
BEA WebLogic Workshop[™] 8.1 (beta) JumpStart Guide

Getting Started with BEA WebLogic Workshop 8.1



Copyright

Copyright © 2003 BEA Systems, Inc. All Rights Reserved
March 1, 2003

Restricted Rights Legend

This document may not, in whole or in part, be photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc. Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems, Inc.

Trademarks

Copyright © 2003 BEA Systems, Inc. All rights reserved. BEA, Tuxedo, and WebLogic are registered trademarks and BEA WebLogic Enterprise Platform, BEA WebLogic Server, BEA WebLogic Integration, BEA WebLogic Portal, BEA WebLogic Platform, BEA WebLogic Express, BEA WebLogic Workshop, BEA WebLogic Java Adapter for Mainframe, and BEA eLink are trademarks of BEA Systems, Inc. All other company and product names may be the subject of intellectual property rights reserved by third parties.



BEA Systems, Inc.
2315 North First Street
San Jose, CA 95131 U.S.A.
Telephone: +1.408.570.8000
Facsimile: +1.408.570.8901
www.bea.com

TABLE OF CONTENTS

COPYRIGHT	2
HOW TO USE THIS GUIDE	4
ADDITIONAL EDUCATIONAL RESOURCES.....	4
MORE INFORMATION, DATASHEETS, AND PRODUCT BROCHURES.....	4
EVALUATION SOFTWARE.....	4
PRESS RELEASES.....	4
DEVELOPER RESOURCES	4
BEA WEBLOGIC WORKSHOP 8.1™ PRODUCT OVERVIEW	5
VISUAL DEVELOPMENT ENVIRONMENT.....	5
THE RUN-TIME FRAMEWORK	7
JAVA CONTROLS	9
APPLICATION TYPES IN BEA WebLogic Workshop 8.1™	11
BEA WEBLOGIC WORKSHOP 8.1 JUMPSTART EXERCISES	13
INSTALLING BEA WebLogic Workshop™.....	13
THE JUMPSTART EXERCISE SCENARIO – AVITEK ELECTRONICS	14
BEA WebLogic Workshop™ PRODUCT EVALUATION STEPS	15
EXERCISE #1. GETTING ORIENTED AND SETTING UP SAMPLE DATA	16
VISUAL DEVELOPMENT ENVIRONMENT.....	17
AUTOMATED DEPLOYMENT	18
EXERCISE #2. BUILDING A CUSTOM JAVA CONTROL.....	20
BUILT-IN JAVA CONTROLS	20
CUSTOM JAVA CONTROLS	21
ASYNCHRONOUS CONTROLS.....	25
EXERCISE #3. CREATING AN ENTERPRISE-CLASS WEB SERVICE	29
SUPPORT FOR ASYNCHRONOUS WEB SERVICES	31
AUTOMATED DEPLOYMENT AND TESTING.....	32
LOOSE-COUPLING	35
EXERCISE #4. CREATING AN ENTERPRISE-CLASS WEB APPLICATION.....	40
JAVA PAGE FLOW TECHNOLOGY.....	40
AUTOMATED DATA-BINDING	45
AUTOMATED DEPLOYMENT AND TESTING.....	48
(BONUS) EXERCISE #5: CREATING A WEB APPLICATION FROM A JAVA CONTROL.....	53
(BONUS) EXERCISE #6: USING XMLBEANS FOR HANDLING XML IN JAVA	57
BEHIND THE SCENES.....	61
CONCLUSION.....	62
APPENDIX: BEA WEBLOGIC WORKSHOP 8.1™ SPECIFICATIONS	63
NEXT STEPS: PRODUCT INFORMATION AND DEVELOPER RESOURCES	63
ABOUT BEA	64

How To Use This Guide

This JumpStart Guide brings together some essential material that may be helpful as you prepare to explore and evaluate BEA WebLogic Workshop 8.1 beta. This document provides an overview of BEA WebLogic Workshop 8.1, a detailed look at its features and benefits, a discussion on application development with BEA WebLogic Workshop 8.1, and important information you may need as you evaluate the product. We recommend you read through this guide before you begin to explore the software itself. BEA has provided a product demonstration that lets you quickly and comprehensively explore the capabilities of BEA WebLogic Workshop 8.1 beta. Detailed step-by-step instructions for these demonstrations are contained within this JumpStart Guide.

In addition to the material covered in this guide, you may want to make use of the wealth of online resources available to you at the BEA Web site (www.bea.com) including datasheets, product brochures, white papers, demo software, press releases, and more.

Additional Educational Resources

More Information, Datasheets, and Product Brochures

BEA WebLogic Workshop Product Web site
<http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/workshop>

Evaluation Software

BEA WebLogic Workshop 8.1
<http://commerce.bea.com/showproduct.jsp?family=WLV&major=8.1&minor=-1>

Press Releases

Press releases related to BEA and products
www.beasys.com/press/room.shtml

Developer Resources

dev2dev Online, the BEA developer community network
<http://dev2dev.bea.com>

Listing of BEA WebLogic Workshop-related newsgroups
www.beasys.com/support/newsgroup.shtml

BEA WebLogic Workshop 8.1™ Product Overview

BEA WebLogic Workshop 8.1 is a unified, simplified, and extensible development environment that makes it incredibly easy for all developers, not just J2EE experts, to build enterprise-class, standards-based applications across the entire BEA WebLogic Platform. BEA WebLogic Workshop's unique programming model accelerates software development by providing simplified abstractions designed to enable all developers to build applications better and dramatically faster than with traditional programming approaches.

The primary components of Workshop's innovative programming model are the visual development environment, run-time framework, and Java Controls.

Visual Development Environment

The WebLogic Workshop 8.1 visual development environment delivers unprecedented ease-of-use for J2EE application development. The Design View (shown in **Figure 1**) provides a graphical depiction of the application being created so that developers can view and visually edit the application's interaction with clients and back-end resources. Moreover, the same visual development environment is shared by all WebLogic Enterprise Platform 8.1 applications, dramatically reducing the learning curve for building, testing, and debugging custom controls, Web services, Web applications, portals, and integration applications.

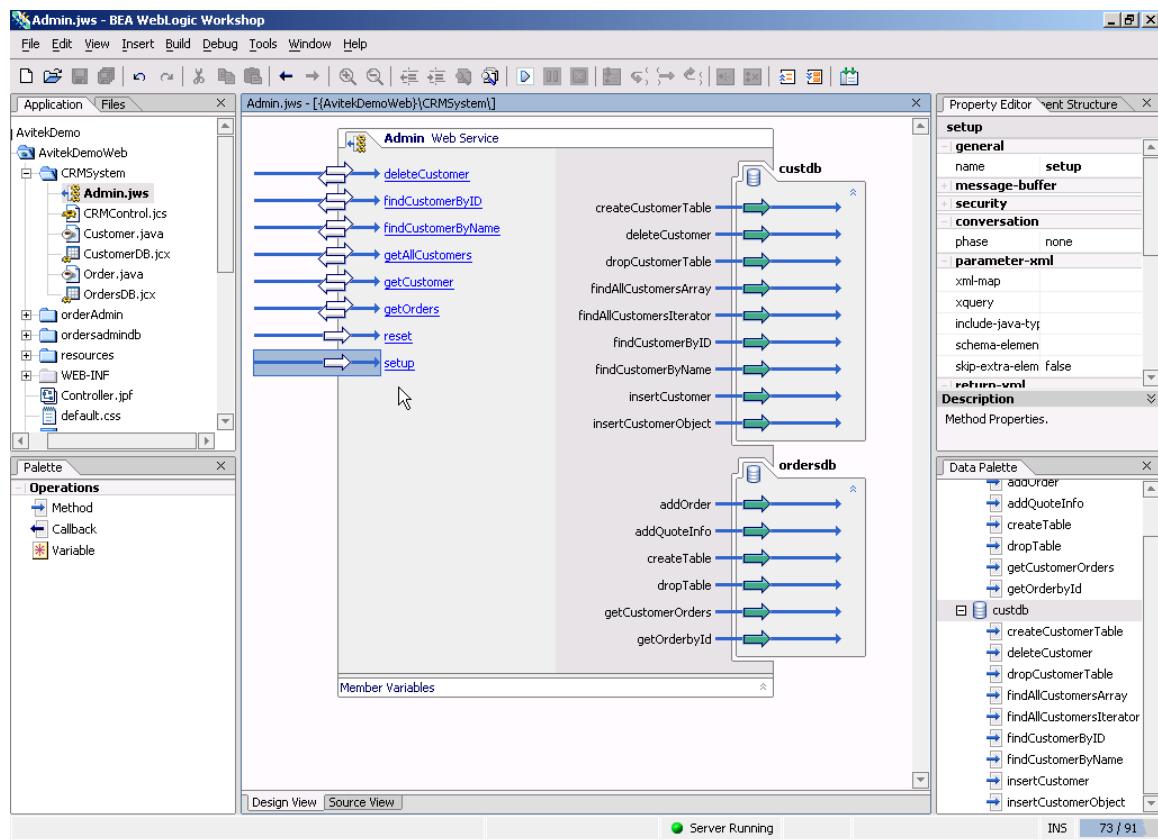


Figure 1. BEA WebLogic Workshop 8.1 Visual Development Environment

The WebLogic Workshop programming model is based on intuitive concepts such as controls, methods, and properties to enable event-based development, thereby eliminating the need for developers to master complex J2EE APIs and object-oriented programming disciplines. The underlying mechanism to enable this simplified development mode is the use of annotated Java code. The WebLogic Workshop visual development environment produces standard Java files with additional annotations inserted by Workshop (for example, when the developer sets properties or adds controls) to specify the appropriate run-time application behavior. These annotations enable the Workshop run-time framework to automatically generate the J2EE infrastructure components, thereby abstracting the user from the low-level infrastructure plumbing that would otherwise be required. As a result, developers are free to focus on writing business logic to solve the business problems at hand.

Developers write Java code using the Source View, as shown in **Figure 2**. Developers here have direct access to the source code, including the code annotations, and two-way code editing features ensure that any changes made in the Design View are instantly reflected in the Source View, and vice versa. The Source View allows developers to write procedural Java code to handle methods and events, and assists developers with many productivity-enhancing features such as code-completion, syntax checking, and auto-import.

