefox and Firebug, you have a good environment for testing your iWidget. You will also want to test the iWidget under Microsoft Internet Explorer regularly to ensure it works properly in this browser as well.

To start testing with the Universal Test Client:

1. Right-click the iwidget definition file (`ITSOMoviesDashboardWidget.xml`) in the project and select **Run As** → **Run on Server**.
2. Select the server and click **Finish**.

Figure 5-17 on page 224 shows the widget in the Universal Test Client.

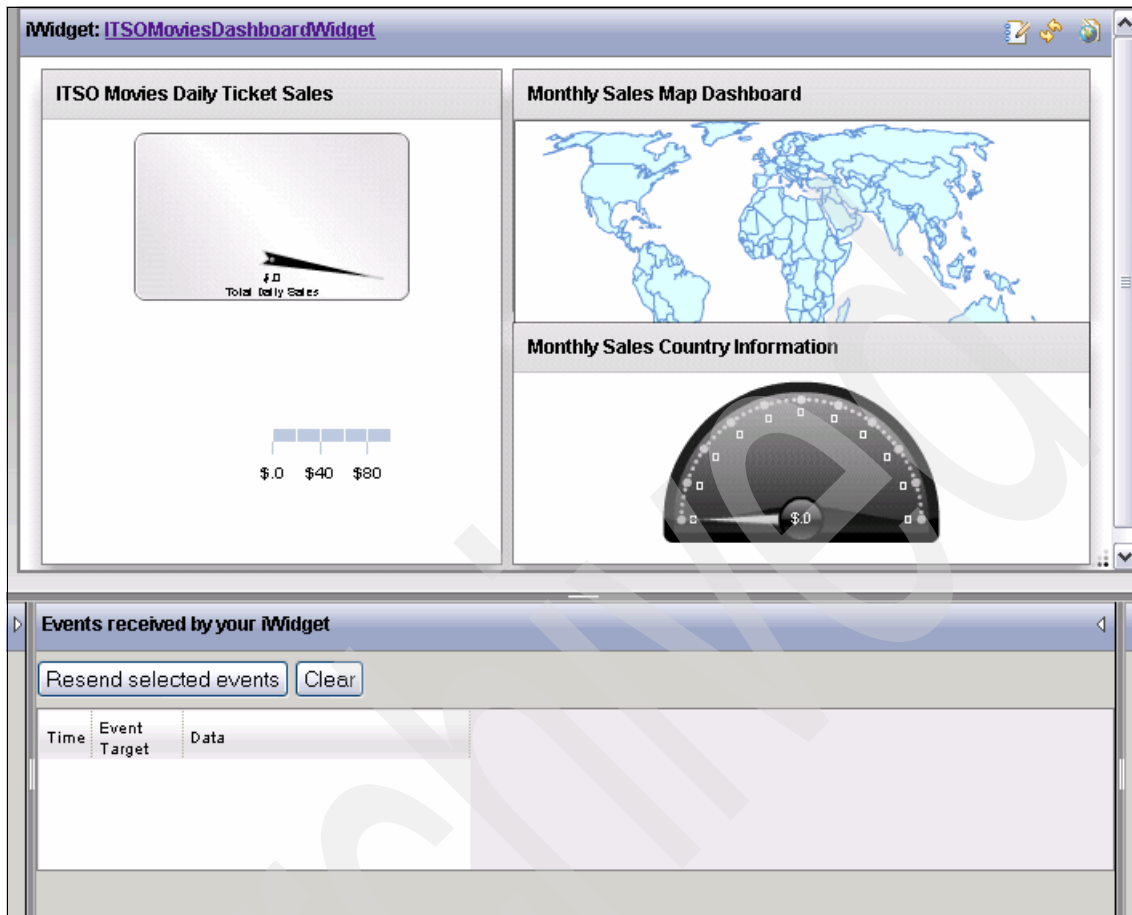


Figure 5-17 Universal Test Client for iWidget

Note: In the Universal Test Client, the Flex application is not loaded from the Business Space Environment itself. That is why you will not see the KPIs in the Elixir components. However, you can test that the iWidget is loaded properly.

5.6.5 Creating an edit mode for the iWidget

One of the useful features of the iWidget system is the ability to define parameters for the widget. In our scenario, we will show how to add a new parameter that controls the number of seconds between two refreshes of the dashboard. In chapter 5.4.4, "Displaying the KPI in Elixir components" on

page 195, this period has been hard-coded to 60 seconds in the Adobe Flex code. We will transform this refresh rate into a parameter of the widget so that the user of our widget will be able to change it.

Edit mode in the XML definition file

First of all, we want to modify the iWidget definition file and define the new parameter using the following steps:

1. Open the iWidget definition file `ITSOMoviesDashboadWidget.xml` in WID.
2. In the iWidget Editor click **ITSOMoviesDashboadWidget**.
3. Click **Add** and select **Item Set** in the Add Item window.
4. Click the new **Item Set** and type attributes for the Id field.
5. With the Item Set selected, click **Add** again.
6. Select **item** in the Add Item window.
7. Type `refreshRate` for the Id field.
8. Type `false` for the Read only field.
9. Type `60` for the Value.

This creates a `refreshRate` attribute with default value of 60 seconds.

The iWidget definition file should also contain the HTML fragment that will be used when the widget is in editing mode. This HTML fragment will represent the editing window for our iWidget.

To do so, in the iWidget Editor:

1. Click **Contents (Edit)**
2. Enter the HTML of Example 5-22

Example 5-22 HTML for iWidget editing window

```
<div id="_IWID_ITSOMoviesDashboardWidgetEditHolder">
  <div style="padding:10px;width:250px;overflow:auto;">
    <span style="font-size:1.2em">Refresh Rate (seconds):</span>
    <input id="_IWID_refreshRate"/>
  </div>
  <div style="padding:10px">
    <input type="button" value="OK"
      onclick="iContext.iScope().save()"
      style="margin-right: 13px;"/>
    <input type="button" value='Cancel'
      onclick="iContext.iScope().cancel();"/>
  </div>
```

</div>

This HTML fragment, displayed in Figure 5-18, defines:

- ▶ An HTML input field to enter the refresh rate.
- ▶ A **Save** button that calls a save method of the iWidget.
- ▶ A **Cancel** button that calls a cancel method of the iWidget.

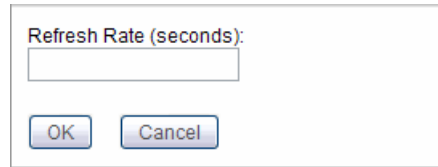


Figure 5-18 iWidget editing window

The XML definition file is now complete. The final XML file is shown in Example 5-23.

Example 5-23 ITSOMoviesDashboardWidget.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<iw:iwidget id="ITSOMoviesDashboardWidget"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns:iw="http://www.ibm.com/xmlns/prod/iWidget"
  supportedModes="view edit"
  mode="view" lang="en"
  iScope="itsomovies.ITSOMoviesDashboardWidget">

  <iw:itemSet id="attributes">

    <iw:item id="refreshRate" readOnly="false" value="60" />
  </iw:itemSet>

  <iw:resource src="ITSOMoviesDashboardWidget.js" id="jsId" />
  <iw:resource src="elixir.js" id="elixirId" />

  <iw:content mode="view">
<![CDATA[<div id="_IWID_ITSOMoviesDashboardWidgetViewHolder"></div>]]>
  </iw:content>

  <iw:content mode="edit"><![CDATA[<div
id="_IWID_ITSOMoviesDashboardWidgetEditHolder">
  <div style="padding:10px;width:250px;overflow:auto;">
    <span style="font-size:1.2em">Refresh Rate (seconds):</span>
```



```

        <input id="_IWID_refreshRate"/>
    </div>
    <div style="padding:10px">
        <input type="button" value="OK"
            onclick="iContext.iScope().save()"
            style="margin-right: 13px;"/>
        <input type="button" value='Cancel'
            onclick="iContext.iScope().cancel();" />
    </div>
</div>]]>
    </iw:content>
</iw:iwidget>

```

Implementation of the refresh rate in the Flex dashboard

In 5.4.4, “Displaying the KPI in Elixir components” on page 195, we have defined the `initializeDashboard` method that starts the timer used to periodically refresh the dashboard. We want to modify this method so that the refresh rate is no longer hard-coded. We also need to define a new `setRefreshRate` method in the Flex code to be able to change the refresh rate.

Example 5-24 shows the new `initializeDashboard` method and the `setRefreshRate` method in the `ITSOMoviesDashboard.mxml` file.

Example 5-24 initializeDashboard and setRefreshRate method

```

protected function initializeDashboard():void
{
    ExternalInterface.addCallback("setRefreshRate", setRefreshRate);

    var refreshRate:Number =
        ExternalInterface.call("ibm_ilogvisu_redbook.invokeWidget",
            FlexGlobals.topLevelApplication.parameters.widgetId,
            "getRefreshRate");

    if (isNaN(refreshRate)) refreshRate = 60;

    refreshKPIs();

    timer = new Timer(Math.max(20000, refreshRate*1000));
    timer.addEventListener(TimerEvent.TIMER, function (e:Event) : void {
        refreshKPIs();
    });
    timer.start();
}

```

```
public function setRefreshRate(refreshRate:Number) : void {
    if (timer) timer.delay = Math.max(20000, refreshRate*1000);
}
```

In this code, the `ExternalInterface.addCallback` method is used to allow the `setRefreshRate` Flex method to be called from JavaScript.

The refresh rate is retrieved by calling a JavaScript method called `getRefreshRate` through the Flex-JavaScript mediator that we created in 5.6.1, “Creating the iWidget definition” on page 212. We will define this new JavaScript method in the iWidget JavaScript implementation.

Implementation of the edit mode in JavaScript

When the user edits the settings of the iWidget, we want the editing window to show the actual value of the refresh rate. This is done by implementing the `onEdit` method in the iWidget Dojo class (`ITSOMoviesDashboardWidget.js`) as shown in Example 5-25.

Example 5-25 onEdit and getRefreshRate methods

```
getRefreshRate:function(){
    var att = this.iContext.getiWidgetAttributes();
    return att.getItemValue("refreshRate");
},

onEdit:function() {
    var widgetId = this.iContext.widgetId;

    var refreshRate = this.getRefreshRate();
    var refreshRateNode
        = this.iContext.getElementById("_" + widgetId + "_refreshRate");
    refreshRateNode.value = refreshRate;
},
```

The `getRefreshRate` function retrieves the refresh rate attribute from the iWidget. The `onEdit` function is called when the iWidget is in edit mode and we fill the HTML refresh rate input field of the editing window with the actual refresh rate value.

The last step consists in the implementation of the `save` and `cancel` methods that are called from the window buttons.

In Example 5-26 on page 229, the `save` method stores the new refresh rate in the iWidget attribute and calls the `setRefreshRate` method on the Flex side to

change the refresh rate before switching back to the view mode. The cancel method simply goes back to the view mode without changing anything.

Example 5-26 Save and cancel methods

```
save:function() {
    var widgetId = this.iContext.widgetId;
    var refreshRateNode
      = this.iContext.getElementById("_" + widgetId + "_refreshRate");
    var att = this.iContext.getiWidgetAttributes();
    att.setItemValue("refreshRate", refreshRateNode.value);
    att.save();

    if (dojo.isIE) {
        window[this.flexId].setRefreshRate(refreshRateNode.value);
    } else {
        document[this.flexId].setRefreshRate(refreshRateNode.value);
    }
    this.iContext.iEvents.fireEvent("onModeChanged", null,
                                    {newMode : "view"});
},

cancel:function() {
    this.iContext.iEvents.fireEvent("onModeChanged", null,
                                    {newMode : "view"});
},
```

We now have an improved iWidget with a refresh rate parameter. The iWidget can be tested again through the Universal Test Client for iWidget.

5.6.6 Registering the iWidget in Business Space

In order to be able to use the widget inside Business Space, we must first register the new widget in the Business Space widget catalog.

Registering the widget in Business Space consists in creating an archive file in compressed format that contains the enterprise archive file (EAR file) of the widget, and additional files that describe the widget itself.

We have to provide an endpoints file that describes the endpoints used by the widget and a catalog file that contains information about the widget and widget catalog.

The endpoints file characterizes the endpoint for our widget. The endpoint in the file is identified by an ID (which can be anything) and it points to the URL where the widget is installed.

The catalog file contains information such as the definition of a new category for the Business Space widget palette and the URI of the widgets in this new category.

Our compressed archive file must have the following directory structure:

```
ear\<WIDGET.EAR>
catalog\<WIDGETCATALOG.XML>
endpoints\<ENDPOINT.XML>
```

After our compressed archive is created, we will use the `installBusinessSpaceWidgets` administration task of the server to register our widget.

Creating the compressed archive

The compressed deployment file can be created in WID through a new general project.

Note: Here we show the steps to create the compressed archive file in WID. You can also create the folder structure outside WID and use any archive tool to create the compressed file.

To create the project follow these steps:

1. In WID, click **File** → **New Project**.
2. In the New Project window, find the **General** category and open it.
3. Under the General category, click **Project**.
4. Click **Next**.
5. Type `ITSOWidgetRegistration` as the project name.
6. Click **Finish**.

We can now create the three directories necessary for the registration package:

1. Right-click **ITSOWidgetRegistration** and select **New** → **Folder**.
2. In the New Folder window enter `ear` for the Folder name.
3. Click **Finish**.
4. Repeat these steps to create the `endpoints` and `catalog` folders.

Creating the endpoints file

Under the endpoints folder, we can create the endpoints file with the following steps:

1. Right-click the endpoints folder we have created in WID and select **New** → **File**.
2. Type `ITSOMoviesDashboardWidgetEndPoints.xml` as the file name.
3. Fill the new file with the XML content shown in Example 5-27.

Example 5-27 ITSOMoviesDashboardWidgetEndPoints.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- START NON-TRANSLATABLE -->
<tns:BusinessSpaceRegistry
xmlns:tns="http://com.ibm.bspace/BusinessSpaceRegistry"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://com.ibm.bspace/BusinessSpaceRegistry
BusinessSpaceRegistry.xsd ">

    <tns:Endpoint>
        <tns:id>{itsomovie}itsomoviesDashboard</tns:id>
        <tns:type>{itsomovie}itsomoviesDashboard</tns:type>
        <tns:version>1.0.0.0</tns:version>
        <tns:url>/ITSOMoviesDashboardWidget</tns:url>
        <tns:name>Location of our dashboard widget
    </tns:name>
    <tns:description>Location of dashboard widgets
    </tns:description>
    </tns:Endpoint>

</tns:BusinessSpaceRegistry>
<!-- END NON-TRANSLATABLE -->
```

ITSOMoviesDashboardWidget is the name of our iWidget project in WID, and it is also the root location of the deployment of the widget.

This endpoint file defines `{itsomovie}itsomoviesDashboard` as the id for pointing to the root of the deployed widget. We will use this endpoint to refer to this directory in the catalog file.

Creating the catalog file

Inside the catalog folder, we create the catalog file with the following steps:

1. Right-click the **catalog** folder in WID and choose **New** → **File**.
2. Type `catalog_ITSOMoviesDashboardWidget.xml` as the file name.