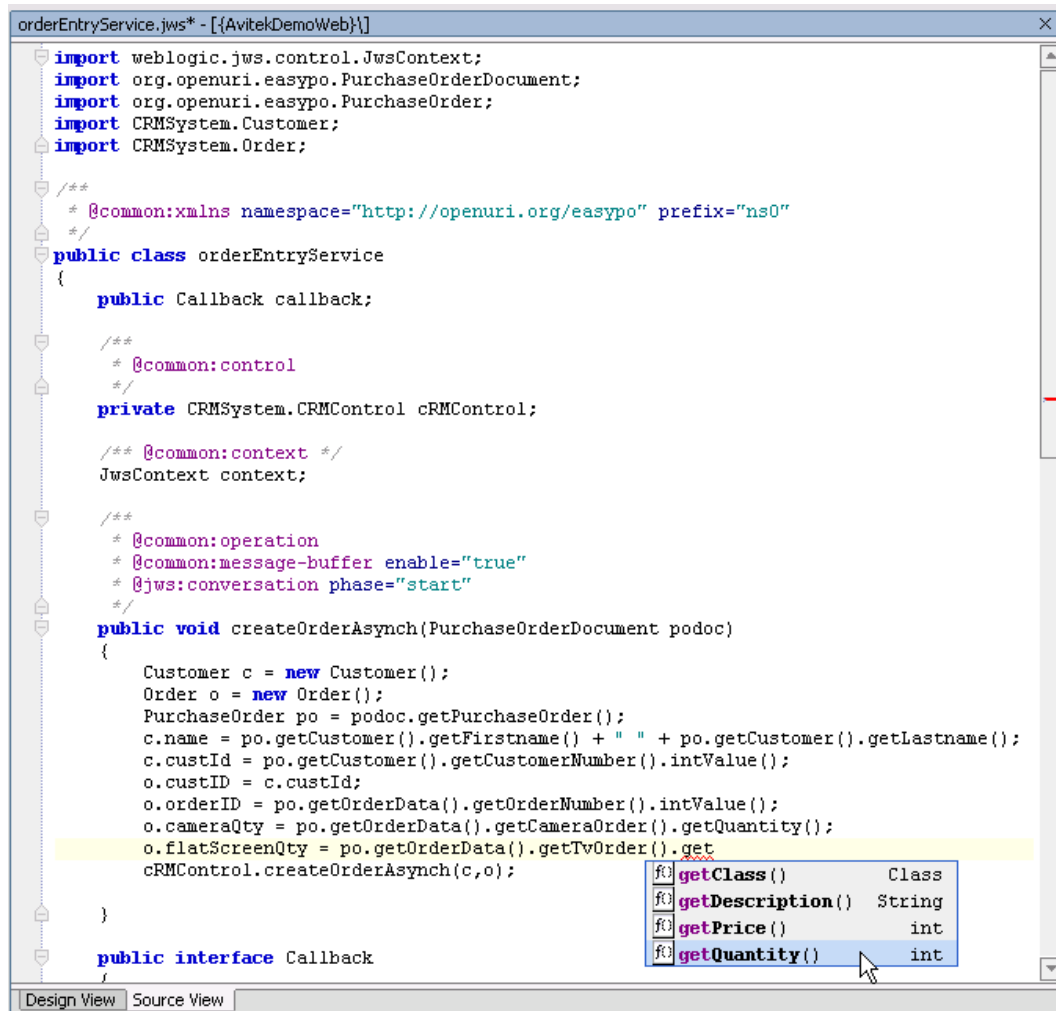


Figure 2. Code completion features assist developers in the Source View

WebLogic Workshop also features an industrial-strength, multi-language, integrated debugger empowering developers to easily build, test, and debug applications from the same visual environment. The BEA WebLogic Workshop 8.1 debugger allows debugging of the deployed code running in the application server rather than the local Java files, enabling real-time feedback and higher application quality.

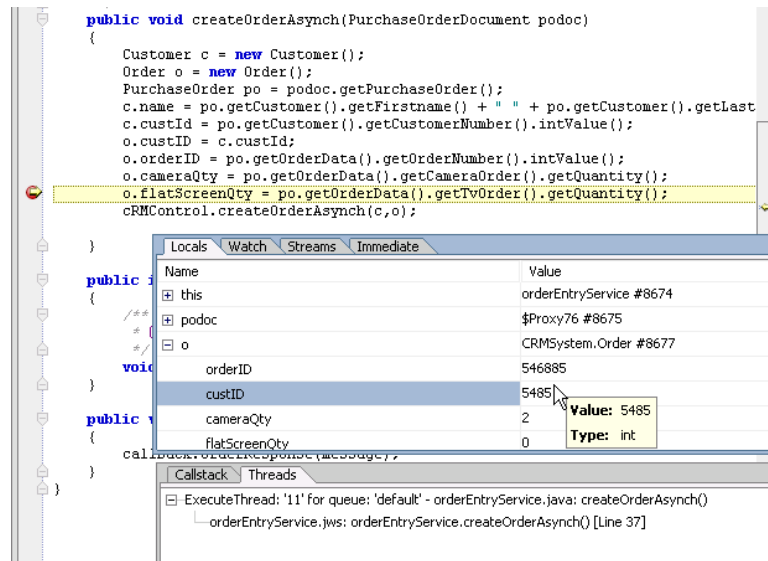


Figure 3: WebLogic Workshop's integrated debugger in action

Furthermore, WebLogic Workshop 8.1 improves the development experience by including many IDE enhancements such as source code control integration, an enhanced windowing environment, complete tool configurability, and shared tools and palettes.

The Run-Time Framework

BEA WebLogic Workshop's run-time framework provides the abstraction layer between developers and complex J2EE system infrastructure. Instead of dealing with time-consuming API-level infrastructure code, component configuration, and deployment details, developers are free to use the visual development environment to write procedural Java code where necessary and access advanced functionality through simple, declarative annotations. The WebLogic Workshop run-time framework interprets the annotated code to generate the appropriate J2EE code including standard EJB, JMS, and JDBC components. And in this process, the run-time framework also manages the design and implementation issues associated with J2EE architectures, so that applications are implemented on reliable, scalable, and secure enterprise-class architectures.

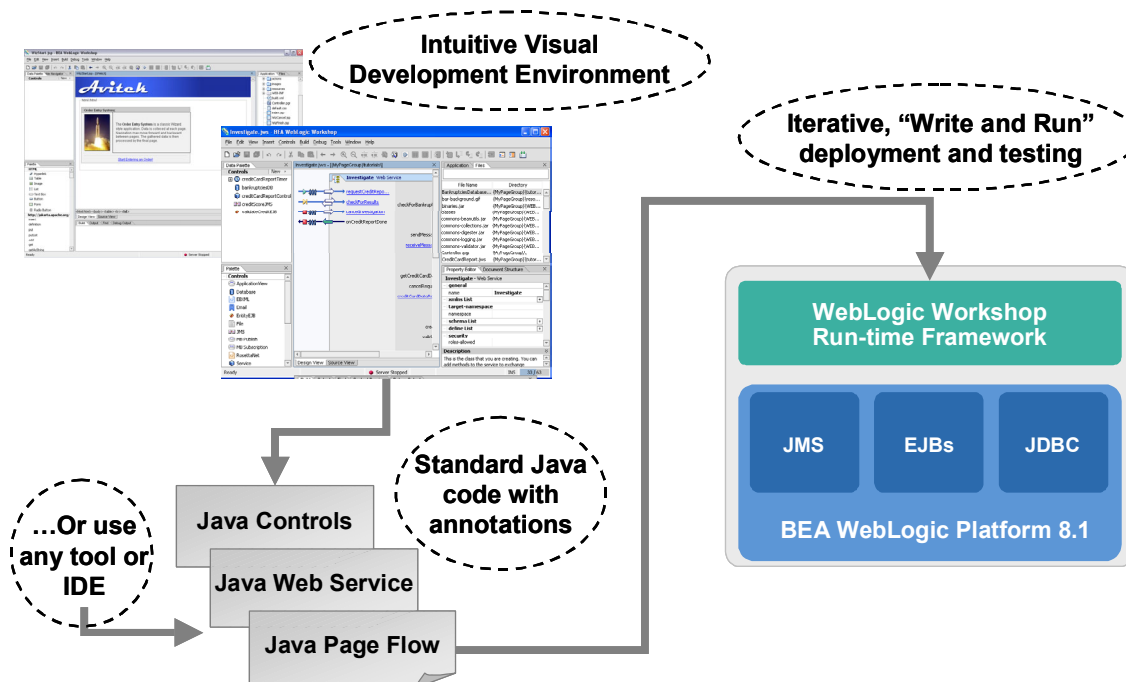


Figure 4. Connecting the Design-Time and Run-Time components

The WebLogic Workshop run-time framework – a standard J2EE application that runs on top of BEA WebLogic Server 8.1 – also represents the convergence layer that unifies all WebLogic Enterprise Platform application types.

BEA WebLogic Workshop's run-time framework abstracts the developer from the intricate details of developing and deploying enterprise-class applications in the following ways:

- **Java Web Service (JWS) File Support:** Introduced in WebLogic Workshop 7.0, Web service applications are developed as JWS files – standard Java files with a set of Javadoc annotations that allow developers to specify properties and gain access to key Web service functionality through simple declarative annotations. This approach conveniently eliminates the need for developers to worry about SOAP marshalling, Java-to-XML binding, WSDL file creation, underlying bean deployment, and much more. JWS is in the process of being standardized through JSR (Java Specification Request) 181.
- **Java Page Flow (JPF) File Support:** WebLogic Workshop 8.1 Web applications are built with Java Page Flow, a Java innovation that makes it easy for developers to build enterprise-class Web applications using a visual approach to specify the flow of pages. Developers can write business logic in actions that can be called from pages in the application. And to greatly simplify data accessibility, WebLogic Workshop provides tag libraries plus drag-and-drop wizards to automate data-binding between pages and data elements (including fields, lists, trees, grids, etc.) from any database, Web service, or Java Control. In addition, the run-time framework automatically provides support for details such as sessions and state-management, and ultimately creates Java Page Flow applications on an underlying MVC (model-view-controller) architecture leveraging Struts.
- **Java Controls (JCX/JCS) File Support:** Java Controls enable developers to access back-end enterprise resources through a simplified interface already familiar to many developers. Instead of calling a series of J2EE API calls, developers can implement methods and handle events to access information from assets such as databases, external web services and EJB components. Customization of controls can be performed by simply setting properties. The run-time framework also makes Java Controls extensible so that all users can write their own

custom control. Controls make it easy to build powerful, asynchronous business logic that can be used from conversational web services and workflows. More details are available in the next section.

- **Portal/Integration Application Extensions:** WebLogic Workshop 8.1 Platform Edition includes framework extensions that enable developers to build, test, and deploy advanced portal and workflow/EAI/BPM applications in conjunction with BEA WebLogic Portal 8.1 and BEA WebLogic Integration 8.1, respectively. The Workshop run-time framework offers a unified architecture leading to a seamless developer experience across all these application types.
- **Support for Asynchronous Communications:** Writing the plumbing code for asynchronous communication and two-way messaging can be tedious and time consuming. WebLogic Workshop simplifies this process by providing a simple property setting to enable message-buffering, which automatically creates asynchronous support for Web services or custom controls. The framework handles the details of managing the queue and abstracts the user away from the J2EE Messaging APIs. For asynchronous Web services, Workshop provides a simple conversational metaphor for message exchange with clients. The WebLogic Workshop framework automatically correlates related messages and manages state over long running conversations.
- **XQuery Mapping and XMLBeans:** WebLogic Workshop provides a range of solutions to ensure true loose-coupling between applications for greater application reliability and scalability. The XQuery Mapping features allows a developer to visually map data fields, and the run-time framework automatically handles the complex, underlying data transformations. Likewise, XMLBeans provides strongly-typed Java object interfaces while preserving full access to the underlying XML messages. Together these solutions, supported by the WebLogic Workshop run-time framework, ensure that XML data is always a "first class citizen."
- **"Write and Run" Deployment:** WebLogic Workshop provides an enhanced iterative development model through the run-time framework's support for one-click automatic deployment to the BEA WebLogic Enterprise Platform. This enables developers to focus on ensuring code quality without the concerns of the underlying platform details, rapidly accelerating the build-and-test cycle and dramatically increasing productivity.
- **Automated Test Harness:** The run-time framework automatically generates a browser-based test client for each deployed application to make it easy to verify correct functionality and eliminate costly bottlenecks in the test cycle. This enables developers to go from writing code to testing its correctness in seconds.

Java Controls

In WebLogic Workshop 8.1, Java Controls provide developers the ability to easily connect with existing data, systems, applications, and business logic. Java Controls are visual components with methods and properties that handle all of the J2EE details in connecting to an external resource or piece of business logic. Developers interact with controls by handling events and settings properties. Workshop also makes it easy to build custom controls using the similar, simplified Java code approach available elsewhere in Workshop. These characteristics empower developers to declaratively specify behavior and then focus on handling events and calling methods with standard procedural Java code instead of learning APIs. Moreover, Java Controls provide best-practices for accessing resources and application logic to ensure that resources are optimally used.

BEA WebLogic Workshop 8.1 comes with a set of built-in Java Controls including the following:

- **Web Service Control** lets developers access Web services as local objects without worrying about any of the lower-level Web service protocol details. The developer need only specify

the WSDL file for the target service and the control will automatically prepare the connection and expose the target service's methods in the visual development environment. The Service Control makes it easy to connect to Web services regardless of the implementation technology, including easy access to .Net-based services.

- **Database Control** lets developers focus on what they need from a database without having to worry about the JDBC (Java Database Connectivity) API. Developers can focus on what they know (SQL statements and Java functions) and Workshop's run-time framework automatically handles the underlying plumbing.
- **EJB Control** enables a team development approach in which the application developer may use enterprise business logic that already exists, or is being developed by an enterprise developer. The EJB Control simplifies the EJB programming model and makes a remote EJB object appear as just another class on which you can set properties and call methods.
- **JMS Control** allows easy access to message queues and shields the developer from the details of the Java Messaging Service API. Developers can focus on the content of the messages being sent and received instead of the machinery around them.
- **Timer Control** helps developers program in an asynchronous world. The Timer Control can be used to implement message timeouts, poll web services to determine completion, and coordinate multiple asynchronous message replies.
- **And many more:** WebLogic Workshop 8.1 adds many more built-in Java Controls to make it easy for developers to connect to and use IT assets, including FTP, eMail, Tuxedo, Portal, Integration Controls, and more.

In addition to the suite of built-in Java Controls mentioned above, BEA WebLogic Workshop 8.1 also makes the Java Controls model extensible so that all users (including ISVs) can build custom Java Controls – re-usable business logic components that seamlessly plug into the Workshop development environment. To create a custom Java Control, developers simply use familiar visual designers to specify the interface (i.e. the methods and events supported), set properties to dictate run-time behavior, and write the business logic using procedural Java code to implement these methods.

The extensible nature of Java Controls makes them the ideal model for deploying service-oriented architectures in which business logic components are created as independent modules that can be used and re-used to service multiple types of end-user applications. The many benefits of using Java Controls to promote this type of efficient, modular design for software reuse include the following:

- Java Controls are able to describe themselves at design-time so that developers can gain a visual representation of server-side business logic in the familiar Workshop development environment.
- Java Controls expose a set of simple properties by which a user can implement advanced run-time functionality such as asynchronous communication, security roles, lifecycle events, transaction support, etc. Authors of custom controls have the freedom to populate the Properties sheet for the benefit of the controls' downstream users.
- Java Controls are easily packaged and distributable as standard JAR files. Moreover Java Controls are nestable so it is easy for them to become the component of re-use.
- Java Controls enable connectivity to any IT asset, ISV application, or piece of business logic while preserving the developer's abstraction from the majority of the infrastructure complexity. This means developers can largely focus on procedural Java instead of J2EE or other vendor-specific APIs.

As a result, Java Controls are a compelling way to encapsulate business logic in reusable components that become the central building blocks for all WebLogic Platform applications.

Application Types In BEA WebLogic Workshop 8.1™

The visual development environment, run-time framework, and Java Controls together enable BEA WebLogic Workshop's simplified programming model. Importantly, this innovative development approach is unified such that developers need only learn a single programming model to build and integrate the full suite of WebLogic Platform applications including Web services, Web applications, portals, and integration applications.

Enterprise-Class Web Services

Enterprise IT professionals recognize that Web services can provide tremendous business value by reducing infrastructure complexity and managing the costs traditionally associated with integrating applications across disparate systems and geographies. But to be successful, it is critical to go beyond the simple, synchronous view of Web services prevalent today. Building and extending upon Workshop's strong leadership in Web services, BEA WebLogic Workshop 8.1 provides built-in architectural support for enterprise-class web services – featuring asynchrony, loose coupling, business-level documents, enhanced security, and reliable messaging. These characteristics are critically important in building a web services architecture capable of effectively addressing enterprise challenges, such as application integration. Moreover, WebLogic Workshop offers XQuery mapping and XMLBeans technology, two breakthrough innovations that will make Java developers an order of magnitude more productive when working with XML-based data and documents.

In Exercise #3, you will create an enterprise-class Web service, and in Exercise #6, you will gain exposure to the power of XMLBeans.

Powerful Web Applications

BEA WebLogic Workshop 8.1 introduces a set of visual designers, controls and framework extensions that enable developers to create powerful server-side applications with dynamic JSP/HTML user interfaces. Workshop helps developers build dynamic, data-bound pages with drag-and-drop visual editors, and furthermore, makes it incredibly easy to specify powerful application functionality via Java Page Flow technology. Java Page Flow provides an abstraction layer that helps developers assemble enterprise-class Web applications simply by specifying pages, actions, navigation, and data, without having to worry about complexities such as sessions, state management, and data-binding. Page Flows are built from the ground up to support a robust, MVC style architecture, and actually are compiled into standard Java code that runs on top of the open source Struts framework. BEA WebLogic Workshop 8.1 empowers developers to use a simplified, visual programming model to build complex, multi-state, data-rich Web applications that combine business and presentation logic and leverage a common Java Controls methodology.

In Exercise #4, you will complete a Web application, and in Exercise #5, you will automatically generate a new Web application directly from a Java Control.

Portals and Integration Applications

Across most IT environments, developers traditionally encounter a heterogeneous tools environment where disparate tools and skill-sets are required for building and integrating custom applications versus portal applications and workflow applications. This deficiency has stifled developer productivity and heightened infrastructure complexity. But now WebLogic Workshop Platform Edition 8.1 leads the industry by enabling developers to incorporate advanced EAI/BPM and process portal functionality into custom development projects by unifying workflow, presentation, and business logic in a single environment using the same programming model. To

accomplish this, WebLogic Workshop 8.1 Platform Edition includes framework extensions, additional controls and visual designers to provide developers the ability to build and integrate all portal applications and integration applications, in conjunction with WebLogic Portal and WebLogic Integration, respectively.

The product demonstrations described in this Guide are based on applications supported in the WebLogic Workshop Application Developer Edition, and thus do not include examples focusing on WebLogic Portal and WebLogic Integration functionality.

BEA WebLogic Workshop 8.1 JumpStart Exercises

This JumpStart Guide will lead you through the steps of building enterprise-class applications with BEA WebLogic Workshop™ 8.1. Along the way, you will become familiar with Java Controls and use them to access relational databases. You will also build a custom Java Control to enable asynchronous integration with multiple databases. Finally, you will leverage this custom control in building both an enterprise-class Web service and an enterprise-class Web application.

In doing the JumpStart exercises, you will experience Workshop's visual development environment and the simple, shared programming model it offers. Moreover, you will gain an appreciation for the Workshop run-time framework and its ability to simplify J2EE development and deployment. Once the JumpStart Guide has been completed, you will have completed entire development cycles within minutes - building applications, integrating with external resources, deploying to enterprise-proven application infrastructure, and testing in real-time.

Building a similar application without the help of BEA WebLogic Workshop 8.1 would usually take several days – but with WebLogic Workshop 8.1, you should be finished in under an hour!

Installing BEA WebLogic Workshop™

The JumpStart exercises require that you install both WebLogic Workshop 8.1 beta and the JumpStart Kit. You will need:

- BEA WebLogic Workshop 8.1 beta – You can install the program from CD or download at: <http://commerce.bea.com/showproduct.jsp?family=WLW&major=8.1&minor=-1>
- JumpStart Kit (BEA_WebLogicWorkshop81_JumpStartKit.zip) – Includes this guide as well as project files required for the JumpStart Exercises. This zip file can be downloaded from the BEA WebLogic Workshop website at: http://www.bea.com/framework.jsp?CNT=step_2.htm&FP=/content/new_releases/products/workshop/get_started/

Once you have these files, the process of installing BEA WebLogic Workshop 8.1 and preparing for the JumpStart Exercises consists of the following steps:

- Double-click on platform810_win32.exe. This starts the BEA WebLogic Platform 8.1 installation program and will install BEA WebLogic Workshop 8.1™.
- Click Next to progress past the Welcome screen
- Read and accept the License Agreement, and select Next to continue
- Select Next while leaving the default choice of c:\bea as your BEA Home Directory
- Select the default selection for "Typical Installation" and click Next
- Select Next while leaving the default choice of C:\bea\weblogic81b as the Product Directory
- Select Next to begin the installation process, which will take approximately 15 minutes
- On the Installation Complete dialog box, clear the checkboxes at the bottom of the screen (no need to install XMLSpy or use the configuration wizard at this time) and click Done

-
- After the product has been successfully installed, you need to prepare the project files. Simply unzip the contents of the *BEA_WebLogicWorkshop81_JumpStartKit.zip* to a new folder named *c:\workshop81demo*. After doing this, you should have three files in the *c:\workshop81demo* directory:
 - *BEA_WebLogicWorkshop81_JumpStartGuide* (this document)
 - *demo-orderentry.zip* (the application template – no need to unzip this file!)
 - *purchase-order.xsd* (an example schema for use in the exercises)
 - You will use these files during the course of the product tutorial. You're all set to go!