3. Fill the new file with the XML content shown in Example 5-28.

Example 5-28 catalog_ITSOMoviesDashboardWidget.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<catalog id="ITSOMoviesDashboard_ID">
  <category name="ITSOMoviesDashboard">
    <title>
      <nls-string lang="en">ITSO Movies Dashboards </nls-string>
    </title>
    <description>
      <nls-string lang="en">Dashboards for ITSO
Movies</nls-string>
    </description>
    <entry id="ITSOMoviesDashboardWidget"
      unique-name="ITSOMoviesDashboardWidget">
      <title>
        <nls-string lang="en">ITSO Movies Dashboard</nls-string>
      </title>
      <description>
        <nls-string lang="en">ITSO Elixir Dashboard to Monitor
Ticket Sales</nls-string>
      </description>
      <shortDescription>
        <nls-string lang="en">ITSO Elixir Dashboard to Monitor
Ticket Sales</nls-string>
      </shortDescription>

      <definition>endpoint://{itsomovie}itsomoviesDashboard/DashboardWidge
t/ITSOMoviesDashboardWidget.xml</definition>

      <preview>endpoint://{itsomovie}itsomoviesDashboard/DashboardWidget/i
mages/ITSOMoviesDashboardWidgetPreview.gif</preview>

      <icon>endpoint://{itsomovie}itsomoviesDashboard/DashboardWidget/imag
es/ITSOMoviesDashboardWidget.gif</icon>

      <previewThumbnail>endpoint://{itsomovie}itsomoviesDashboard/Dashboar
dWidget/images/ITSOMoviesDashboardWidget.gif</previewThumbnail>
    </entry>
  </category>
</catalog>
```

In this file, we can find the following information:

- ▶ The <category> tag defines a new category for the Business Space palette. The title of this category is 'ITSO Movies Dashboards' as indicated by the <title> tag.
- ▶ The category has only one widget entry defined by the <entry> tag. This entry is our new widget with the title 'ITSO Movies Dashboard.'
- ▶ The <definition> tag gives the location of the widget XML definition file.
- ▶ The <preview>, <icon>, and <previewThumbnail> tags point to the images that we have stored in the DashboardWidget/images directory.

Creating the EAR file

The .ear folder of the compressed file must contain the Enterprise Archive (EAR) file of the widget. To create the EAR file, we go back to our widget project in WID.

1. Right-click the **ITSOMoviesDashboardWidgetEAR** project in the Enterprise Explorer view in WID and select **Export** → **EAR file**.
2. In the Export window shown in Figure 5-19, for the Destination entry, locate the destination ear folder of our ITSOWidgetRegistration project.
3. Click **Finish**.

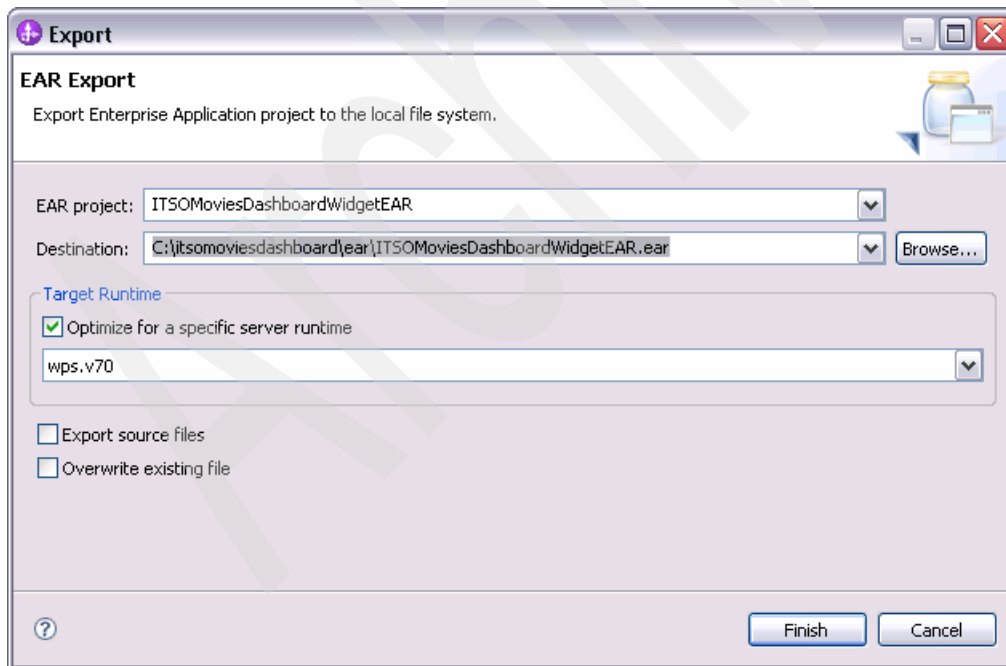


Figure 5-19 EAR Export window

Exporting to compressed format

Now that all the elements are in the folders, the compressed file can be created through the following steps:

1. Right-click the **ITSOWidgetRegistration** and select **Export**.
2. In the Export window, select **General**, then **Archive File**. Click **Next**.
3. In the Export window for archives, enter the name and location of the compressed archive in the “To archive file:” field.
4. Select only the ear, catalog, and endpoints directories. Make sure the .project file and the ITSOWidgetRegistration folder are not selected.
5. For the options, select **Save in zip format** and **Create only selected directories**.
6. Click **Finish**.

Note: At this point, you might want to open the resulting compressed file to ensure it has the correct directory structure.

Registering the widget

Now that the archive file is created, the widget can be installed using the `installBusinessSpaceWidgets` administration task:

1. From a console window, run `wsadmin.bat`, which is located under `<PROFILE_ROOT>/bin`:

`wsadmin -user <USER_NAME> -password <PASSWORD>`
2. Make sure that the server you will run the command against is up and running.

If you are installing to a non-clustered environment, issue the following command:

```
wsadmin>$AdminTask installBusinessSpaceWidgets {-nodeName  
<NODE_NAME> -serverName <SERVER_NAME> -widgets  
<PATH_TO_WIDGET_ZIP_PACKAGE>}
```

If you are installing to a clustered environment, issue the following command:

```
wsadmin>$AdminTask installBusinessSpaceWidgets {-clusterName  
<CLUSTER_NAME> -widgets <PATH_TO_WIDGET_ZIP_PACKAGE>}
```

`<NODE_NAME>` is the name of the node hosting Business Space, `<SERVER_NAME>` is the name of the server hosting Business Space, `<CLUSTER_NAME>` is the name of the cluster hosting Business Space, and `<PATH_TO_WIDGET_ZIP_PACKAGE>` points to the compressed file that we have just created.

3. Restart the server.

To verify the installation of the widget in Business Space:

1. Log on to Business Space.
2. Click **Edit Page** on the current page.
3. Select ITSO Movies Dashboards(1) in the left drop-down menu.
4. You should see the registered widget as in Figure 5-20.



Figure 5-20 Registered widget

5.6.7 Using the widget in Business Space

If you have not created a business space yet, you can follow the steps described in 5.5.2, “Creating the Web Site widget” on page 209 for creating the business space and the associated page. This will be necessary to use the new dashboard widget in Business Space. Instead of using the Web Site widget, we will use our new “ITSO Movies Dashboard” widget.

Figure 5-21 on page 236 shows the new widget in the Business Monitor Palette and in a page.

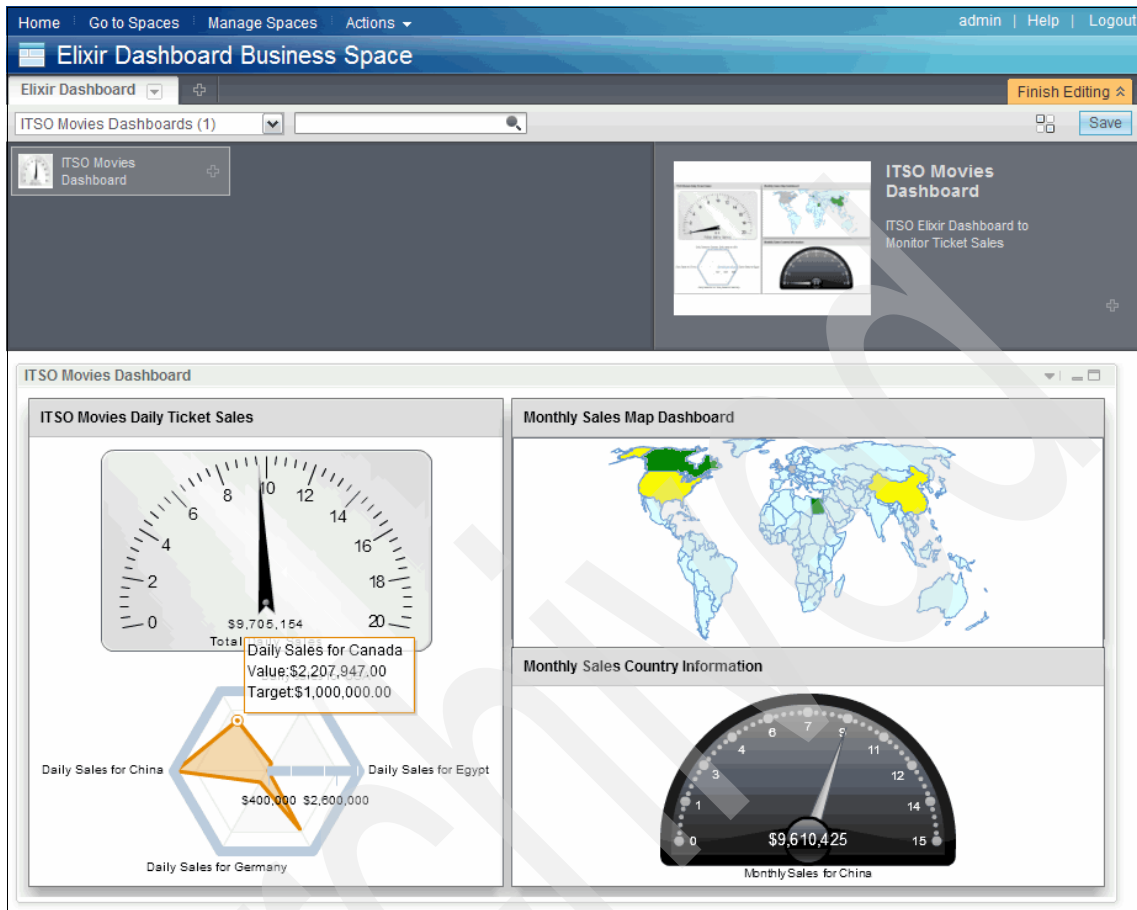


Figure 5-21 The ITSO Movies Dashboard widget in Business Space

To change the refresh rate of the dashboard, in the Edit Page mode, click the arrow on the top right corner of the iWidget and choose **Edit Settings**.

The iWidget window appears and allows you change the refresh rate as shown in Figure 5-22 on page 237.

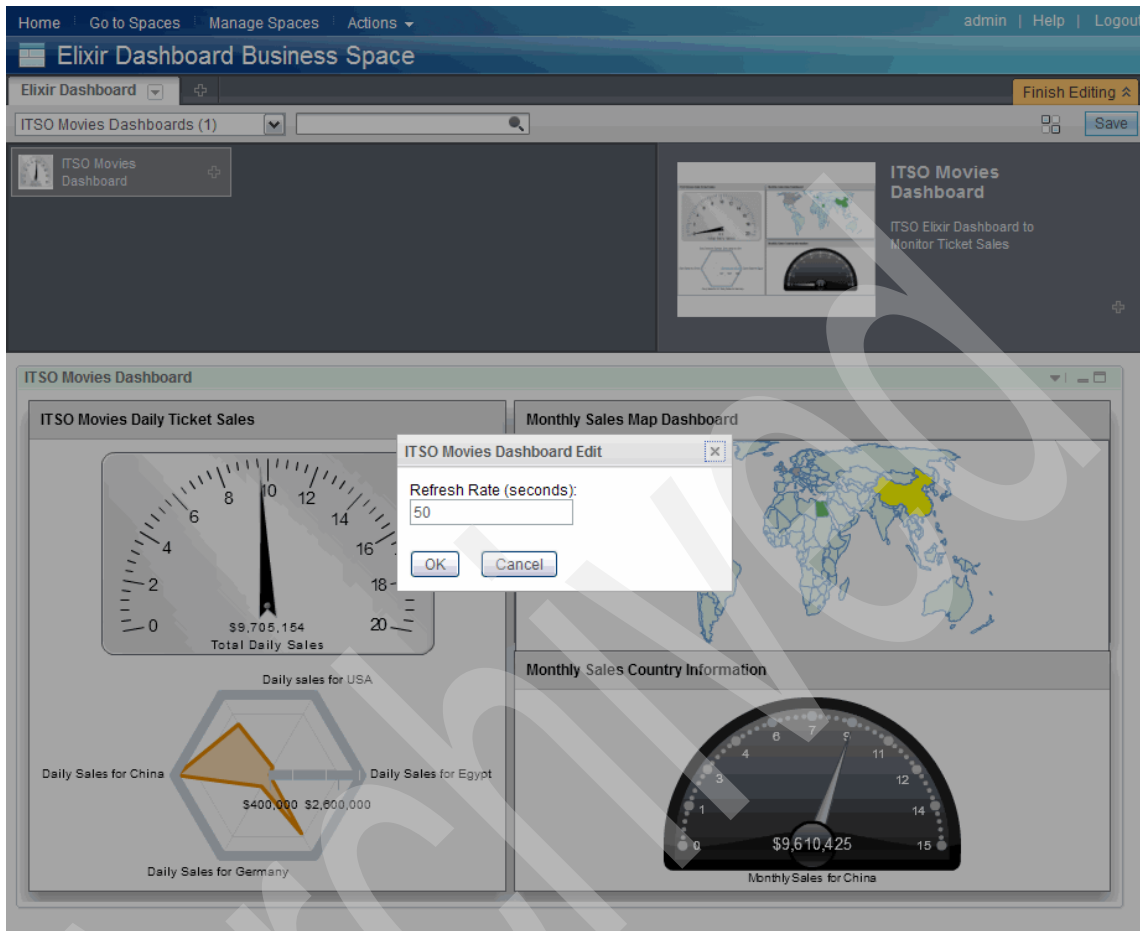


Figure 5-22 Changing the refresh rate

We have created a dashboard showing the sales KPIs of the ITSO Movies company.

This scenario is a simple example showing the integration of Business Monitor, Business Space, and ILOG Elixir Enterprise. From this example scenario, you can add more parameters to the iWidget. Elixir components have many properties that can be manipulated as parameters for iWidgets to control the style of the components such as the color of series in the radar chart and the background or text color of the gauges.

It is also possible to create more complex dashboards using Elixir components such as the Gantt component, the Trampy, or the Diagram components.

Integrating ILOG JViews JSF components in WebSphere Dashboard Framework

IBM WebSphere Dashboard Framework is a powerful and flexible tool for rapidly building dashboard applications whose data are from various information systems.

In this chapter, we discuss the integration of IBM ILOG JViews JSF components into WebSphere Dashboard Framework. We describe how WebSphere Dashboard Framework currently integrates with ILOG JViews engine and how to develop a customized ILOG JViews Gantt JSF application with WebSphere Dashboard Framework. We also describe how to deploy the dashboard application to WebSphere Portal Server or WebSphere Application Server.

6.1 Introduction to WebSphere Dashboard Framework

IBM WebSphere Dashboard Framework is a flexible tool that augments the capabilities of IBM WebSphere Portal to rapidly build service-oriented architecture (SOA) dashboards. WebSphere Dashboard Framework augments the capabilities of IBM WebSphere Portlet Factory, adding dashboard-specific features, such as a robust alerting module, high fidelity charting, flexible filtering technology, and a set of dashboard-specific design components called “builders.”

WebSphere Dashboard Framework provides the tooling that allows for the creation of dashboard applications with little programming effort. These dashboards can be built and customized quickly to suit the exacting needs of the business in a fraction of the effort usually required with traditional development techniques and still provide enough features and richness to allow the creation of powerful user interfaces.

More details about WebSphere Dashboard Framework can be found in IBM Information Center at the following URL:

<http://publib.boulder.ibm.com/infocenter/wdfhelp/v6r1m5/index.jsp>

6.2 WebSphere Dashboard Framework installation

This section discusses the installation requirements for WebSphere Dashboard Framework.

6.2.1 Software requirements

Before installing IBM WebSphere Dashboard Framework, review the software requirements. See the detailed system requirements document at the following URL:

<http://www.ibm.com/support/docview.wss?rs=477&uid=swg21419194>

6.2.2 Installing WebSphere Dashboard Framework

More detailed installation information can be found in IBM Information Center at the following URL:

<http://publib.boulder.ibm.com/infocenter/wdfhelp/v6r1m5/index.jsp>

The information is located in the section **WebSphere Dashboard Framework → Installation → Installing and configuring WebSphere Dashboard Framework**.

Figure 6-1 shows the WebSphere Dashboard Framework start menu after installation on Windows.

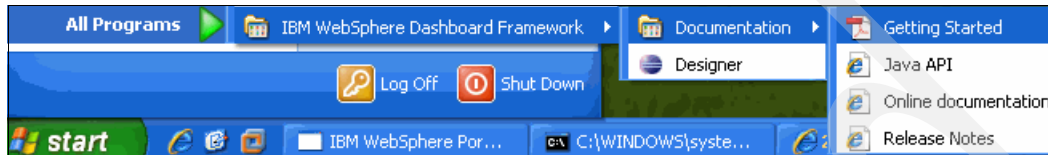


Figure 6-1 WebSphere Dashboard Framework start menu

6.3 How WebSphere Dashboard Framework integrates with ILOG JViews in the current release

The current WebSphere Dashboard Framework 6.1.5 release has been integrated into the IBM ILOG JViews Charts engine, which replaces the Greenpoint WebCharts engine imported previously. It also ships with IBM ILOG JViews Diagrammer and Maps engines for the dashboard application.

To use ILOG JViews engines, WebSphere Dashboard Framework has integrated them into WebSphere Portlet Factory builders. You can find detailed builder information in the IBM Information Center at the following URL:

<http://publib.boulder.ibm.com/infocenter/wdfhelp/v6r1m5/index.jsp>

For Charts builder, it is under the section of **WebSphere Dashboard Framework → Builder help → Charts builder**.

Figure 6-2 on page 242 shows the Charts builder.

Charts
Adds dynamic charts to a page, including area, bar, bubble, dial, line, pie, radar, scatter, and stair charts.

Properties

Name *

Page Location

Location Technique: ☒ On Named Tag ☐ Relative to Named Tag ☐ Advanced

Page *

Tag *

Chart Style and Data

Use these inputs for specifying the source of the chart style and chart data. The style is typically created using the IBM ILOG JViews Charts Designer, and come from a file in the servable content area. The data can be tabular format or IBM ILOG JViews Charts XML data source format. In general the 'RowsetRow' is the best choice for all data that comes from IBM Data Access builders.

Chart Type

Provide Custom Style ☐

Chart Data *

Data Format

Figure 6-2 WebSphere Dashboard Framework Charts builder

Map builder in the section **WebSphere Dashboard Framework** → **Builder help** → **Map builder**.

Figure 6-3 shows the Map builder.

Map
Creates a map that contains color-coded regions showing status information from dynamic sources.

Properties

Name *

Use Theme ☒

Initialization Action

Data *

Page Title

Web 2.0 support

Enable Partial Page Updates ☐

Chart Properties

Map File

Column For Region *

Column For Value

Min Property For Region *

Figure 6-3 WebSphere Dashboard Framework Map builder

6.4 How to develop a customized JViews Gantt JSF application

In this section, we detail the process of developing a customized JViews Gantt JSF application.

6.4.1 Software requirements for this section

To run the sample, the following software is required:

1. WebSphere Dashboard Framework 6.1.5
2. WebSphere Portal Server 6.1.5
3. WebSphere Application Server Network Deployment 7.0
4. ILOG JViews Enterprise 8.7

Note:

- ▶ WebSphere Dashboard Framework 6.1.5 is currently shipped with jar libraries from ILOG JViews Enterprise 8.6 patch 12. In this section, we will integrate with the latest ILOG JViews Enterprise 8.7 patch 4.
- ▶ Also upgrade WebSphere Application Server with the latest fix pack to avoid deployment issues. We use WebSphere Application Server fix pack 11 in this section.

6.4.2 Integration scenario

In our scenario, we created the WebSphere Portlet Factory project and model, used a group of builders shipped with WebSphere Portlet Factory Designer to load Gantt xml data into JViews Gantt data model, and then accessed JViews Gantt JSF component in the jsp page.

Finally, through different deployment configurations, we can deploy the project either in WebSphere Application Server or run it as a portlet in WebSphere Portal Server.

Note: We provide two sample projects in the sample code provided with this book. You can download the additional web material associated with this book (see “Locating the web material” on page 287) and find the projects under the Chapter 6 directory.

- ▶ GanttDemo is a JViews Gantt Java Project that handles reading an XML data source in WebSphere Dashboard Framework context and feeds it into a JViews Gantt data model in a JViews Gantt JSF backbean.
- ▶ WDFGanttDemo is a WebSphere Portlet Factory Project that handles all kinds of builders and accesses JViews Gantt JSF components in a JSP page.

In the next sections we provide only code fragments for detailed explanation. You can reference the complete code for the sample projects in the additional material for chapter 6.

6.4.3 Create WebSphere Portlet Factory project

1. Start WebSphere Portlet Factory from Windows Start menu as shown in Figure 6-4.

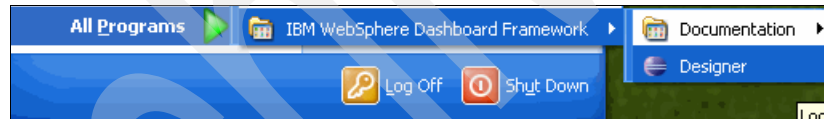


Figure 6-4 Start WebSphere Portlet Factory

2. Create a new WebSphere Portlet Factory Project by clicking **File** → **New** → **WebSphere Portlet Factory Project** in the workspace as shown in Figure 6-5 on page 245.

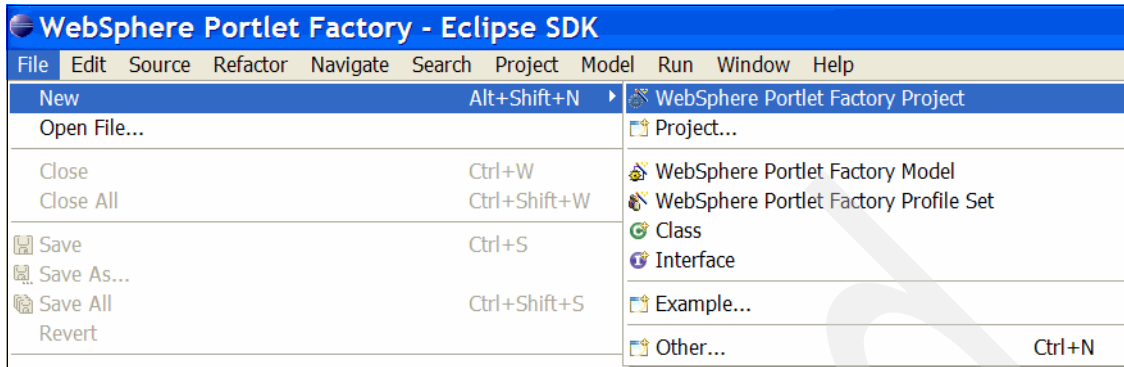


Figure 6-5 New WebSphere Portlet Factory project

3. Type your project name in the **Project Name** wizard page. In the case shown in Figure 6-6, we call the project WDFGanttDemo. Click **Next**.

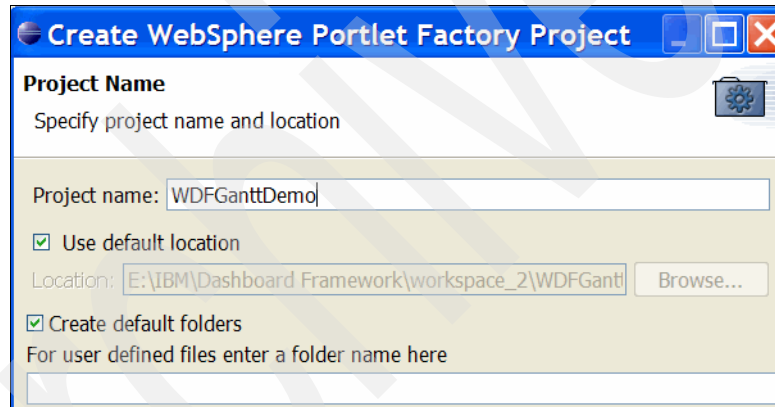


Figure 6-6 Input project name

4. In the **Feature Set page** wizard page, click **WebSphere Dashboard Framework** → **Dashboard framework** as shown in Figure 6-7 on page 246, then click **Next**.

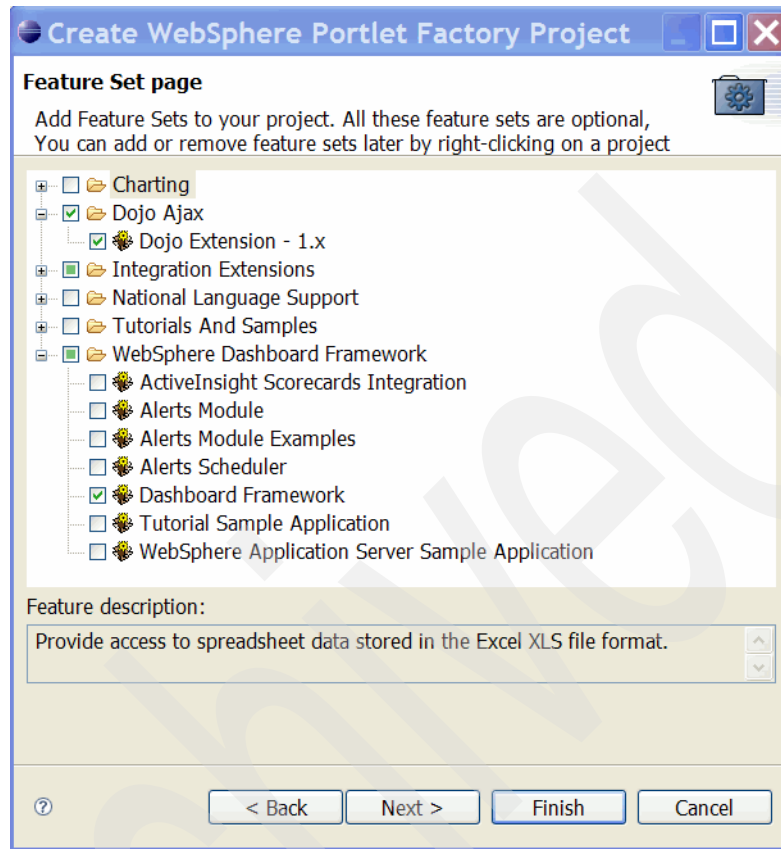


Figure 6-7 Select Dashboard Framework

5. In the **Server Configuration** wizard page, select WASCE as the default Server for publishing your application as shown in Figure 6-8, then click **Finish**.

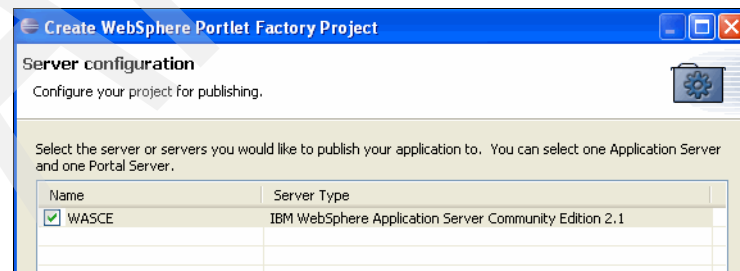


Figure 6-8 Choose WASCE as the default server

6. Click **Yes** if you need to override the existing file as shown in Figure 6-9. Your project will be created.

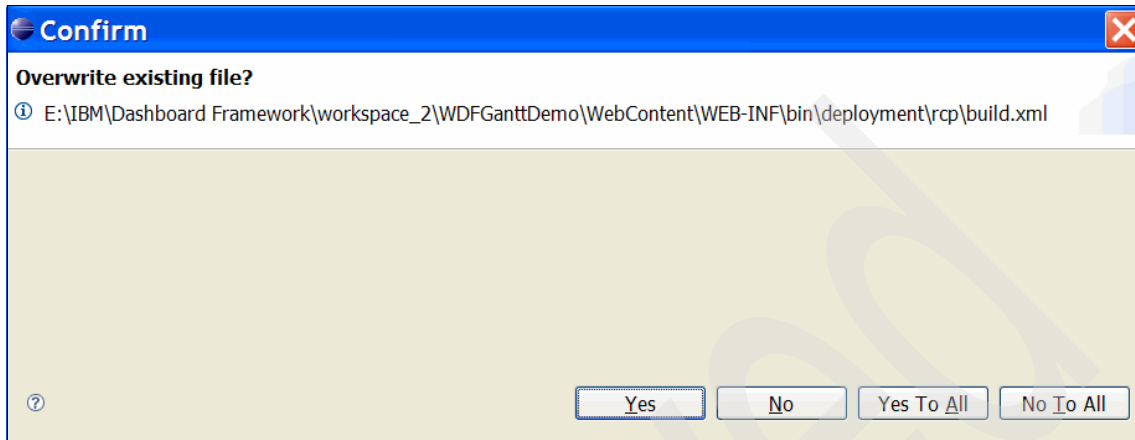


Figure 6-9 Confirmation to overwrite existing file

7. If WAS CE is stopped you will be asked whether you want to start it. Click **Yes** to start WAS CE server.
8. Add JViews Gantt Support and overwrite jviews jar as seen in the following table.

Jar name	Copy From	Copy/Overwrite To
jviews-gantt-all.jar	{JViews_Installation_Home}\jviews-gantt87\lib	{DashboardFramework_WorkSpace}\WDFGanttDemo\WebContent\WEB-INF\lib
jsf-impl-1.2_07-b03-FCS.jar jsf-api-1.2_07-b03-FCS	{JViews_Installation_Home}\jviews-framework87\lib\external	{DashboardFramework_WorkSpace}\WDFGanttDemo\WebContent\WEB-INF\clientLibs
jviews-framework-all.jar	{JViews_Installation_Home}\jviews-framework87\lib	{DashboardFramework_WorkSpace}\WDFGanttDemo\WebContent\WEB-INF\lib
jviews-chart-all.jar	{JViews_Installation_Home}\jviews-framework87\lib	{DashboardFramework_WorkSpace}\WDFGanttDemo\WebContent\WEB-INF\lib

Jar name	Copy From	Copy/Overwrite To
jviews-diagrammer-all.jar	{JViews_Installation_Home}\jviews-framework87\lib	{DashboardFramework_WorkSpace}\WDFGanttDemo\WebContent\WEB-INF\lib
jviews-maps-all.jar	{JViews_Installation_Home}\jviews-framework87\lib	{DashboardFramework_WorkSpace}\WDFGanttDemo\WebContent\WEB-INF\lib

Note: The libraries under WEB-INF\lib will be deployed into the WAR file automatically, but the libraries under WEB-INF\clientLibs will not. You can put the jars there only for compilation purpose.

9. Refresh the project in WebSphere Portlet Factory Designer to add the jar references into **Java Build Path** automatically.

6.4.4 Create JViews Gantt Java project

Next we will create a new JViews Gantt Java project by performing the following steps:

1. Create a Java Project by click **File** → **New** → **Java Project** in the workspace as shown in Figure 6-10.

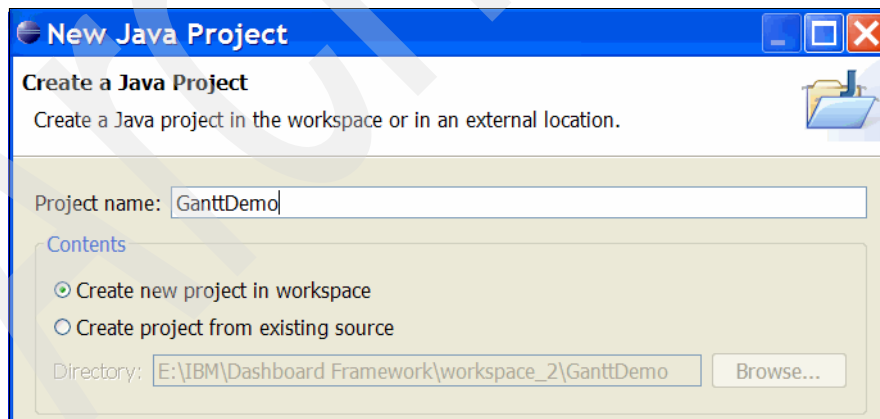


Figure 6-10 Create JViews Gantt Java Project

2. Input your project name in the **Create a Java Project** wizard page. In the case, we call the project GanttDemo. Click **Finish**.

Create Java class for Gantt JSF backbean

Under the project, create a Java class called DemoXmlBuilderBean, which will read the XML data source from WebSphere Dashboard Framework context and support Gantt interactions like selection and expansion.

1. Read the shared variable for XML data source in the constructor of DemoXMLBuilderBean class as shown in Example 6-1.

Example 6-1 Read XML data source in the constructor of DemoXmlBuilderBean class

```
public class DemoXmlBuilderBean {
    /**
     * The XML Data Source binding component.
     */
    private IlvFacesGanttXMLDataSource xmlDS = new IlvFacesGanttXMLDataSource();

    /**
     * The gantt view component binding.
     */
    private IlvFacesHierarchyChartView ganttView = new IlvFacesDHTMLGanttChartView();

    public DemoXmlBuilderBean() {
        /**
         * Get the shared variable builder for XML data source from the context
         * and feed it into Gantt model by using Gantt XML datasource
         */
        ExternalContext ec = FacesContext.getCurrentInstance()
            .getExternalContext();
        HttpSession session = (HttpSession) ec.getSession(false);
        com.bowstreet.services.base.TaggedData data =
        (com.bowstreet.services.base.TaggedData) session.getAttribute("GanttDataXML_ID");
        String xmlString=data.getValue().toString();
        IlvGanttModel model = new IlvDefaultGanttModel();

        javax.xml.parsers.DocumentBuilderFactory factory =
        javax.xml.parsers.DocumentBuilderFactory
            .newInstance();
        javax.xml.parsers.DocumentBuilder db;
        try {
            db = factory.newDocumentBuilder();
            org.xml.sax.InputSource inStream = new org.xml.sax.InputSource();
            inStream.setCharacterStream(new java.io.StringReader(xmlString));
            Document doc = db.parse(inStream);
            IlvGanttXMLDataSource xmlDs = new IlvGanttXMLDataSource();
            xmlDs.getGanttDocumentReader().readGanttModel(doc, model);
        } catch (Exception e) {
```

```

        e.printStackTrace();
    }
    IlvGanttChart gantt = new IlvGanttChart();
    gantt.setGanttModel(model);
    ganttView.setChart(gantt);
    xmlDS.setValue(model);
}
}

```

-
2. Add interactions for selection and expansion in the method of `onValueChanged` as shown in Example 6-2.

Example 6-2 Support selection and expand

```

/**
 * Value change listener called on a modification of the select one menu.
 * The activity is selected, visible (parent nodes are expanded and the view
 * is centered on this activity) and expanded.
 *
 * @param event
 *         The value change event.
 */
public void onValueChanged(ValueChangeEvent event) {
    String activityId = (String) event.getNewValue();

    if (activityId != null && !activityId.equals("default")) {

        // Find the activity form its id.
        IlvActivity activity = (IlvActivity) activityById.get(activityId);

        // Set the start and end activity date to edit.
        startActivityDate = new Date(activity.getStartTime().getTime());
        endActivityDate = new Date(activity.getEndTime().getTime());

        // Set the visible interval to the activity time interval plus 2
        // weeks.
        centerView(startActivityDate, endActivityDate);

        IlvGanttModel model = getGanttModel();
        IlvHierarchyChart chart = ganttView.getChart();

        // Collapse the previous selected activity
        IlvActivity cActivity;
        if (currentActivity != null) {
            cActivity = currentActivity;

```



```

        while (cActivity != model.getRootActivity()) {
            cActivity = model.getParentActivity(cActivity);
            chart.collapseRow(cActivity);
        }
        chart.collapseRow(currentActivity);
    }

    // Make the activity visible.
    cActivity = activity;
    while (cActivity != model.getRootActivity()) {
        cActivity = model.getParentActivity(cActivity);
        chart.expandRow(cActivity);
    }
    chart.expandRow(activity);

    // Selection update.
    chart.deselectAllRows();
    chart.selectRow(activity, true);
    currentActivity = activity;
}
}
}

```

Create Java class for contextual pop-up menu

Under the project, create another Java class called DemoFactory that implements the interface of `IlvMenuFactory` to generate dynamic pop-up menu when user right-clicks the image:

1. Implement the `createMenu` method for the interface of `IlvMenuFactory` as shown in Example 6-3.

Example 6-3 Create contextual pop-up menu

```

public class DemoFactory implements IlvMenuFactory, IlvDHTMLConstants {
    protected final static String EXPAND_MENU_MODEL_ID = "expand";

    protected final static String SCROLL_MENU_MODEL_ID = "scroll";

    protected final static String IMAGES_DIR = "/data/images/";
    /**
     * Dynamically builds a menu depending on the selected activity and the
     * current set on the client view.
     *
     * @param graphicComponent
     *         The gantt chart component.
     * @param selectedObject
     */
}

```

```

*           The selected activity when the popup menu was triggered.
* @param menuModelId
*           The menu model ID set on the current interactor of the client
*           view.
*/
public IlvMenu createMenu(Object graphicComponent, Object selectedObject,
    String menuModelId) {

    IlvHierarchyChart chart = (IlvHierarchyChart) graphicComponent;
    IlvMenu root = new IlvMenu("root");

    createInteractorItems(root, menuModelId);
    root.addChild(new IlvMenuSeparator());
    createZoomItems(root);
    createContextualActions(chart, root, selectedObject);
    createInfosSubMenu(root, selectedObject);
    return root;
}

```

2. Implement specific methods for creating menu items as described in Example 6-4.

Example 6-4 Implement specific methods for creating menu items

```

/**
 * Creates menu item to set the expand/collapse row, the select row
 * interactor or scroll interactor, depending on the current
 * interactor set on the client view.
 * @param root The root menu
 * @param menuModelId The menu model ID.
 */
protected void createInteractorItems(IlvMenu root, String
menuModelId) {
    boolean enabled = !EXPAND_MENU_MODEL_ID.equals(menuModelId);
    IlvMenuItem toggleRow = new IlvMenuItem("Expand/Collapse
Row...");
    toggleRow.setImage(IMAGES_DIR + "arrowpm.gif");
    toggleRow.addActionListener(new JavaScriptActionListener(
"expandButton.doClick()"));
    toggleRow.setEnabled(enabled);
    root.addChild(toggleRow);

    enabled = !SCROLL_MENU_MODEL_ID.equals(menuModelId);
    IlvMenuItem scroll = new IlvMenuItem("Scroll View...");
    scroll.setImage(IMAGES_DIR + "pan.gif");

```

```

        scroll.setActionListener(new JavaScriptActionListener(
            "scrollButton.doClick()"));
        scroll.setEnabled(enabled);
        root.addChild(scroll);
    }

    /**
     * Creates menu items with icons and javascript action attached to
     * zoom in, zoom out or zoom to fit.
     *
     * @param root The root menu
     */
    protected void createZoomItems(IlvMenu root) {
        IlvMenuItem zoomIn = new IlvMenuItem("Zoom In");
        zoomIn.setImage(IMAGES_DIR + "zoom.gif");
        zoomIn
            .setActionListener(new JavaScriptActionListener(
                "gantt.zoomIn()"));
        root.addChild(zoomIn);

        IlvMenuItem zoomOut = new IlvMenuItem("Zoom Out");
        zoomOut.setImage(IMAGES_DIR + "unzoom.gif");
        zoomOut.setActionListener(new JavaScriptActionListener(
            "gantt.zoomOut()"));
        root.addChild(zoomOut);

        IlvMenuItem zoomFit = new IlvMenuItem("Zoom To Fit");
        zoomFit.setImage(IMAGES_DIR + "zoomfit.gif");
        zoomFit.setActionListener(new JavaScriptActionListener(
            "gantt.zoomToFit()"));
        root.addChild(zoomFit);
    }

    /**
     * Creates a submenu filled with information on the current activity
     * selected when the popup menu was triggered.
     *
     * @param root The root menu.
     * @param selectedObject The activity selected, or null.
     */
    protected void createInfosSubMenu(IlvMenu root, Object
selectedObject) {
        if (selectedObject instanceof IlvActivity) {

```

```

        root.addChild(new IlvMenuSeparator());
        IlvActivity node = (IlvActivity) selectedObject;
        IlvMenu info = new IlvMenu("Node Properties",
            "/data/images/properties.gif");
        root.addChild(info);

        String propImg = "/data/images/property.gif";

        info.addChild(new IlvMenuItem(node.getID(), propImg));
        info.addChild(new IlvMenuItem(node.getName(), propImg));
        info.addChild(new IlvMenuItem(node.getStartTime().toString(),
            propImg));
        info
            .addChild(new IlvMenuItem(node.getEndTime().toString(),
                propImg));
    }
}

/**
 * Create a submenu with actions on the current activity
 * selected when the popup menu was triggered.
 *
 * @param chart the current gantt view
 * @param root The root menu
 * @param selectedObject The activity selected, or null.
 */
public void createContextualActions(IlvHierarchyChart chart, IlvMenu
root,
    Object selectedObject) {
    if (selectedObject instanceof IlvActivity) {
        root.addChild(new IlvMenuSeparator());

        IlvGanttModel model = chart.getGanttModel();
        IlvActivity activity = (IlvActivity) selectedObject;
        int count = model.getChildActivityCount(activity);
        if (count > 0) {
            String s = chart.isRowExpanded(activity) ? "Collapse Row"
                : "Expand Row";
            IlvMenuItem expandRow = new IlvMenuItem(s);
            expandRow.setImage(IMAGE_DIR + "arrowclick.gif");
            FacesMethodBindingActionListener l = new
FacesMethodBindingActionListener(
                IMAGE_SERVLET_CONTEXT,
                "#{ganttBean.expandCollapse}");

```

```

        expandRow.setActionListener(1);
        root.addChild(expandRow);
    }
}
}

```

Add libraries reference in the project

In order to remove compilation errors, add the proper library references to the project:

1. Right-click the **GanttDemo** project and select **Properties**.
2. Add the libraries referenced in the **Java Build Path** from WDFGanttDemo project as follows and shown in Figure 6-11.
 - factory.jar - WDFGanttDemo/WebContent/WEB-INF/lib
 - j2ee.jar - WDFGanttDemo/WebContent/WEB-INF/clientLibs
 - jst-api-1.2_07-b03-FCS.jar - WDFGanttDemo/WebContent/WEB-INF/clientLibs
 - jsf-impl-1.2_07-b03-FCS.jar - WDFGanttDemo/WebContent/WEB-INF/clientLibs
 - jviews-framework-all.jar - WDFGanttDemo/WebContent/WEB-INF/lib
 - jviews-gantt-all.jar - WDFGanttDemo/WebContent/WEB-INF/lib

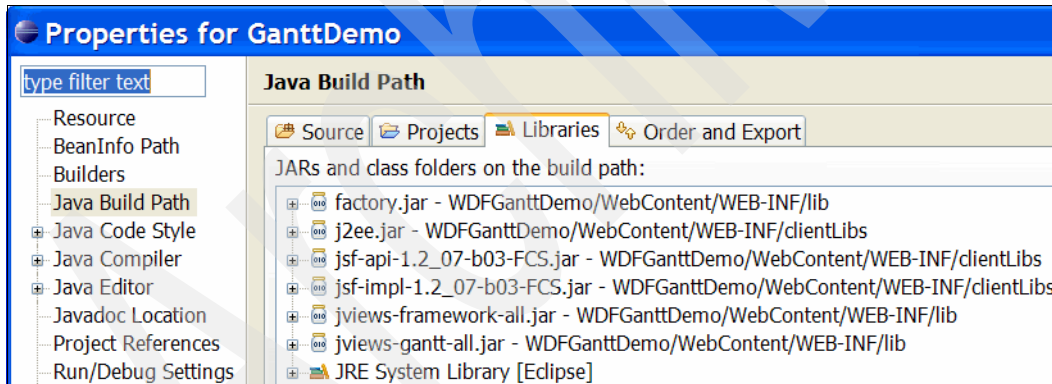


Figure 6-11 Add library reference for the project

Export the project as the jar

To use the classes in the WDFGanttDemo project, we have to export it as a JAR file and put it in the WDFGanttDemo/WebContent/WEB-INF/lib directory:

1. Right-click the **GanttDemo** project and select **Export**.
2. Select the Jar file in the wizard, then click **Next**.

3. Input the right file path for export as shown in Figure 6-12, then click **Finish**.

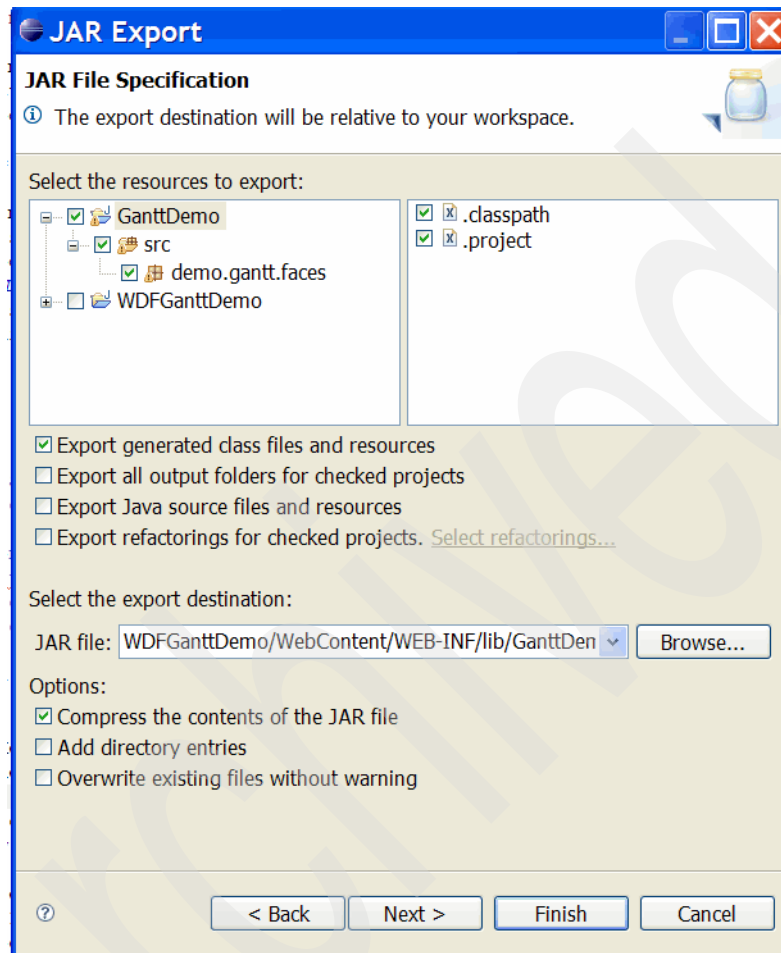


Figure 6-12 Export jar file for GanttDemo project

Note: The reason for putting java classes into GanttDemo project instead of WDFGanttDemo project directly is that all compiled java classes in WDFGanttDemo project will be put into WDFGanttDemo/WebContent/WEB-INF/work/classes. This is a special folder that can only be accessed by Portlet context method. Because we will use iFrame to access the gantt.jsp page directly, we cannot access java classes under that directory.

6.4.5 Configure WebSphere Portlet Factory Project to allow JViews Gantt support

You will need to configure the Portlet Factory Project to allow support for Gantt:

1. Declare JViews Gantt JSF component in JSF configuration.

Edit JSF configuration by double clicking the file `WDFGanttDemo\WebContent\WEB-INF\faces-config.xml` to open it in Faces Config Editor. Add the reference for the Gantt JSF component `demo.gantt.faces.DemoXmlBuilderBean` that was implemented under the GanttDemo project in “Create Java class for Gantt JSF backbean” on page 249 as shown in Example 6-5.

Example 6-5 Add Gantt JSF component

```
<managed-bean>
  <description>The java bean that contains the logic of this
sample</description>
  <managed-bean-name>ganttxmlBean</managed-bean-name>
<managed-bean-class>demo.gantt.faces.DemoXmlBuilderBean</managed-bea
n-class>
  <managed-bean-scope>session</managed-bean-scope>
</managed-bean>
```

2. Add listener class to declare the use of ILOG JViews services in `web.xml`.

Edit `web.xml` by double-clicking the file `WDFGanttDemo\WebContent\WEB-INF\web.xml` to open it in XML Editor. Add the declaration for using JViews Gantt components in the application as shown in Example 6-6.

Example 6-6 Add listener class

```
<listener>
<listener-class>ilog.views.util.servlet.DeploymentLicenseRequired_fo
r_IBM_ILOG_JViews_Gantt_Deployment</listener-class>
</listener>
```

Note: Before you can use an IBM ILOG JViews 8.7 feature, this feature must be declared through a call to the method `IlvProductUtil.DeploymentLicenseRequired` with an argument that indicates the set of features that you intend to use and for which you will therefore need a deployment license when you deploy the application. You can find more detailed information at the following URL:

<http://publib.boulder.ibm.com/infocenter/jviewgle/v8r7/index.jsp>

The information is in the following section:

IBM ILOG JViews Gantt V 8.7 → General information → Deployment licenses → Declaring the use of IBM ILOG JViews services

3. Add image and data resources to the project.

Create data folder under the WebContent directory in the WDFGanttDemo project, and copy needed images and data resources from the sample project in the directory of Chapter 6\WDFGanttDemo\WebContent\data into the folder as shown in Figure 6-13.

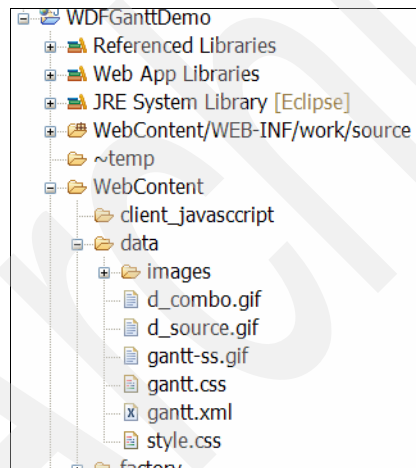


Figure 6-13 Copy resources into WebContent\data folder

4. Add Gantt JSP page to the project:
 - a. Create a pages folder under WebContent directory in the WDFGanttDemo project, and create the jsp page called `ganttt_xml_builder.jsp` under the pages folder as shown in Figure 6-14 on page 259.

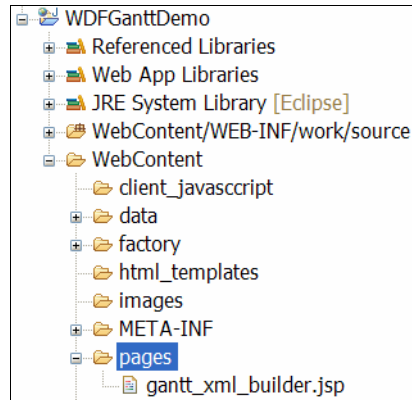


Figure 6-14 Create jsp page for holding Gantt JSF component

- b. Edit the gantt_xml_build.jsp page and add the code for accessing Gantt JSF component as shown in Example 6-7.

Example 6-7 Edit gantt_xml_build.jsp page to access Gantt JSF component