The JumpStart Exercise Scenario – Avitek Electronics

This example involves a fictitious company named Avitek, a major consumer electronics firm. Avitek has a proprietary, home-grown order management system. Today, only a limited number of authorized distributors are able to electronically submit product orders to Avitek, with all other forms of order entry being done manually. Avitek maintains a number of databases and back-end computing resources that are an integral component of the firm's operations. Any initiative to extend the current order management system would be required to leverage the existing IT infrastructure which has been developed piece-meal over the last twenty years. This need for application integration had rendered prior attempts at modernization too complex and certainly too expensive.

Recently, many of Avitek's major competitors have introduced additional channels of distribution. To remain competitive in its industry, Avitek must respond by increasing its channel offerings as well. First, Avitek wants to offer all electronics distributors the opportunity to easily connect with Avitek's Order Entry System via standards-based Web services. Second, Avitek desires to capitalize on the growing demand for e-commerce by offering a hosted Web application that allows end-users to purchase certain items on-line directly from Avitek. To be successful, Avitek must ensure that these applications have the following features:

- Standards-based to ensure interoperability with the largest number of distributors
- Flexible to allow changes to the system and the information exchanged between partners over time
- Integrates with existing data, systems, applications, and business logic required for executing order management processes
- Designed to accommodate the latency and service disruptions associated with dealing with legacy systems and external data sources
- Reliable, available, and scalable to meet the demands of its expanding user base and maintain their industry reputation for quality.

Because Avitek needs to move quickly to build and extend its new Order Entry System, and many of their key J2EE architects are already busy maintaining the core infrastructure, Avitek is planning to rely on its existing staff of application developers to build this new system. Since many of these developers may not be experts in J2EE, the chosen technology platform will need to be accessible by these developers, but at the same time support the rock-solid reliability, availability, and scalability that Avitek's partners and customers demand.

To meet these requirements, Avitek has decided to use BEA WebLogic Workshop 8.1 to help solve the pressing business problem at hand, and improve the project's time-to-value. A diagram of the planned Avitek Order Entry System is provided below in **Figure 5**.

In this product tutorial, you will build the Avitek Order Entry System that will help Avitek maintain and extend its market share in the competitive electronics industry.

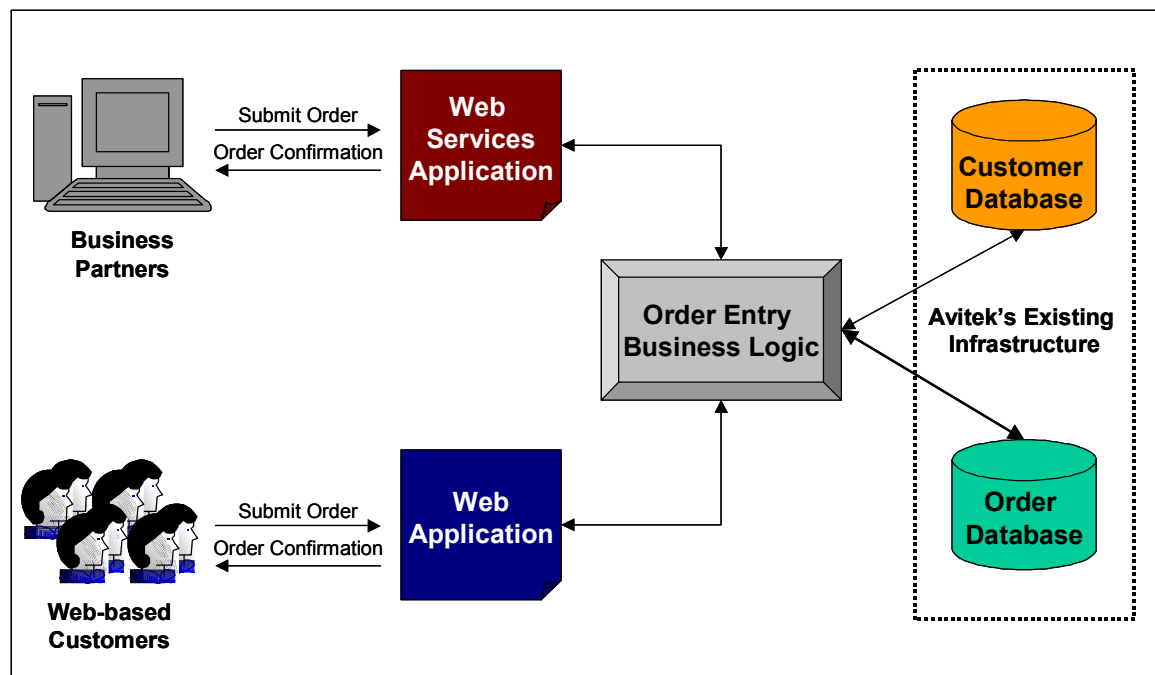


Figure 5. Architecture for Avitek's envisioned Order Entry System

BEA WebLogic Workshop™ Product Evaluation Steps

This JumpStart Guide will take you through the steps of building an enterprise-class Order Entry System that will increase Avitek's business efficiency, customer responsiveness, and market presence.

In the first exercise, you will become familiar with the Workshop development environment and set-up the application workspace for the subsequent exercises.

In the second exercise, you will create a custom Java Control that asynchronously integrates with both the Avitek Customer Database and the Avitek Order Database and also implements the necessary business logic to perform order management functions.

In the third exercise, you will leverage the custom control you built to create an asynchronous, secure, and loosely-coupled enterprise-class Web service that empowers Avitek to instantly expose advanced Order Entry functionality to thousands of potential business partners.

In the fourth exercise, you will again leverage the custom control, but this time to build, test, and deploy a Web application with dynamic JSP/HTML user interface thereby enabling Avitek to effectively reach the millions of potential customers who prefer to shop online.

For evaluators interested in further exploring features of BEA WebLogic Workshop, we have also provided two bonus exercises that highlight additional product capabilities. Exercise 5 demonstrates the ability to quickly build "starter" Web applications from a Java Control – in this case an application to administer the Avitek Order Database. Exercise 6 demonstrates the power of XMLBeans for elegantly handling XML data and documents in applications created using BEA WebLogic Workshop 8.1.

Exercise #1. Getting Oriented and Setting up Sample Data

In order to offer a tutorial that allows you to easily experience the full breadth of features and functionality of BEA WebLogic Workshop 8.1 in an efficient manner, we provide in this JumpStart Kit an application template that pre-defines several application components which you will extend into a set of enterprise-class applications constituting Avitek's improved Order Entry System.

- To leverage the Order Entry System template for this tutorial, copy the *demo-orderentry.zip* file from the *c:\workshop81demo* folder to the *C:\bea\weblogic81b\workshop\templates* directory. Note that there is no need to extract the components of the *demo-orderentry.zip* file – just copy the file in its entirety. If you installed the BEA WebLogic Workshop application in a BEA home other than *c:\bea* then adjust the above destination accordingly.

Now you are ready to begin exploring BEA WebLogic Workshop 8.1:

- Navigate the Windows Start menu to launch BEA WebLogic Workshop 8.1: **Start Menu → Programs → BEA WebLogic Platform 8.1 Beta → WebLogic Workshop 8.1**
- The BEA WebLogic Workshop environment will open to display a sample application, but you will create a new application for the Avitek Order Entry demo.
 - First, close down any open projects by selecting **File → Close Application**.
 - Using the menus, select **File → New → Application**. In the *New Application* dialog box that will appear, select *Demo: Order Entry Demo Application* in the right-side pane and enter an application name such as *AvitekDemo*.
 - In the Server field, specify *c:\bea\weblogic81b\samples\workshop* as the domain for this application, as shown in **Figure 6**.
 - Click **Create**, and then click **Yes** on the following dialog box to affirm you want to create a new directory.

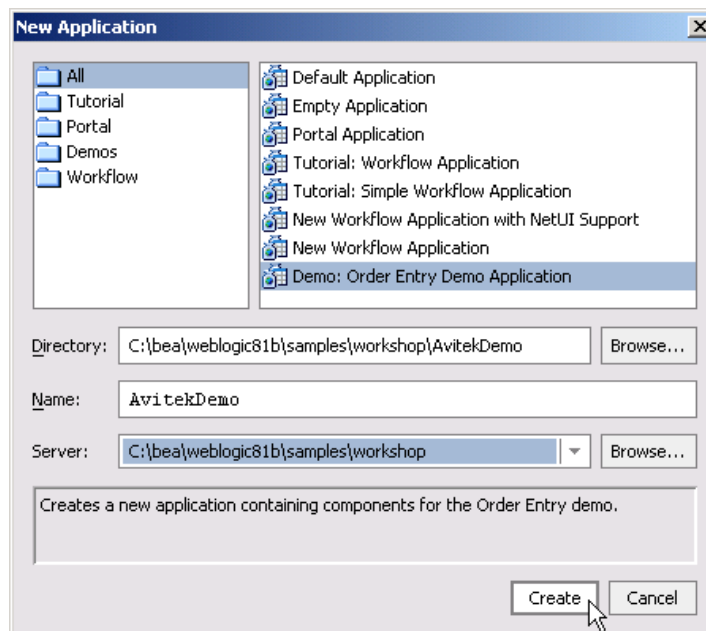


Figure 6: Create a new Workshop application based on the provided Order Entry Demo template

- Now that the *AvitekDemo* application has been created based on the Order Entry Demo template, you can open any of its member files by simply navigating in the Application Window (top-left corner).
- To get started, expand the *CRMSystem* folder under the *AvitekDemoWeb* project tree, and double-click on *Admin.jws*. This action will open the *Admin.jws* web service file in the Design View.