```
<f:view>
  <h:form id="form1">
    <!-- interactors -->
    <jvgf:rowExpandCollapseInteractor id="tableExpand" menuModelId="expand"/>
    <jvgf:rowExpandCollapseInteractor id="sheetExpand" menuModelId="expand" />

    <jvgf:sheetScrollInteractor id="sheetScroll" menuModelId="scroll" />
    <jvgf:tableScrollInteractor id="tableScroll" menuModelId="scroll"/>

    <jvgf:nodeSelectInteractor id="sheetNode"
invocationContext="IMAGE_SERVLET_CONTEXT"
valueChangeListener="#{ganttXmlBean.nodeValueChanged}" menuModelId="select"/>
    <jvgf:nodeSelectInteractor id="tableNode"
invocationContext="IMAGE_SERVLET_CONTEXT"
valueChangeListener="#{ganttXmlBean.nodeValueChanged}" menuModelId="select"/>

    <!-- first panel grid : the toolbar -->
    <h:panelGrid columns="20" styleClass="border" bgcolor="#DCE6E9"
      style="height:34px" columnClasses="tdc" border="0">
      ...
    </h:panelGrid>

    <!-- Second panel grid: the view and messages -->
```

```

<h:panelGrid styleClass="border" bgcolor="#DCE6E9" columns="1"border="0">

    <jvgf:ganttView id="gantt"
        binding="#{ganttXmlBean.ganttView}"
        chart="#{ganttXmlBean.ganttView.chart}"
        style="width:800px;height:400px"
        styleSheets="/data/gantt.css"
        imageFormat="PNG"
        tableInteractorId="tableExpand"
        sheetInteractorId="sheetExpand"
        messageId="messages"
        waitingImage="/data/images/ilog-anim.gif">
        <jvgf:ganttContextualMenu factory="#{ganttXmlBean.menuFactory}"/>
    </jvgf:ganttView>
    <jv:messageBox id="messages" styleClass="text"
style="width:600px;height:20px" />
</h:panelGrid>
</h:form>
</f:view>

```

Note: You can review IBM ILOG JViews Gantt product guide documentation for details about JViews Gantt JSF application development at the following URL:

<http://publib.boulder.ibm.com/infocenter/jviewgle/v8r7/index.jsp>

The information is in the following section:

IBM ILOG JViews Gantt V 8.7 → Programmer's documentation → Building Web Applications → Developing JViews Gantt JSF applications

6.4.6 Add builders in WebSphere Portlet Factory

The next step is to add the builders in WebSphere Portlet Factory:

1. Create the Gantt model in WDFGanttDemo project.
 - a. Switch to Model Navigator View by clicking **Window → Show View → Model Navigator** as shown in Figure 6-15 on page 261.

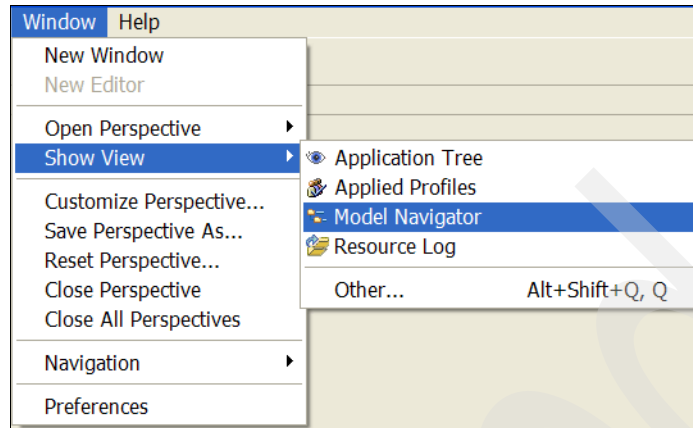


Figure 6-15 Switch to Model Navigator view

- b. Right-click the **models** package under the WDFGanttDemo project and select **New** → **WebSphere Portlet Factory Model** as shown in Figure 6-16.

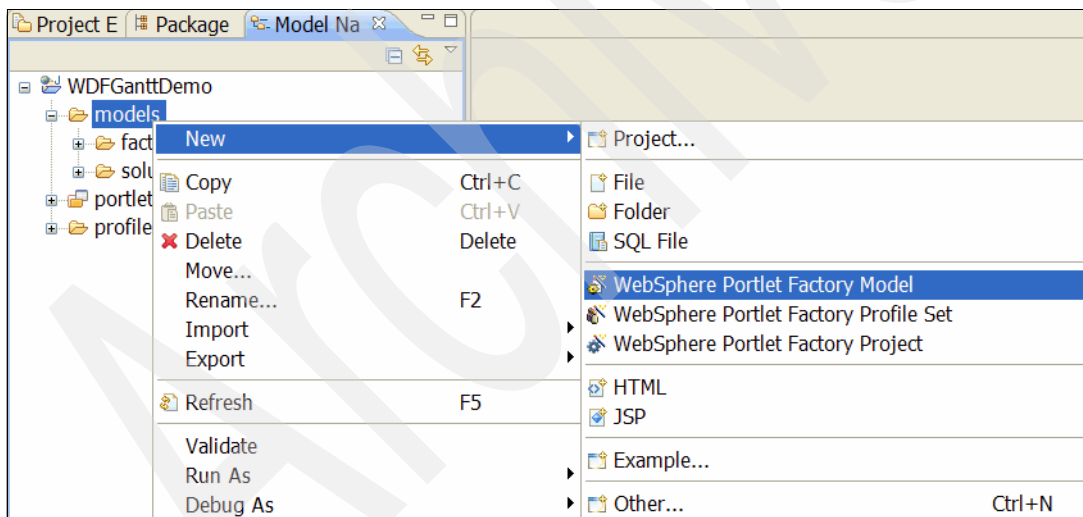


Figure 6-16 Create new WebSphere Portlet Factory Model for Gantt

- c. Select **WDFGanttDemo** in the **Choose Project** wizard page and click **Next**.
- d. On **Select Model** wizard page, select **Main and Page** under the **Factory Starter Models** tab as shown in Figure 6-17 on page 262, then click **Next**.

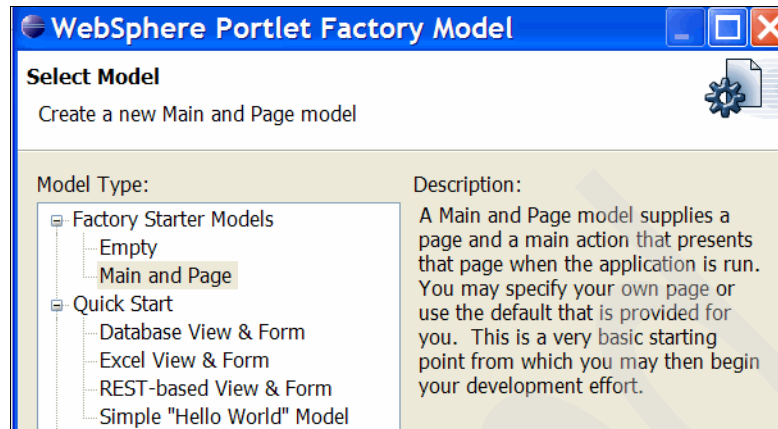


Figure 6-17 Select Main and Page as the model

- e. On the **Page Settings** wizard page, select the default setting and click **Next**.
- f. On the **Save New Model** Wizard page type `gantt` as the model name as shown in Figure 6-18 on page 263, then click **Finish**.

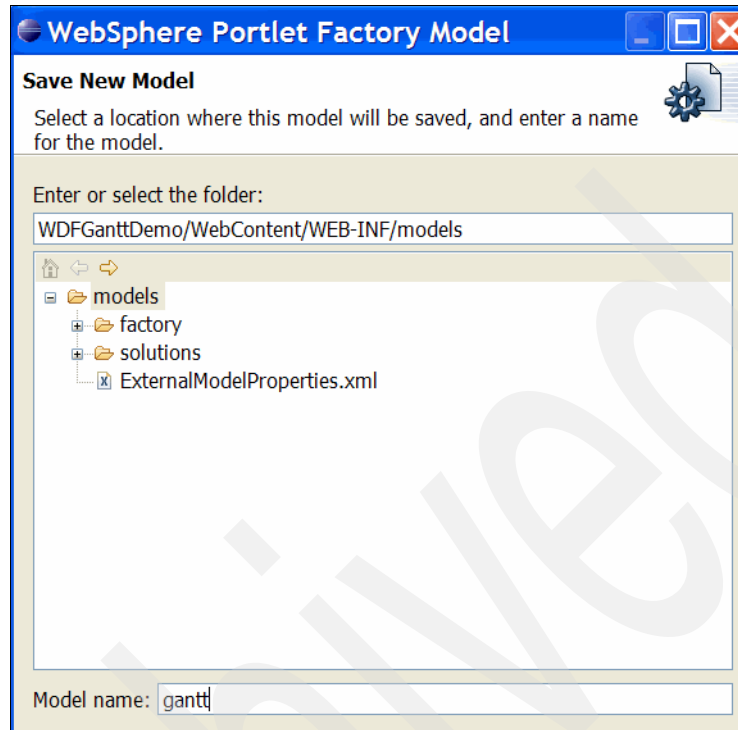


Figure 6-18 Input Model name

The new model of **gantt.model** is created under **models** package.

2. Add html builder under gantt model
 - a. In the Model Navigator window, double-click the new **gantt.model** model to open it in Model Editor.
 - b. Open the Outline view by clicking **Window** → **Show View** → **Other...** → **Outline**.
 - c. In the Outline view, click **Add a Builder** as shown in Figure 6-19.

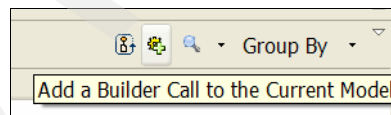


Figure 6-19 Click the button to add a builder to the current model

- d. In the Builder Picker window, select **HTML** builder as shown in Figure 6-20 on page 264, and click **OK** to open it in **Model Editor** panel.

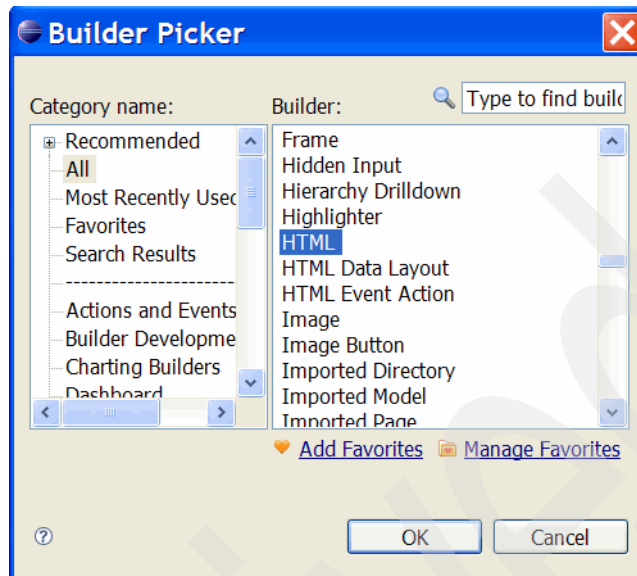


Figure 6-20 Select HTML Builder

- e. In the Model Editor window, type `htmlBuilder` in name field, and select `page1` and `myForm` for the **Page** and **Tag** fields. In the **HTML** field, input the `iframe` content shown in Example 6-8.

Example 6-8 iframe html content

```
<iframe class='wpf_wait_when_printing' id="gantChart"
name="gantChart" src='<%= request.getContextPath()
%>/faces/pages/gantt_xml_builder.jsp' allowtransparency='yes'
marginheight='0' marginwidth='0' border='0' scrolling='no'
frameborder='no' width='800' height='600'>
</iframe>
```

- f. Click **OK** to save the builder in the model as shown in Figure 6-21 on page 265.

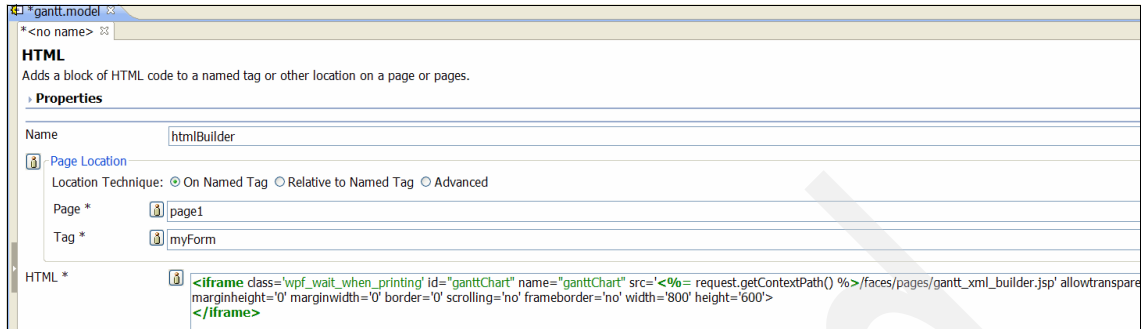


Figure 6-21 input for HTML builder

3. Add “XML” builder under gannt model in WDFGanttDemo project.
 - a. In the Outline window, click the **Add a Builder** button.
 - b. In the Builder Picker window, select **Import To XML** builder as shown in Figure 6-22, and click **OK** to open it in Model Editor window.

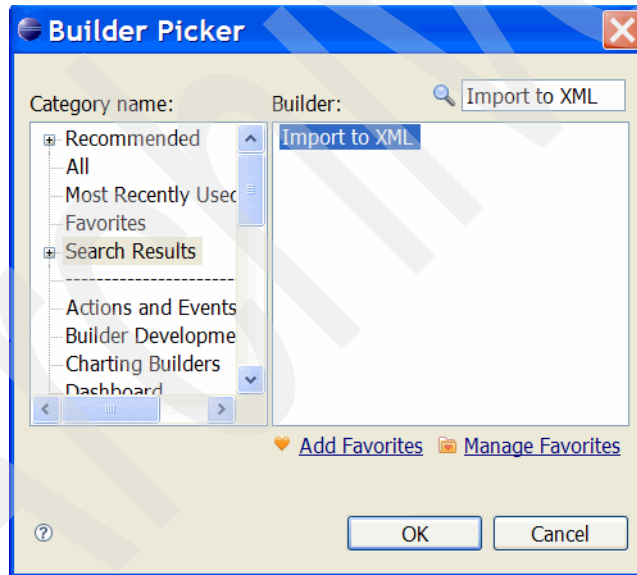


Figure 6-22 Select Import to XML builder

- c. In the Model Editor window, type GanttDataXML in name field and select gannt.xml under data directory as shown in Figure 6-23 on page 266.

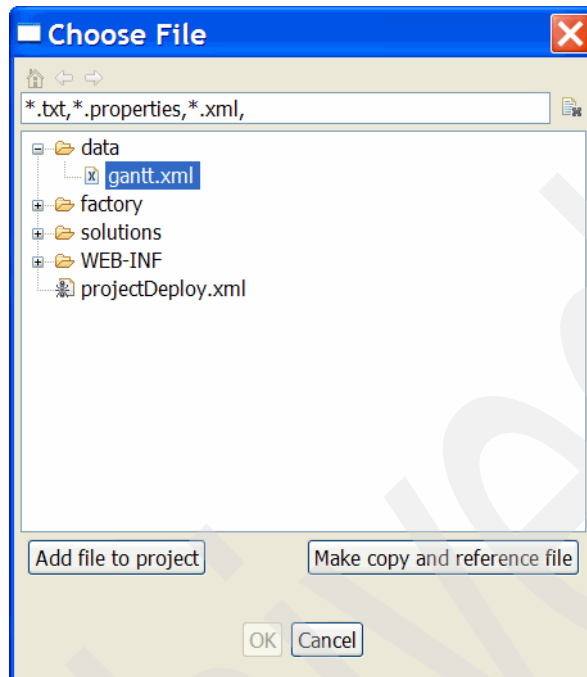


Figure 6-23 Select gantt xml file

- d. Click **OK** to save the builder in the model as shown in Figure 6-24.

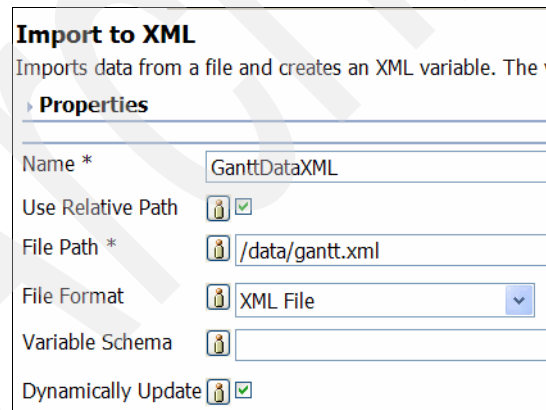


Figure 6-24 Import to XML builder

4. Add “shared variable” builder under gantt model in WDFGanttDemo project.
 - a. In Outline view, click **Add a Builder** button.

- b. In the Builder Picker window, select **Shared Variable** builder as shown in Figure 6-25, and click **OK** to open it in Model Editor window.

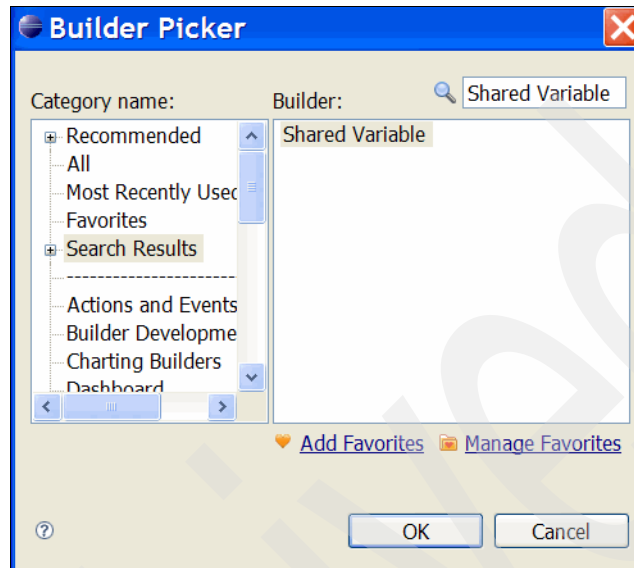


Figure 6-25 Select Shared Variable builder

- c. In **Model Editor**, type GanttDataXML_VAR in the Name field and GanttDataXML in the Variable field, select Session in the Scope field, and input GanttDataXML_ID in the Unique ID field as shown in Figure 6-26, then click **OK** to save the builder in the model.

Shared Variable	
Share a variable across a session or an application	
Properties	
Name *	GanttDataXML_VAR
Variable *	GanttDataXML
Scope *	Session
Unique ID *	GanttDataXML_ID

Figure 6-26 Input fields in Shared Variable builder

6.4.7 Debug the project in WebSphere Portlet Factory Designer

Before deploying the project into the final application server, develop and debug the project in WebSphere Portlet Factory Designer using the embedded WebSphere Application Server Community Edition (WAS CE).

1. Start WAS CE in Debug mode.

To debug a model's Java or JSP, start your server in remote debug mode. The Portlet Factory Designer will prompt you to start the WAS CE when you deploy your project and when you run a model.

- a. If you are manually starting WAS CE, add these two Java options to start the server in debug mode:
 - i. Stop the server.
 - ii. Restart using these start options either from the command line or as a script as shown in Example 6-9:

Example 6-9 Add java options when start WAS CE

```
set JAVA_OPTS=-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=8000
```

- b. If you have already deployed the application, then the file (startdebug.bat or startdebug.sh) has already been generated. You can use this file to manually start your server in debug mode as shown in Example 6-10:

Example 6-10 Start WAS CE in debug mode

```
@REM Starts wasce in debug mode.  
set JAVA_OPTS=-Xdebug -Xnoagent -Djava.compiler=NONE  
-Xrunjdpw:transport=dt_socket,server=y,suspend=n,address=8000  
startup > WPFstartDebug.log
```

2. Publish the application.

- a. In the **WebSphere Portlet Factory** window, switch to the **Package Explorer** view.
- b. Right-click the WDFGanttDemo project and select **Publish Application**.
- c. In the Publish Application window, select **WebSphere Application Server Community Edition** as shown in Figure 6-27 on page 269 and click **OK**.

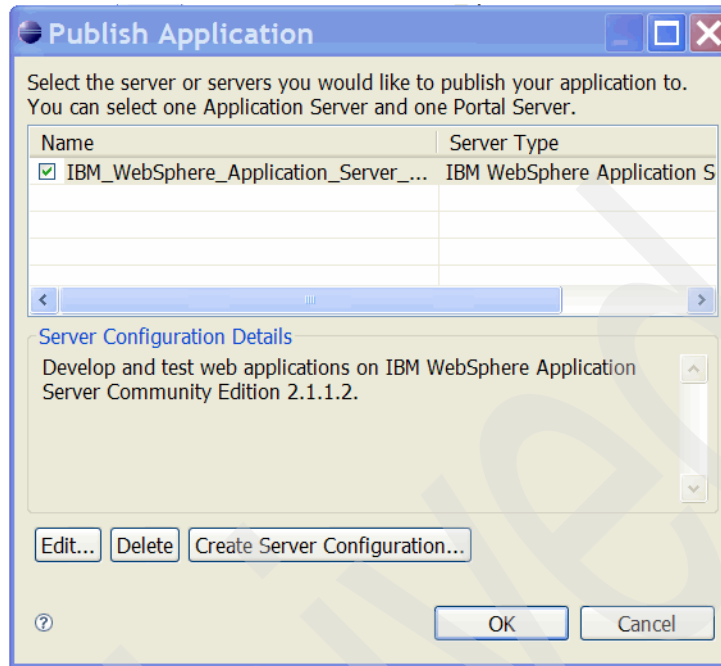


Figure 6-27 Select WebSphere Application Server Community Edition as the server

- d. In the Application Server window, select the **WAS CE in debug mode** check box, then click **Yes** to start WAS CE in debug mode and publish the application automatically.
3. Remote debug in WebSphere Portlet Factory Designer.

WebSphere Portlet Factory Designer supports eclipse remote debug. You can find more details about configurations on eclipse online documentation at the following URL:

<http://help.eclipse.org/galileo/index.jsp?topic=/org.eclipse.jdt.doc.user/concepts/cremdbug.htm>

A sample debug configuration is as shown in Figure 6-28 on page 270.

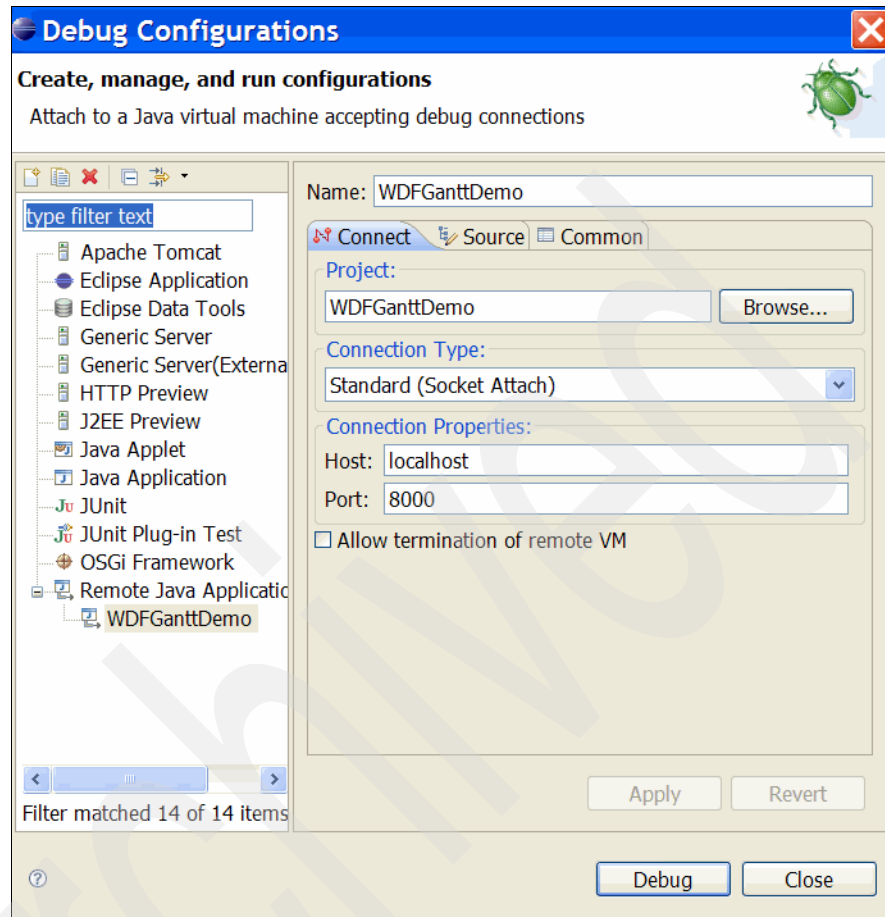


Figure 6-28 Configure Remote Debug in WebSphere Portlet Factory Designer

4. Run the Active Model in WebSphere Portlet Factory Designer:
 - a. In WebSphere Portlet Factory Perspective, switch to the view of **Model Navigator**.
 - b. Right-click WDFGanttDemo\models\gantt.model1, then select **Run As** → **Run active model** as shown in Figure 6-29 on page 271.

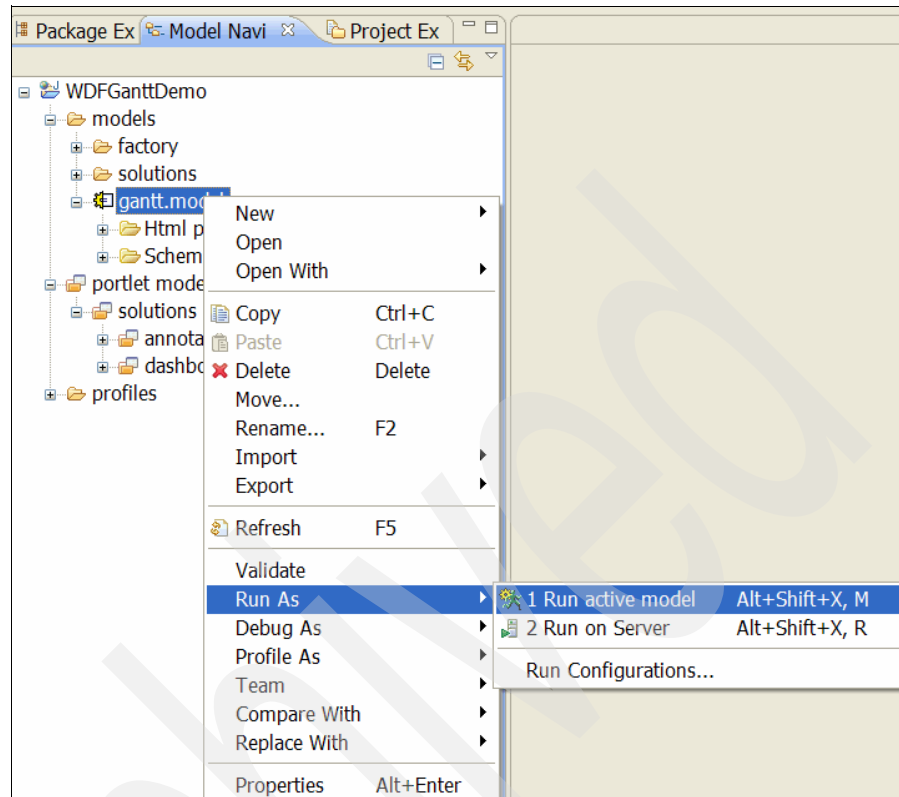


Figure 6-29 Run active model

- c. A web browser will be open and access the Gantt.jsp page automatically as shown in Figure 6-30 on page 272.

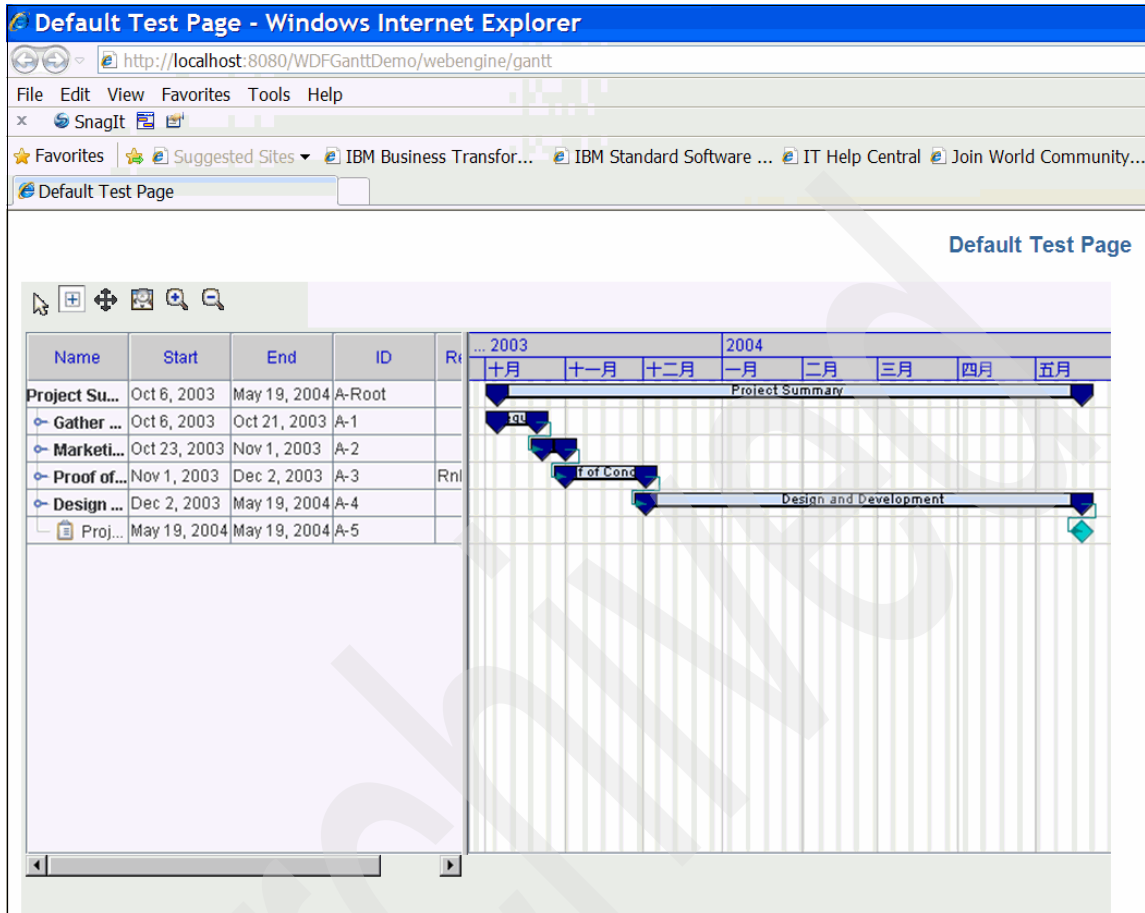


Figure 6-30 Test page when run active model in WebSphere Portlet Factory Designer

6.5 How to deploy the application

By using Export tools in WebSphere Portlet Factory Designer, the sample WebSphere Portlet Factory project created before (see the details in the 6.4, “How to develop a customized JViews Gantt JSF application” on page 242) can be deployed in either WebSphere Application Server or WebSphere Portal Server. In the following sections, we will describe how to deploy the sample application separately.

Note: In this section, we assume that user has already installed WebSphere Application Server Network Deployment 7.0 and WebSphere Portal Server 6.1.5.

6.5.1 Run the application on WebSphere Application Server

This section describes how to run the application on WebSphere Application Server. See the next section for the steps to follow to run the application on WebSphere Portal Server.

Export WebSphere Portlet Factory Application WAR

1. Export the project as a WebSphere Portlet Factory Application WAR.

In Mode Navigator window, right-click the **WDFGanttDemo** project and select **Export** → **WebSphere Portlet Factory Application WAR** as shown in Figure 6-31.

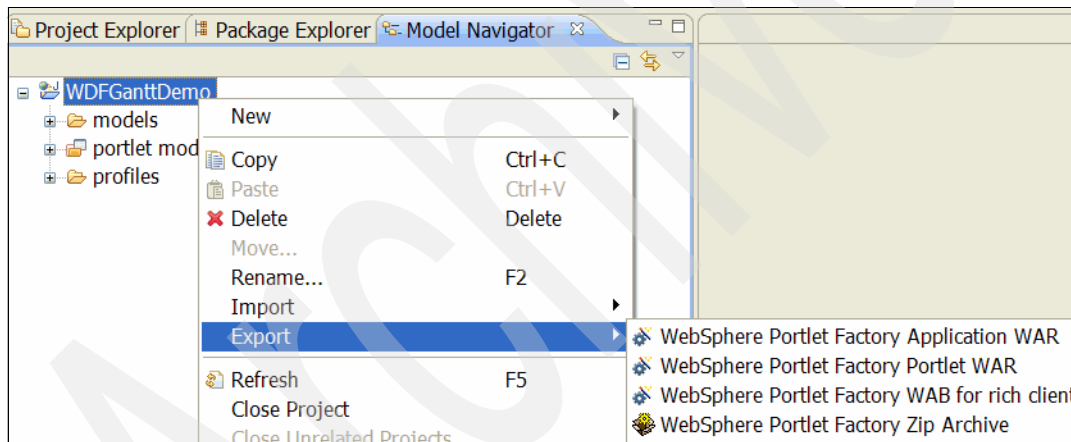


Figure 6-31 Export WebSphere Portlet Factory Application WAR

2. Specify Export Path as shown in Figure 6-32 on page 274, then click **Finish**. The WAR of WDFGanttDemo.war will be generated automatically under the path you specify.

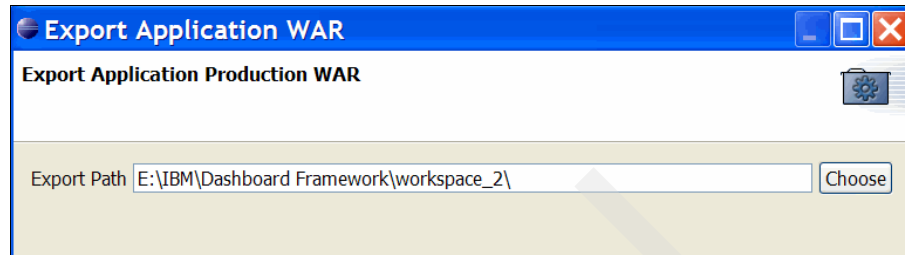


Figure 6-32 Specify Export Path

3. Copy the file WDFGanttDemo/WebContent/index.jsp as shown in Figure 6-33.

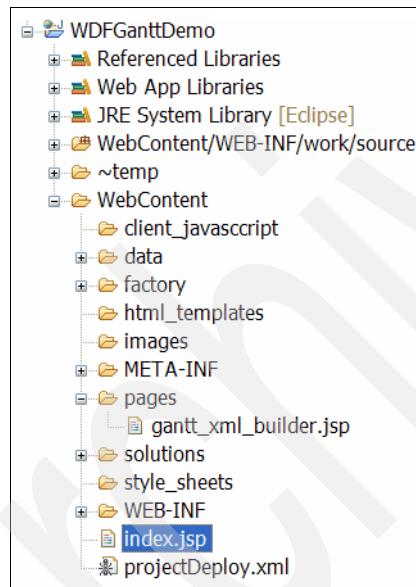


Figure 6-33 index.jsp

Copy the file into the WDFGanttDemo.war as shown in Figure 6-34 on page 275.

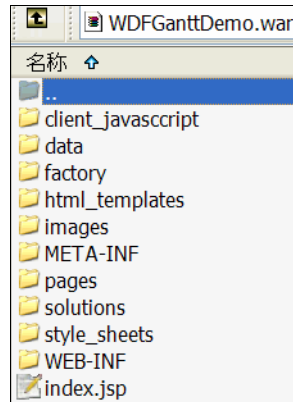


Figure 6-34 Copy *index.jsp* into the WAR

Note: We use the default *index.jsp* as our welcome file in the sample, which will list all the modules in Model Navigator window to be accessed in the web page.

Deploy the WAR into WebSphere Application Server

Next, you will need to deploy the WAR file in WebSphere Application Server:

1. Start WebSphere Application Server Network Deployment.
2. Install the sample application by using the exported WAR and start the application.

Note: More detailed information about application deployment in WebSphere Application Server 7 can be found in IBM Information Center at the following URL:

<http://publib.boulder.ibm.com/infocenter/wasinfo/v7r0/index.jsp>

Look in the section **Network Deployment (All operating systems), Version 7.0 → Developing and deploying applications.**