Visual Development Environment

Now is a good time to get acquainted with the WebLogic Workshop 8.1 visual development environment, as shown in **Figure 7**. The environment hosts a number of visual designers corresponding to the application type being created. If you have opened *Admin.jws* as instructed above, you are seeing the JWS designer for working with a Web services application. In the center of the screen is the Design View which provides developers with a graphical representation of the Web service. The canvas displays the operations available to clients via arrows on the left-side of the canvas, and Java Controls to interface with back-end resources and business logic are shown via arrows on the right-side of the canvas. The Design View is surrounded by a number of dockable and configurable windows to aid the developer's progress. For example, the Application Window provides easy access and viewing of resources contained in the active application and the Property Editor enables the developer to quickly set properties on the controls and methods.

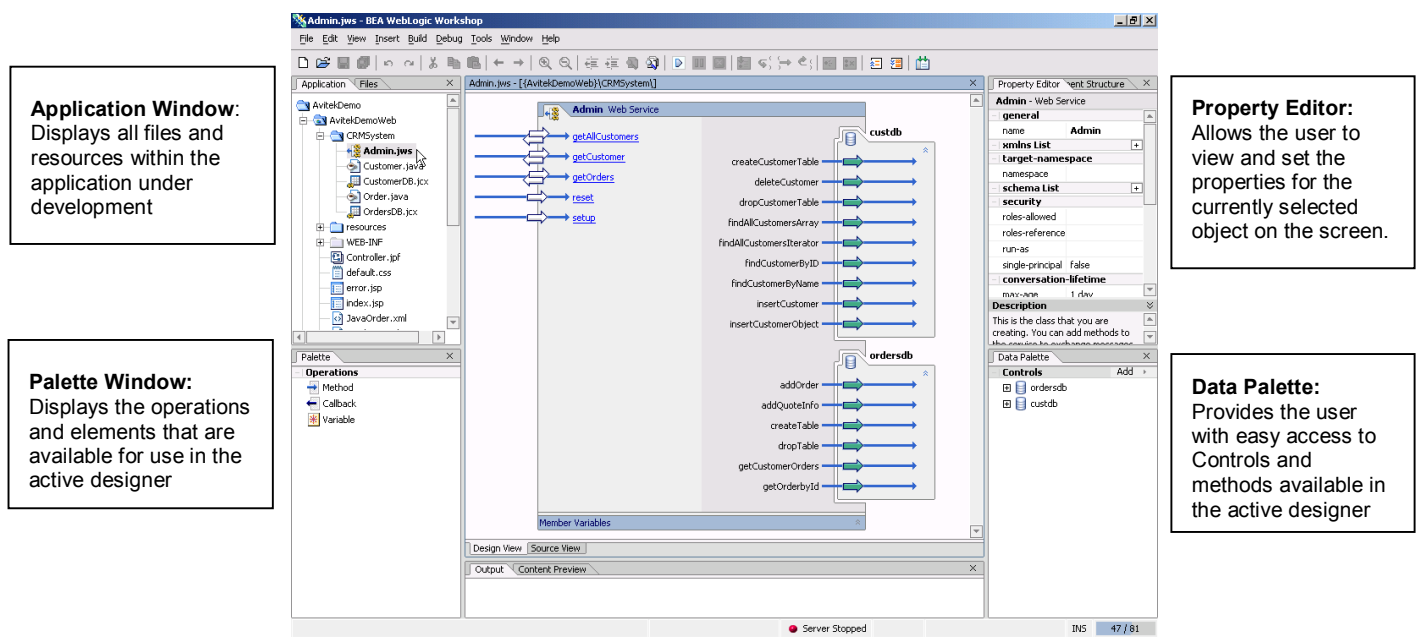


Figure 7. BEA WebLogic Workshop 8.1 Visual Development Environment

The Avitek Order Entry System will require integrating with two databases: one for customers and a second for orders. For the purposes of this demonstration, you need to ensure the database tables are properly established. For your convenience, handy administrative methods to configure the database resources are provided via the *Admin.jws* Web services application.

- To build the database and load the sample data, you need to run *Admin.jws* by using the menus to select **Debug** → **Start** or by clicking the **Start** icon in the toolbar. Optionally, you could select **Debug** → **Start without Debugging** for higher performance since you won't be requiring the integrated debugger at this time.

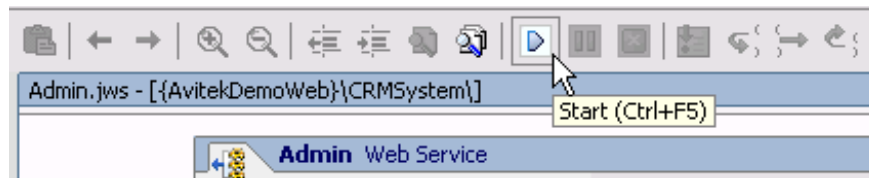


Figure 8. Use the toolbar to start the *Admin.jws* Web service application

- If you have not already done so, you will be prompted to start the WebLogic Server, which you should do now. Note that starting the WebLogic Server may take a few minutes to complete. (**Note:** Do not shut down the “Starting WebLogic Server” window as this action would shut down the server as it is starting up. You may however hide the window by clicking the **Hide** button.)

Automated Deployment

After the *Admin.jws* application is built, it will be automatically deployed to the BEA WebLogic Server 8.1 and the *Workshop Test Browser* will automatically appear. This is evidence of WebLogic Workshop's automated deployment and integrated testing features that you will have an opportunity to explore in more detail in Exercise #3.

- Create and load the sample databases using the *Admin* Web service. In the Workshop Test Browser, invoke the *Setup* method by clicking on the **Setup** button as shown in **Figure 9**. This method will create and load the Customers and Orders database tables with sample data that you will use for the duration of the evaluation. The *Setup* method leverages Java Controls for connecting with these two databases; you will learn more about Java Controls in subsequent exercises as well.
 - Note: If you are running this example for the second time and want to clean out old test data, first select the *Reset* method, then perform the *Setup* function.



Figure 9. The Workshop Test Browser displays the convenient test client interface

-
- After the *Setup* method has processed, you can trace the actions that were invoked on the Customer and Order Databases by reading the events described in the Workshop Test Browser. When finished, close the Workshop Test Browser. Optionally you could click on the **Test Operations** link within the test client to execute another *Admin.jws* Web service method such as *getCustomer* or *getOrders*.
 - Back in the BEA WebLogic Workshop Design View, close the *Admin.jws* file (click on the X in the top-right corner) to clear the workspace, and now you are ready to move on to building a custom Java Control in Exercise #2.

Exercise #2. Building a Custom Java Control

Java Controls are a central element in the Workshop programming model and enable developers to easily connect with existing data, systems, applications, and business logic. Java Controls are essentially business logic components that have methods, events, and properties and let you visually and declaratively specify behavior, focus on handling events, and calling methods, instead of writing complex J2EE API-intensive code.

Built-In Java Controls

To see a Java Control in action, let's explore the Database Controls for Avitek resources that were pre-defined in the Order Entry Demo template.

- In the Application Window, navigate into the *CRMSystem* folder and double-click on the *CustomerDB.jcx* file. This loads the Avitek Customer Database control in the Design View.

As shown in **Figure 10**, the *CustomerDB* control, like all Java Controls, is viewable via a visual representation in the Design View that shows its methods and events on the left-side of the canvas.

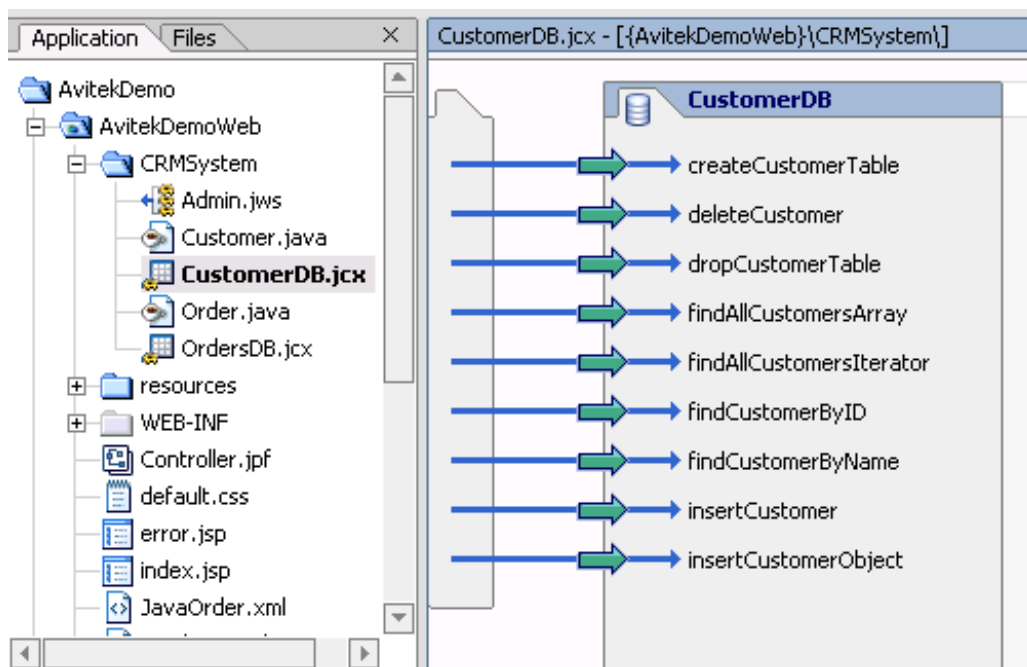


Figure 10. The *CustomerDB* Java Control depicted visually in the Design View

To better understand how the database control allows developers to easily leverage any SQL database without knowing JDBC API commands, try the following:

- Double-click on the arrow icon associated with the *insertCustomerObject* method. This brings up a dialog box (shown in **Figure 11**) in which any developer can simply map SQL parameters to the corresponding Java method definition. This is an example of how a Java Control can hide the complexities of J2EE from the developer – most application developers know SQL, but many do not know (or want to know) JDBC. Hit **Cancel** when you are done exploring the *SQL/Interface Editor*.

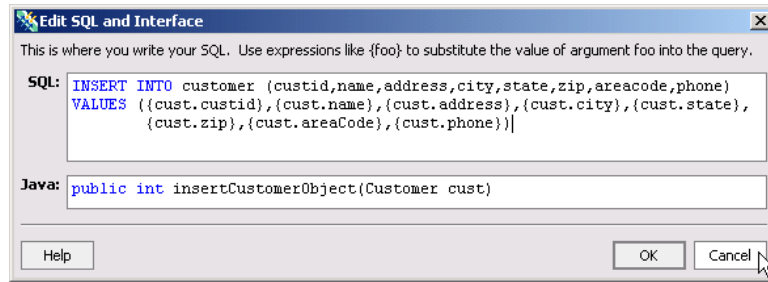


Figure 11: Explore the *SQL and Interface Editor* to map SQL statements to Java methods

The Database Control is one of the standard Java Controls that come pre-packaged with WebLogic Workshop 8.1. Other built-in Java Controls provide easy access to major enterprise resources such as EJB components, Web services, and JMS queues. All these controls are consistent in their ease-of-use and ability to prevent developers from needing to write complex, object-oriented J2EE infrastructure code to accomplish necessary integration objectives. You can see the controls that are built-in to Workshop by clicking on the New button in the data palette.

Custom Java Controls

Beyond the built-in Java Controls, WebLogic Workshop 8.1 offers an extensible controls architecture so that developers have the ability to create reusable components of business logic that can integrate with any enterprise resource, ISV application, or piece of business logic. Custom Java Controls can leverage built-in Java Controls and are themselves nestable and easily reusable. Moreover, any Java Control can be incorporated in any and all projects spanning across Web services, Web applications, and other WebLogic Platform applications.

To create a custom Java Control for the CRM System:

- Close the *CustomerDB* control in the Design View. To get started with building a new control, right-click on the *CRMSystem* project folder in the Application Window, and select **New → Java Control**.

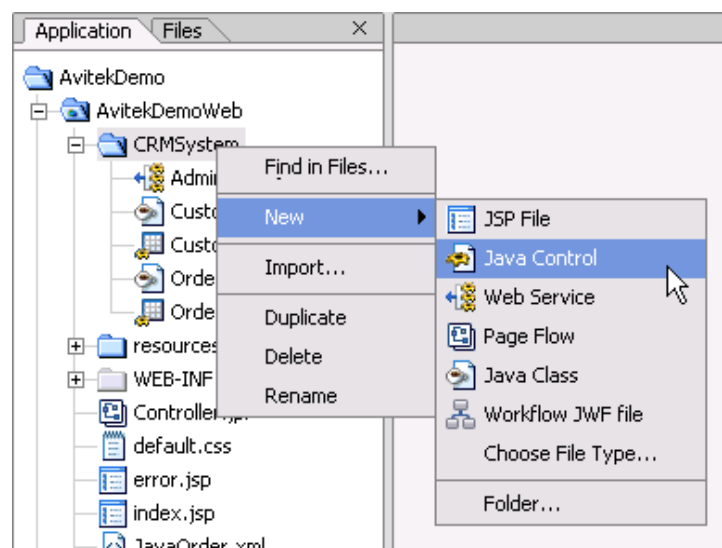


Figure 12. Create a new Java Control.

- In the *New Control* dialog box, insert the name *CRMControl.jcs* and click **Create**. This will result in a new custom Java Control being displayed in the Design View. Since it has no components, it just shows up as an empty canvas for now.

The purpose of the new custom Java Control is to provide a single component that will in turn handle interfacing with existing Avitek database systems that are involved in the order management process. In particular, any Order Entry system that you build will need to communicate with the Customer Database and the Order Database. As mentioned above, you already have access to pre-defined Java Controls that will help facilitate this integration. Once this control has been built, it can be leveraged easily from any other WebLogic Platform application.

- To start building the *CRMControl*, you will first add the back-end resources that it will require. You can do so by simply dragging-and-dropping the *CustomerDB.jcx* item from the Application Window on to the *CRMControl* icon on the Design View canvas.

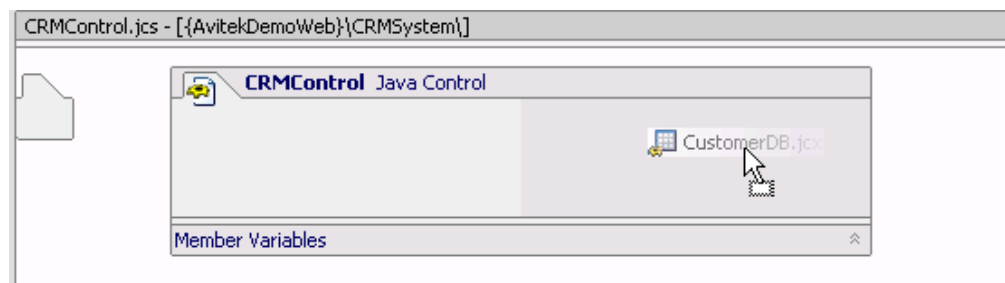


Figure 13. Drag-and-drop the Customer Database control to the CRMControl canvas

- Similarly, drag-and-drop the *OrdersDB.jcx* item from the Application Window to the *CRMControl* canvas.

With just these two simple drag-and-drop operations, you have already completed the process of integrating your custom *CRMControl* with Avitek's databases – without writing a line of code! The Design View will reflect these additions as shown in **Figure 14**.

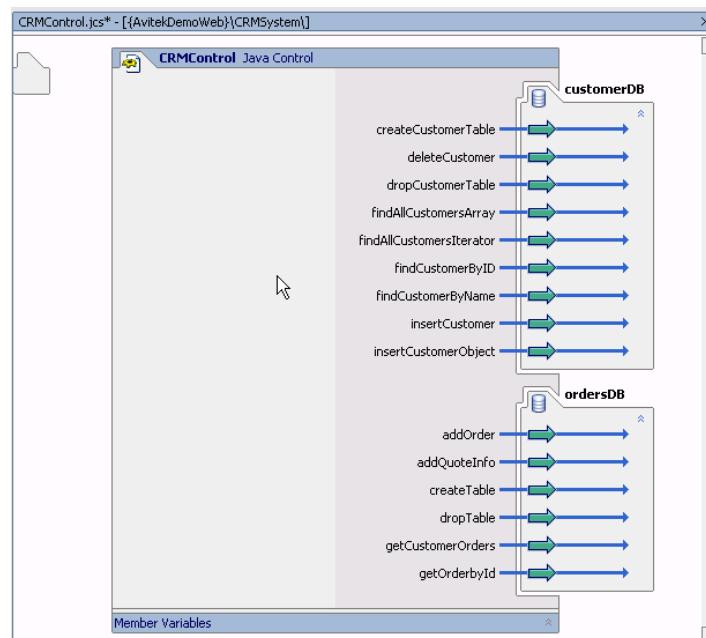


Figure 14. The Design View reflects database control additions to CRMControl

Next, you need to specify the methods a client may use to invoke the *CRMControl*. Workshop makes this step equally easy by automatically populating the Data Palette window with methods associated with the Java Controls that have been added to the project. These “pass-through” methods are immediately available to provide a client with access to any control’s methods.

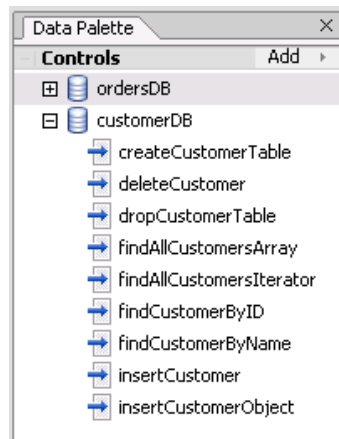


Figure 15. Listing of “pass-through” methods in the Data Palette

- To provide the desired functionality for *CRMControl*, you need to provide access to three methods of the *Customer Database* control: *findAllCustomersArray*, *findCustomerById*, and *insertCustomerObject*. Simply find the matching entries in the Data Palette (shown in **Figure 15**) and drag each from the Data Palette on to the *CRMControl* in the Design View. This provides a quick and easy mechanism to extend the existing functionality available in the control. These methods can be later customized if additional business logic is required.

The *CRMControl* will be updated in the Design View as shown below in **Figure 16**:

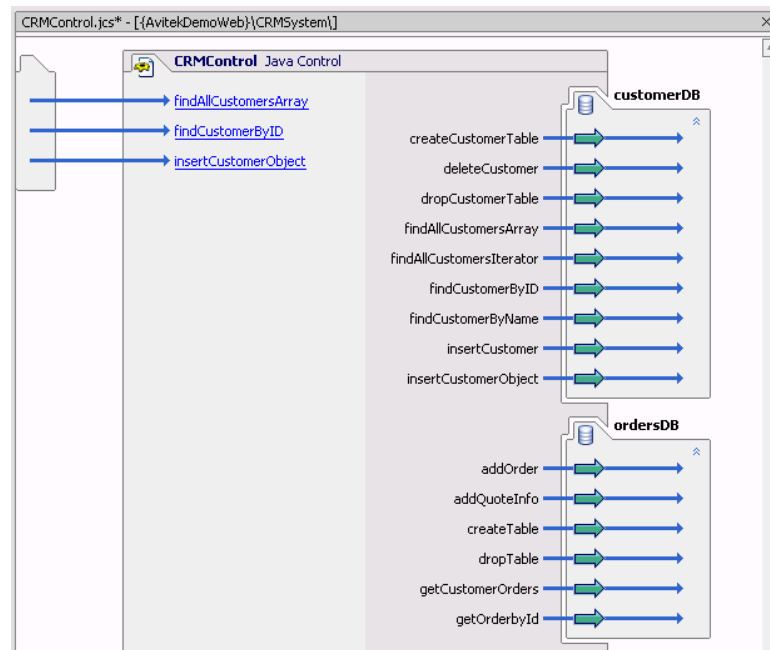


Figure 16. Visual depiction of *CRMControl* after adding “pass-through” methods

In addition to the three pass-through methods, you need to add a new method that allows a client of the custom *CRMControl* to submit a new order and do the appropriate processing with the Customer and Order databases.

- To create a new method, drag the *Method* icon from the Palette window (lower left corner) onto the *CRMControl* in the Design View. Specify the name of the new method as *createOrder*.