3. Access the demo by navigating to <http://localhost:9080/ganttDemo>. You should see the window shown in Figure 6-35 on page 276.

Available Models

The following models are contained under the \WEB-INF\models\ directory

1. Open the model – all models are located under the \WEB-INF\models\ directory
2. Choose the Run > Run... menu option
3. In the Run dialog, select a IBM Model configuration and click the Run button
4. Your model will launch in a separate browser window.

Models

[factory/adapters/CustomEditPage](#)
[factory/adapters/model_not_set](#)
[factory/adapters/not_configured](#)
[factory/core/empty](#)
[factory/core/PortletCustomizerEvents](#)
[factory/portletbase/beancrud](#)
[factory/portletbase/data_column_customizer_base](#)
[factory/portletbase/data_view](#)
[factory/portletbase/runtime_profile_selection](#)
[factory/portletbase/service_documentation_base](#)
[factory/portletbase/view_and_form_base](#)
[gantt](#)
[solutions/annotations/admin/AnnotationArchive](#)
[solutions/dashboardbase/Calendar](#)
[solutions/dashboardbase/CalendarCategory](#)
[solutions/dashboardbase/DashboardCommon](#)
[solutions/dashboardbase/data/CalendarCategoryData](#)
[solutions/dashboardbase/data/CalendarData](#)
[solutions/dashboardbase/data_column_customizer_base](#)
[solutions/dashboardbase/empty_customizer](#)
[solutions/dashboardbase/enhance_data_column_modifier_base](#)
[solutions/dashboardbase/LoggingCommon](#)
[solutions/dashboardbase/PeopleFinder](#)
[solutions/dashboardbase/QueryFilterForm](#)

Figure 6-35 Default model list page

4. Click the **gantt** link. You should see the page as shown in Figure 6-36 on page 277.

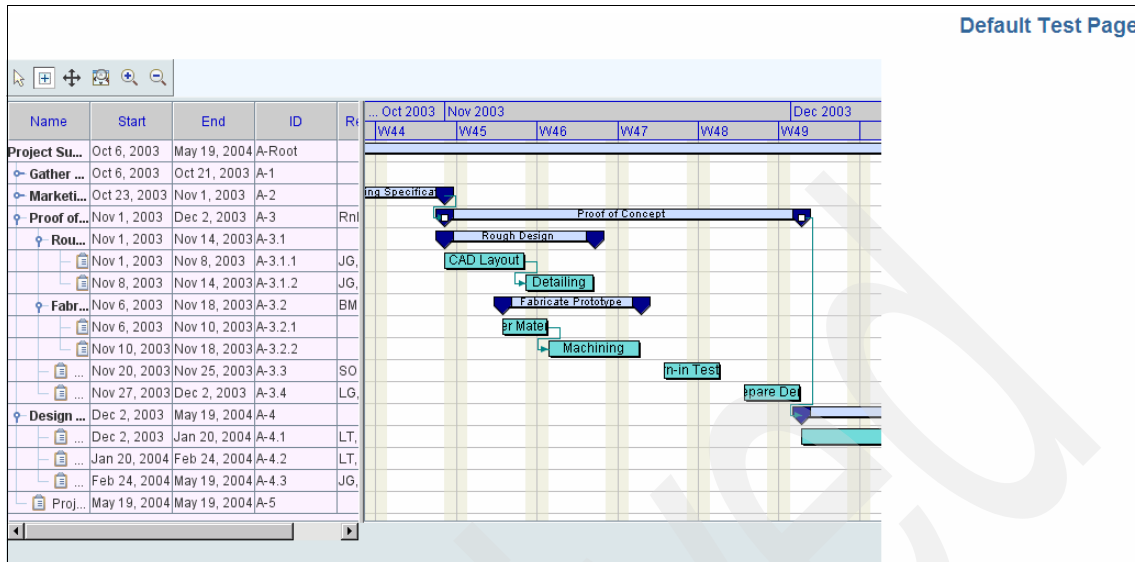


Figure 6-36 Result page running in WebSphere Application Server

6.5.2 Run the application on WebSphere Portal Server

This section describes the steps to follow to run the application on WebSphere Portal Server. See 6.5.1, “Run the application on WebSphere Application Server” on page 273 to run the application on WebSphere Application Server.

Add Portlet Builder

First you need to add Portlet Builder under the Gantt model in the WDFGanttDemo project:

1. In the Model Navigator window, double-click `gant.t.model` to open it in Model Editor.
2. Open Outline view by clicking **Window** → **Show View** → **Other...** → **Outline**.
3. In Outline view, click the **Add a Builder** button.
4. Select **Portlet Adapter** in the Builder Picker window as shown in Figure 6-37 on page 278, then click **OK** to open it in Model Editor.

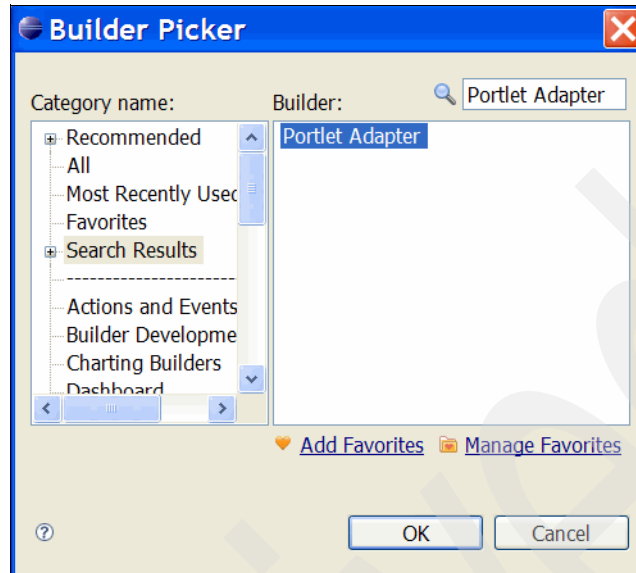


Figure 6-37 Choose Portlet Adapter builder

5. In Model Editor, type `JViewsGanttPortlet` in the Name field and type `JViews Gantt Portlet` for all other portlet properties except Default Locale, which should be `en` as shown in Figure 6-38, then click **OK** to add the builder in Gantt model and save it.


Portlet Adapter	
Allows you to expose profile values for customization v	
Properties	
Name *	JViewsGanttPortlet
Portlet Title	JViews Gantt Portlet
Portlet Short Title	JViews Gantt Portlet
Portlet Keywords	JViews Gantt Portlet
Portlet Description	JViews Gantt Portlet
Default Locale	en
User Help File	

Figure 6-38 Input Portlet properties

Export WAR

Next you need to export the WebSphere Portlet Factory Portlet WAR:

1. In the Mode Navigator window, right-click the **WDFGanttDemo** project and select **Export** → **WebSphere Portlet Factory Portlet WAR** as shown in Figure 6-39.

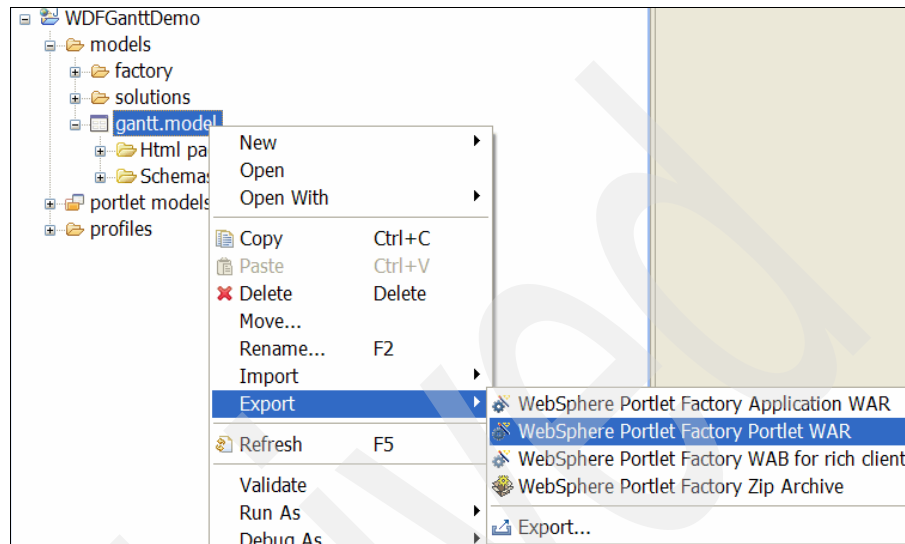


Figure 6-39 Export WebSphere Portlet Factory Portlet WAR

2. In the Export Portlet WAR window, select **Java Standard Portlet 1.0** for Portlet API and specify the **Export Path** as shown in Figure 6-40 (E:\IBM\Dashboard Framework\workspace_2\WDFGanttDemo), then click **Finish** to export the war.

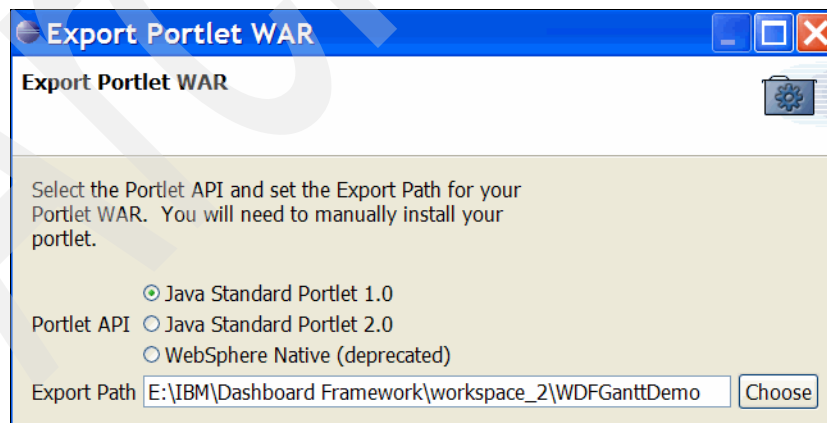


Figure 6-40 Specify export path

Note: The current ILOG JViews 8.7 is only compatible with Portlet 1.0 (JSR 168) specification. You can find more details for this in the JViews 8.7 user manual documentation at the following URL:

<http://publib.boulder.ibm.com/infocenter/jviewgle/v8r7/index.jsp>

The information is in the following section:

IBM ILOG JViews Gantt V 8.7 → Programmer's documentation → Building Web Applications → Interoperability with third party Web technologies → Running JViews JSF applications in JSR 168 portlets

3. Remove listener class in Portlet.xml:
 - a. Open the PA_WDFGanttDemo.war in the Export path specified before as shown in Figure 6-41.

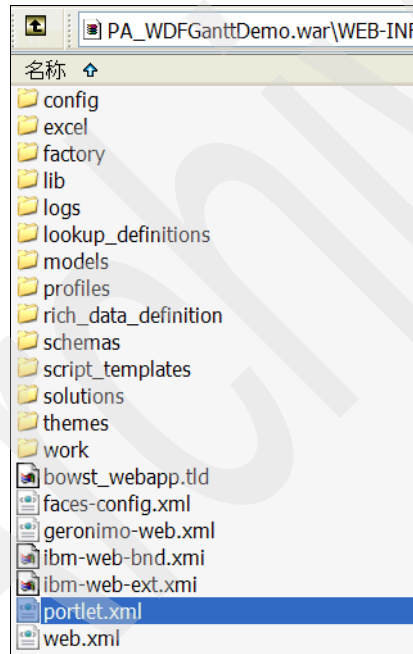


Figure 6-41 PA_WDFGanttDemo.war

- b. Edit portlet.xml under PA_WDFGanttDemo.war\WEB-INF\portlet.xml. Find the lines shown in Example 6-11 on page 281 and remove them. Then save the portlet.xml back to the war file.

Example 6-11 Remove listener class