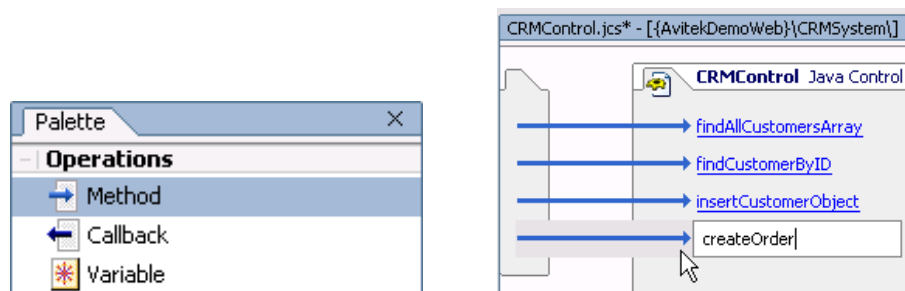


Figure 17. Implement the createOrder Method with drag-and-drop convenience

The *createOrder* method will need to accept information about an order and its associated customer, proceed to create a new customer record in the Customer database if an appropriate record does not already exist, and finally add the relevant order details into the Order database. Implementing this desired business logic and integrating with Avitek's databases will naturally require some code:

- From the Design View, click on the *createOrder* hyperlink to view this method in the Source View.



Figure 18. Hyperlinks make it easy to switch from Design View to Source View

- The Source View provides a convenient environment for writing the necessary Java code directly to the source file. You need to add the following code for the *createOrder* method:

```
public String createOrder(Customer c, Order o)
{
    java.util.Random r = new java.util.Random();
    if (customerDB.findCustomerByID(c.custId) == null)
    {
        c.custId = r.nextInt(100000);
        customerDB.insertCustomer(c.custId, c.name,
            c.address, c.city, c.state, c.zip, c.areaCode,
            c.phone);
    }
    ordersDB.addOrder(c.custId, r.nextInt(100000),
        o.cameraQty, o.flatScreenQty);
    return "Thank you for your order, " + c.name +
        ". Order processed for " + o.flatScreenQty +
        " Flat Screen TV sets and " + o.cameraQty +
        " Digital cameras.";
}
```


When finished typing in the code, the method should appear as shown below:

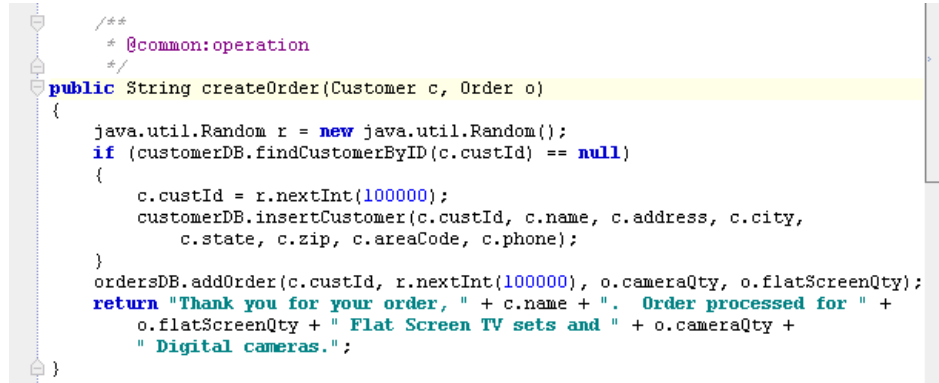
The image shows a code editor window with a source view of a Java method named `createOrder`. The method signature is `public String createOrder(Customer c, Order o)`. The code inside the method block includes a comment `/** @common:operation */`, a random number generator `java.util.Random r = new java.util.Random();`, a check for a customer in the database `if (customerDB.findCustomerByID(c.custId) == null)`, a loop to generate a customer ID and insert the customer into the database, and then adding an order to the database. The method returns a string message: `return "Thank you for your order, " + c.name + ". Order processed for " + o.flatScreenQty + " Flat Screen TV sets and " + o.cameraQty + " Digital cameras.";`

Figure 19. The `createOrder` method viewed in the Source View

Note that the functionality required from *CRMControl* was easily written in a few lines of procedural Java code and that communicating with the back-end databases was similar to working with any local Java class; as shown here, the developer can always abstract from the low-level J2EE complexity to maximize developer productivity.

Asynchronous Controls

Because Avitek may need to process large volumes of orders simultaneously, you want to make sure the *CRMControl* can capture all of these requests reliably, regardless of load. After all, if Avitek loses order requests, they lose business! A good architectural approach in these scenarios is to implement an asynchronous solution. For example, rather than having an external client submit an order request and just wait indefinitely (and possibly unreliably) for order processing to complete, the *CRMControl* places all incoming order requests on a reliable queue, where it is guaranteed to remain until the *CRMControl* is ready to process the request.

Supporting asynchronous communication is an important component of building scalable and reliable enterprise-class applications, however it has typically been a very arduous task due to the need for complex programming and a required understanding of J2EE technologies such as Message-Driven Beans (MDB) and JMS Queues. Fortunately BEA WebLogic Workshop 8.1 makes implementing asynchrony as simple as setting a property. To demonstrate, you will now create a new method that will offer all clients asynchronous access to the custom *CRMControl*.

- From the Design View, insert a new method by dragging the *Method* icon from the Palette window onto the *CRMControl* in the Design View. This time, specify the name of the new method as `createOrderAsynch`.
- With the `createOrderAsynch` method active (click on the arrow icon associated with the method if it is not already active), access the Property Editor and simply set the *Message-Buffer Enable* property to **True** as shown in Figure 20.

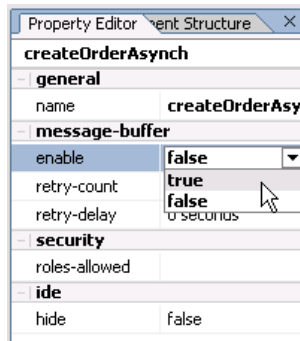


Figure 20. Use the Property Editor to enable message-buffering

You will notice that the Design View is automatically updated with a queue icon in front of *createOrderAsynch* to indicate that this method will communicate in asynchronous mode.



Figure 21. The Design View adds a queue icon to reflect asynchronous mode

As a result, the *createOrderAsynch* method can not simply respond to the calling client as the *CreateOrder* method did. Instead, you will create a Callback that will allow the *CRMControl* to notify the client after *createOrderAsynch* has finished processing the submitted order.

- To create a new callback, drag the *Callback* icon from the Palette window onto the *CRMControl* in the Design View. Specify the name of the new callback as *orderResponse*.

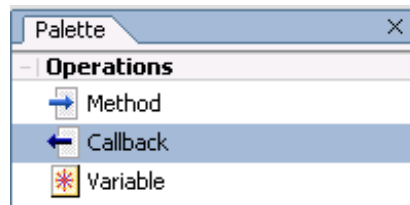


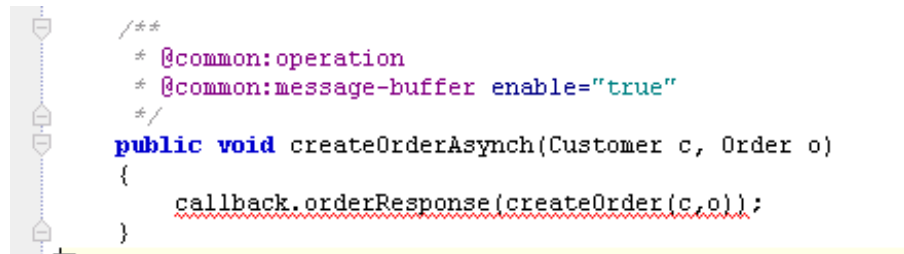
Figure 22. Create a new Callback by dragging-and-dropping the appropriate icon from the Palette Window

As you did previously with the *CreateOrder* method, you will need to also provide the business logic for the *createOrderAsynch* method.

- Switch to the Source View by clicking on the *createOrderAsynch* hyperlink.
- Fortunately the business logic required for *createOrderAsynch* mirrors the code already provided for *createOrder*, so you just need to modify the following two lines of code:

```
public void createOrderAsynch(Customer c, Order o)
{
    callback.orderResponse(createOrder(c,o));
}
```

After making these edits, you will likely notice that the last line was underlined in red, indicating that there is an error in the source code, as shown in **Figure 23**.



```
/**
 * @common:operation
 * @common:message-buffer enable="true"
 */
public void createOrderAsynch(Customer c, Order o)
{
    callback.orderResponse(createOrder(c,o));
}
```

Figure 23. The automatic syntax checker provides real-time alerts to source code errors

Workshop's in-line syntax checking and notification makes it easy for developers to fix mistakes while they are coding in the same context rather than waiting until compile-time. The reason for this particular error is that your invocation of the *orderResponse* callback doesn't match the declaration that was automatically created by Workshop when you added the *orderResponse* operation in the Design View.

- To make this simple fix, scroll down in the Source View to the *orderResponse* declaration, and add a parameter named *message* of type *String*. When you are done the declaration will look like this:

```
public interface Callback
{
    void orderResponse(String message);
}
```

- Switch back to the Design View to see the completed *CRMControl*.

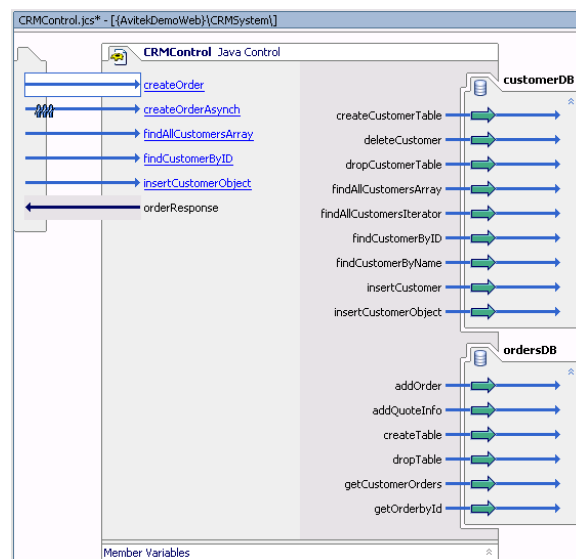


Figure 24. View the completed *CRMControl* in the Design View

In the few steps above, you have created a distributable and reusable component of business logic that provides both synchronous and asynchronous access to the Avitek order management databases. In the following exercises, you will build a Web service and a Web application which will both take advantage of the *CRMControl* you just created.

Exercise #3. Creating an Enterprise-Class Web Service

The custom *CRMControl* you implemented in Exercise #2 accepts customer and order information and does all the necessary back-end processing and database coordination necessary for Avitek's systems. However, you have not yet provided a mechanism for external users to access the *CRMControl* interface. In this exercise, you will expose the *CRMControl*'s functionality via an enterprise-class Web service application that will instantly allow thousands of business partners to interoperate with Avitek's Order Entry system.

- First, clear the workspace by saving and closing all open files.
- To create a new Web service, right-click on the *AvitekDemoWeb* project icon in the Application Window, select **New** → **Web Service**. Name this web service *orderEntryService.jws*.