```
<listener>
<listener-class>com.ibm.wdf.util.JSFStartupListener</listener-class>
</listener>
```

Note: The reason for removing the listener class in `portlet.xml` is to avoid `NullPointerException` when deploy the application in WebSphere Portal Server

Deploy the application to WebSphere Portal Server

These are the simple steps for application deployment into WebSphere Portal Server. More details about WebSphere Portal Server 6.1.5 can be found in IBM Information Center at the following URL:

http://publib.boulder.ibm.com/infocenter/wpdoc/v6r1/index.jsp?topic=/com.ibm.wp.ent.doc_v615/welcome_main.html

Access WebSphere Portal Server

1. Start WebSphere Portal Server 6.1.5 by clicking **Start** → **IBM WebSphere** → **Portal Server 6.1.0.3** → **Start the Server**.

Note: The startup menu is still says **Portal Server 6.1.0.3** after installing WebSphere Portal Server 6.1.5.

2. Access Portal Server administrator console by navigating to `http://your_machine_name:10040/wps/portal`, input **User ID** and **Password**, and then click **Log in**.

Deploy the application into WebSphere Portal Server

1. Click **Administration** and select **Portlet management** → **WebModules** as shown in Figure 6-42 on page 282.

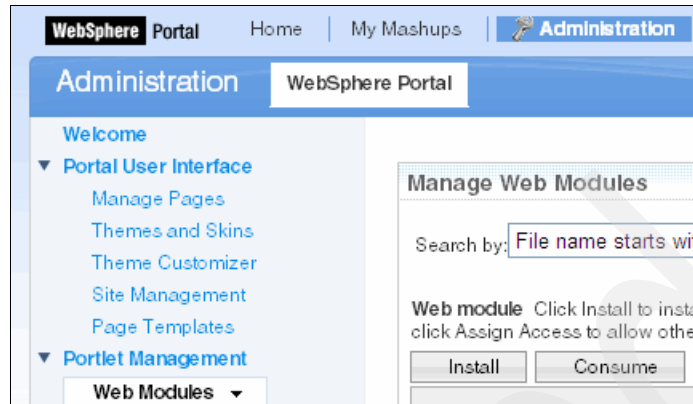


Figure 6-42 Web modules

2. Click **Install** and select your PA_WDFGanttDemo.war file as shown in Figure 6-43, then click Next.

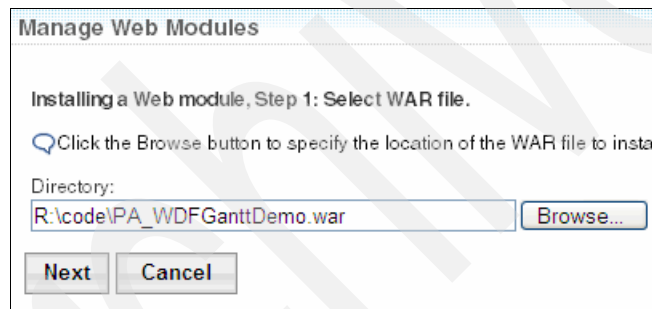


Figure 6-43 Select WAR file

3. Input a display name and context root for the application as shown in Figure 6-44 on page 283, then click **Finish** to deploy and start the application.

The Application will be installed with

The option to limit deployment name

Enterprise Application display name
PA_WDFGanttDemo

Context root
/wps/PA_WDFGanttDemo

☒ Start application
☐ Do not start application

Finish **Cancel**

Figure 6-44 Input display name and context root

Create Portal Page

Next you will need to create a portal page for the application:

1. Click **Administration**, then select **Portal User Interface-> Manage Pages** as shown in Figure 6-45.

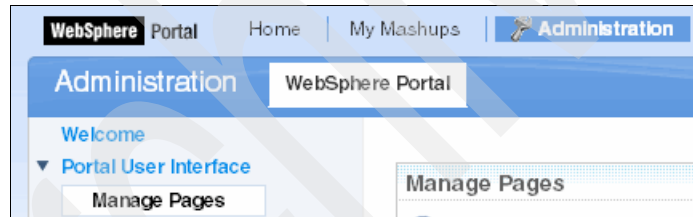


Figure 6-45 Manage Pages

2. Click **Content Root** link in the right panel of Manage Pages, then click **New Page** to open the **Page Properties** window.
3. Type a title name and unique name in the fields, for example jviews as shown in Figure 6-46 on page 284, then click **OK** to create the jviews portal page in the top head tab page.

Edit page: jviews

Use the controls below to work with your pages

Title:

Unique Name:

Note: If the unique name you entered for this page is

Friendly URL name:

Figure 6-46 Create portal page

Add portlet

Add the portlet into the Portal page using the following steps:

1. In **Manage Pages**, find the new created **jviews** page and click the image button of **Edit Page Layout** as shown in Figure 6-47, then open the layout page.

Pages in Content Root Add, Edit, Delete, and Reorder pages

Page 4 of 4 Jump to page:

Title	Unique name or Identifier	Status	
<input type="checkbox"/> Groups Viewer	ibm.portal.Groups.Viewer	Active	<input type="button" value="Up"/> <input type="button" value="Down"/> <input type="button" value="Add"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Refresh"/>
<input type="checkbox"/> People Finder	ibm.portal.People.Finder	Active	<input type="button" value="Up"/> <input type="button" value="Down"/> <input type="button" value="Add"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Refresh"/>
<input type="checkbox"/> Friendly Name Disambiguation	ibm.portal.FriendlyDisambiguation	Active	<input type="button" value="Up"/> <input type="button" value="Down"/> <input type="button" value="Add"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Refresh"/>
<input type="checkbox"/> MashupContentRoot	ibm.portal.MashupContentRoot	Active	<input type="button" value="Up"/> <input type="button" value="Down"/> <input type="button" value="Add"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Refresh"/>
<input type="checkbox"/> jviews	jviews	Active	<input type="button" value="Up"/> <input type="button" value="Down"/> <input type="button" value="Add"/> <input type="button" value="Edit"/> <input type="button" value="Delete"/> <input type="button" value="Refresh"/>

Page 4 of 4 Jump to page:

Figure 6-47 Edit Page Layout

2. In the Edit Layout page, click the **Add portlets** button, type jviews in Search field as shown in Figure 6-48 on page 285, then click **Search**.

Edit Layout

Search by: Title starts with Search: jviews Search

Select	Portlet Title	Description
<input type="checkbox"/>	Login	User authentication portlet
<input type="checkbox"/>	Profile Management	User profile management
<input type="checkbox"/>	WSRP Proxy Portlet	Display remote WSRP content in WSRP consumer
<input type="checkbox"/>	About WebSphere Portal	Displays Version and Copyright Statement
<input type="checkbox"/>	Search Sitemap	This portlet generates a sitemap that is crawlable by search engine
<input type="checkbox"/>	Portlet Wiring Tool	Allows user to edit connections between portlets
<input type="checkbox"/>	Manage Web Modules	Main administration of Web modules, portlet applications, and portlets
<input type="checkbox"/>	Manage Portlets	Manage the portlets that the current user has access to.
<input type="checkbox"/>	Manage Applications	Manage portlet applications that the current user has access to.
<input type="checkbox"/>	Web Service Configuration	Main administration of Web services

Page 1 of 10 1 Go Total: 100 Displayed: 10

Figure 6-48 Search for jviews

3. In the results page, select JViews Gantt Portlet and click **OK** as shown in Figure 6-49. When JViews Gantt Portlet is added into the current page, click **Done**.

Search by: Title starts with Search: jviews

Select	Portlet Title	Description	Unique name	Remote portlet
<input checked="" type="checkbox"/>	JViews Gantt Portlet	JViewsGanttPortlet		

Page 1 of 1 Total: 1 Displayed: 1

OK Cancel

Figure 6-49 Check JViews Gantt Portlet

Access the JViews portal page

To access the JViews portal page, click **More** → **jviews** as shown in Figure 6-50 on page 286.

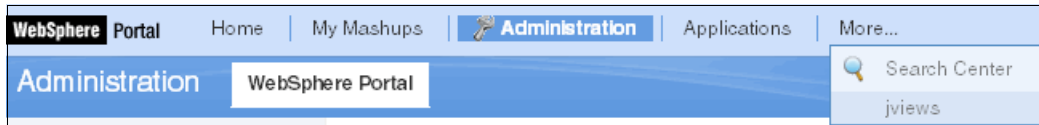


Figure 6-50 Click jviews head tab page

The JViews Gantt Portlet window displays as shown in Figure 6-51.

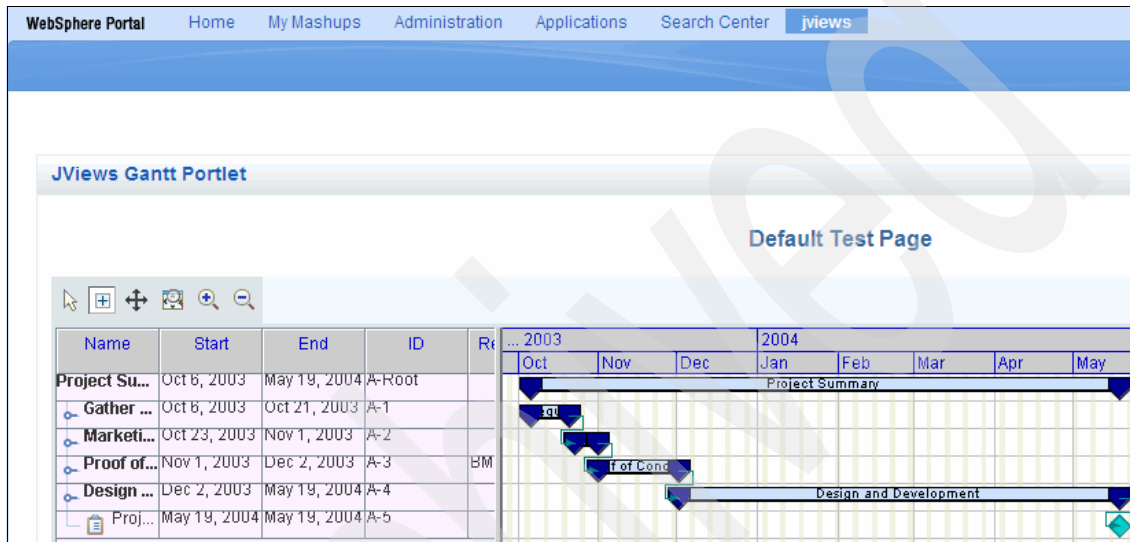


Figure 6-51 JViews Gantt Portlet

6.5.3 Conclusion

WebSphere Dashboard Framework is a full-featured, stand-alone development tool that allows you to quickly create your own dashboard applications from scratch.

IBM ILOG Visualization for Java components provide custom, graphical components for desktop, Ajax, and Eclipse displays. These components, delivered as point-and-click editing tools and full-featured software development kits (SDKs), allow Java developers to add intuitive, graphical displays to their mission-critical applications.

Combining the two together allows the developer to quickly build rich applications that provide the user with a graphical user interface that simplifies complex modern business backend services.

Additional material

This book refers to additional material that can be downloaded from the Internet as described in the following sections.

Locating the web material

The web material associated with this book is available in softcopy on the Internet from the IBM Redbooks web server. Navigate to:

<ftp://www.redbooks.ibm.com/redbooks/SG247917>

Alternatively, you can go to the IBM Redbooks web site at:

ibm.com/redbooks

Select the **Additional materials** and open the directory that corresponds with the IBM Redbooks form number, SG24-7917.

Using the web material

The additional web material file that accompanies this book is named SG247917.zip. It contains a directory for each chapter of this book that provides additional material. Additional material is provided for chapters 2, 3, 4, 5, and 6.

Chapter 2 files:

<i>Folder name</i>	<i>Description</i>
CompletedExamples	Completed projects for all samples in chapter 2
elixir	Resource files needed for ILOG Elixir sample
jviews	Images and style sheets needed for ILOG JViews sample
legacy	Source code for ITSOAirlines original application
rest	Code for the route resource handler

Chapter 3 files:

<i>File name</i>	<i>Description</i>
HR.xml	XML data source file used by the IBM Cognos report
IBM HR JViews.zip	Compressed data sample for the JViews Gantt chart.
IBM HR Elixir.zip	Compressed data sample for the Elixir Gantt chart.

Chapter 4 files:

<i>File name</i>	<i>Description</i>
Data.zip	Compressed data sample for the treemap iWidget
treemapWidget.zip	Compressed Adobe Flash Builder project for treemap Widget
treemap-iwidget.zip	Compressed Rational Application Developer for treemap iWidget

Chapter 5 files:

<i>Folder name</i>	<i>Description</i>
Flex Project	Adobe Flex project for ITSOMoviesDashboard
iWidget Project	WID Project and deployment files for the ITSOMoviesDashboardWidget
Monitoring	Compressed files for the ITSO Movies Business Monitor Model

Chapter 6 files:

<i>Folder name</i>	<i>Description</i>
GanttDemo	JViews Gantt Java Project that handles reading an XML data source in WebSphere Dashboard Framework context and feeds it into the JViews Gantt data model in a JViews Gantt JSF backbean
WDFGanttDemo	WebSphere Portlet Factory Project that handles all kinds of builders and accesses JViews Gantt JSF components in a JSP page

Downloading and extracting the web material

Create a subdirectory (folder) on your workstation, and extract the contents of the web material .zip file into that folder.

Archived

Abbreviations and acronyms

AIR	Adobe Integrated Runtime	LDX	layout data format
AJAX	Asynchronous JavaScript and XML	MVC	model view controller
API	application programming interface	OLAP	online analytical processing
BI	business intelligence	RAD	rapid application development
BPM	Business Process Management	REST	Representational State Transfer
BPMN	Business Process Modeling Notation	RIA	rich Internet applications
BSD	Berkeley Software Distribution	RPC	remote procedure call
CA	certificate authority	RSS	rich site summary
CBE	Common Business Event	SCADA	supervisory control and data acquisition
CMS	Cognos Mashup Services	SDK	software development kit
CRM	Customer Relationship Management	SOA	Service Oriented Architecture
CRUD	create, read, update, delete	SOAP	Simple Object Access Protocol
CSS	cascading style sheet channel subsystem	URI	Uniform Resource Identifier
EAR	enterprise archive	URL	Uniform Resource Locator
ECM	Enterprise Content Management	WAR	Web archive
ERP	Enterprise Resource Planning	WID	WebSphere Integration Developer
ESB	Enterprise Service Bus	XML	extensible markup language
HTML	Hypertext Markup Language		
HTTP	Hypertext Transfer Protocol		
IBM	International Business Machines Corporation		
IDE	integrated development environment		
ITSO	International Technical Support Organization		
JAR	Java archive		
JDBC	Java Database Connectivity		
JSF	JavaServer Faces		
JSON	JavaScript Object Notation		
JSP	JavaServer Pages		
JSR	Java Specification Request		
JVM	Java virtual machine		
KPI	key performance indicators		

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this book.

IBM Redbooks

The following IBM Redbooks publications provide additional information about the topic in this document. Note that some publications referenced in this list might be available in softcopy only.

- ▶ *Building IBM Business Process Management Solutions Using WebSphere V7 and Business Space*, SG24-7861-00
- ▶ *IBM Cognos Business Intelligence V10.1 Handbook*, SG24-7912

You can search for, view, or download Redbooks, Redpapers, Technotes, draft publications and Additional materials, as well as order hardcopy Redbooks publications, at this website:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services



IBM ILOG Visualization Integration Scenarios

(0.5" spine)
0.475" <-> 0.875"
250 <-> 459 pages



IBM ILOG Visualization Integration Scenarios



**Overview of IBM ILOG
JViews Enterprise
and IBM ILOG Elixir
Enterprise**

**Step-by-step
integration
scenarios for ILOG
visualizations**

**Examples with
practical scenarios
to learn from**

IBM ILOG Visualization products allow you to create the most advanced graphical user interfaces for line-of-business applications, help users understand their data better, and react to a changing market faster and smarter.

This IBM Redbooks publication describes two IBM Visualization products: IBM ILOG JViews Enterprise and IBM ILOG Elixir Enterprise. It provides detailed samples and scenarios covering how these products can be integrated with other IBM software such as IBM WebSphere REST Technology, IBM Cognos, IBM Mashup Center, IBM WebSphere Business Monitor and Business Space, and IBM WebSphere Dashboard Framework to provide Web 2.0 and Ajax visualization solutions.

This book is targeted to application interface developers and programmers who develop highly advanced graphical user interfaces using IBM ILOG Visualization products with IBM Cognos, IBM Mashup Center, IBM WebSphere Business Monitor and Business Space, and IBM WebSphere Dashboard Framework.

**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