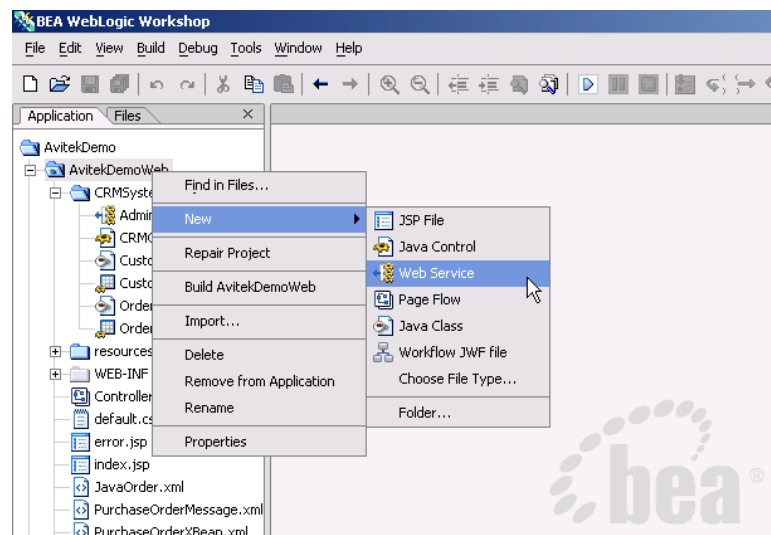


Figure 25. Create a new Web service named *orderEntryService.jws*

As you might expect, you will now see a graphical depiction of the *orderEntryService* application in the Design View, which for now is just an empty canvas. Generally, if you wished to incorporate any built-in Java Controls in a Web service application, you could just select the appropriate control option from the Control Palette. Thanks to your work in the previous exercise, you will only need to leverage the custom *CRMControl* in this *orderEntryService* Web service.

- To leverage the *CRMControl*, select its icon from the Application Window and drag it on to the *orderEntryService* canvas in the Design View.



Figure 26. To add access to *CRMControl*, simply drag its icon on to the *orderEntryService.jws* canvas

The *CRMControl* now appears on the right-side of the Design View canvas, indicating that it is available as a resource for the developer to easily leverage.



Figure 27. The Design View shows *CRMControl* as a back-end resource in *orderEntryService.jws*

Next, you need to create methods which will be invocable by clients of this Web service application. Once again, this is made very easy for you because BEA WebLogic Workshop 8.1 automatically generates “pass-through” methods for accessing the *CRMControl*.

- To add the desired methods, simply drag the *createOrderAsynch* and *orderResponse* entries from the Data Palette to the *orderEntryService* canvas. These pass-through methods allow for convenient exposure of the *CRMControl* methods via this Web service application.

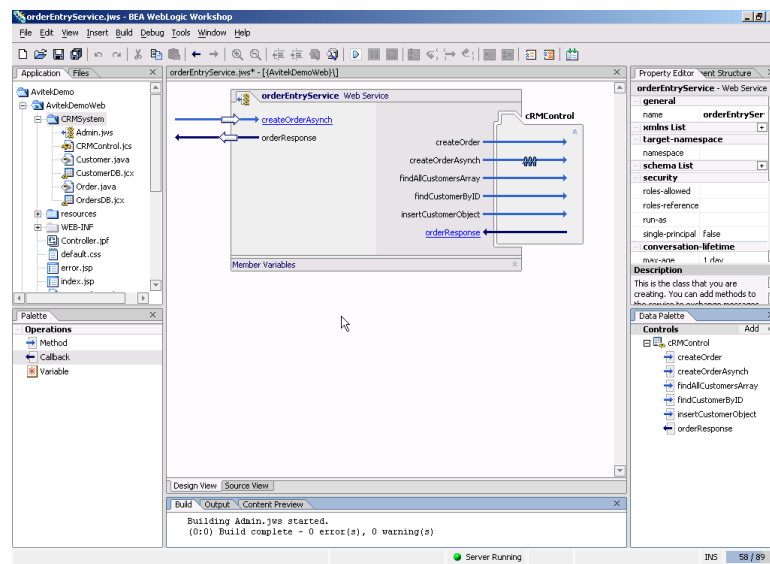


Figure 28. The *createOrderAsynch* and *orderResponse* operations are exposed to clients

With the client-facing methods and back-end resources in place, now is a good opportunity to explore additional features in WebLogic Workshop 8.1 that help to build enterprise-class Web services. For example, asynchrony, loose coupling, strong message-level security, and reliable “once and only once” messaging are all very important features for building reliable and scalable Web services that help customers solve real business problems. Not surprisingly, Avitek would want to take advantage of some of these advanced architectural features.

Support for Asynchronous Web Services

In the real world, most Web services applications cannot immediately return a response. They may need to access a legacy system, contact a third party, or involve a human user. These operations are asynchronous in nature, so it is important to make support for asynchronous Web services as easy as possible. With BEA WebLogic Workshop 8.1, making the `orderEntryService` asynchronous is as simple as setting a few properties:

- With the `createOrderAsynch` method active in the Design View (if necessary, activate it by clicking on the arrow icon for the method), access the Property Editor to see the various customizations available. Set the *Message Buffer Enable* property to **True**. This will result in a queue icon being added to the method's visual representation.

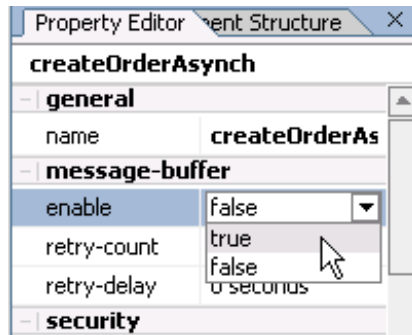


Figure 29. Use the Property Editor to declaratively specify complex application functionality

As you already saw in the previous exercise, enabling the message-buffer property will cause for a JMS queue and the associated messaging infrastructure to be implemented at run-time. Note that in this case, the queue is established between the `orderEntryService` application and its clients, whereas in the previous exercise, the `CRMControl` implemented an asynchronous method for interaction between the `CRMControl` and its clients, which in this case is the `orderEntryService` application. As a result, there are effectively two levels of asynchrony for optimal application scalability and efficiency.

- With the `createOrderAsynch` method still active, locate the *Conversation* section in the Property Editor, and set the phase to **Start**. This setting will be reflected with a green triangle added to the method's visual representation.
- Finally, click on the `orderResponse` callback in the Design View, and change its Conversation phase to **Finish**. This setting will be reflected with a red square added to the callback's visual representation.

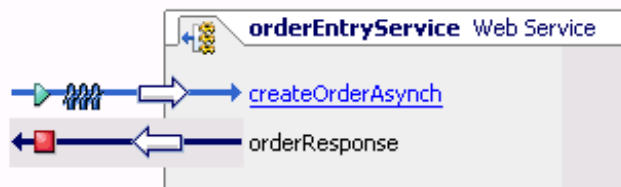


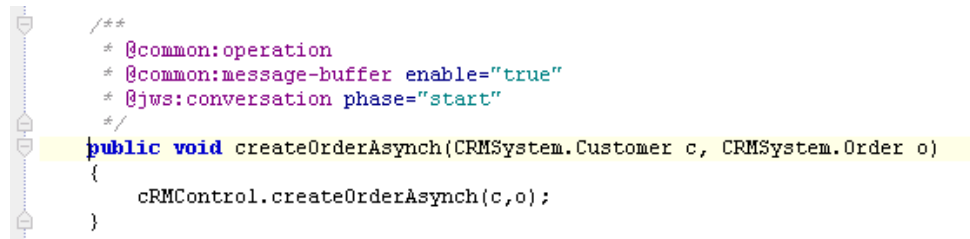
Figure 30. Use WebLogic Workshop's conversation metaphor to manage asynchronous communications

These conversation settings are based on Workshop's conversation metaphor for asynchronous communications. These settings help the application keep track of messages flowing back and forth and make sure that all the messages that are part of the same conversation go to the right place (i.e. the calling client). Moreover, conversations will automatically store any state associated with the conversation in a robust reliable way (using entity EJBs) – developers simply need to declare variables in a JWS class to take advantage of this capability. By establishing the

conversational nature of these methods using declarative property settings, BEA WebLogic Workshop 8.1 automatically handles the details of state management, message correlation, and more.

The BEA WebLogic Workshop™ Property Editor is significant because it does more than just let developers change simple values in a visual way. Properties like conversation phases and message buffering expose sophisticated functionality that would normally require writing 40-80 lines of Java code and understanding the JMS and JNDI APIs. Properties make these powerful J2EE concepts accessible to all developers regardless of J2EE expertise level. To better understand how Workshop can deliver so much power so simply, you should take a minute to see how the property settings are captured:

- From the Design View, click on the *CreateOrderAsynch* hyperlink to switch to the Source View.



```
/**
 * @common:operation
 * @common:message-buffer enable="true"
 * @jws:conversation phase="start"
 */
public void createOrderAsynch(CRMSystem.Customer c, CRMSystem.Order o)
{
    cRMControl.createOrderAsynch(c,o);
}
```

Figure 31. The Source View is synchronized for seamless two-way editing

Notice the source code annotations that are located above the method's declaration, within the JavaDoc annotations. It is these code annotation that get interpreted by the Workshop run-time framework at deployment-time; WebLogic Workshop then automatically enables this functionality in the resulting application by implementing standard J2EE components. This remarkable capability, coupled with the event-based visual design environment, is what makes it easy for all developers, not just J2EE experts, to work productively on the WebLogic Enterprise Platform.

It should be noted that WebLogic Workshop does not generate code – the developer never needs to understand or debug code they have not written. Instead, properties and their resulting annotations declare functionality that is supported by the run-time framework in a container approach.

In just these few steps, you have now created a powerful Web service application that Avitek can immediately deploy to the universe of distributors to enable automated ordering over existing Internet infrastructure. Note that the innovative WebLogic Workshop programming model and run-time framework abstracted you from the minutiae of SOAP marshalling, Java-XML mapping, bean deployment, WSDL file creation, and much more. In this manner, developers are free to focus on handling events and writing business logic, making them dramatically more productive.

Automated Deployment and Testing

You are now ready to test the *orderEntryService*:

- To automatically deploy and run the Web service, uses the menus to select **Debug** → **Start**, or hit the **Start** icon on the menu bar.

When you start the *orderEntryService* application, BEA WebLogic Workshop™ 8.1 automatically compiles and deploys the Web service application to the WebLogic Server. Note that you do not need to worry about writing deployment descriptors, configuring the underlying application server, or packaging your application components. Instead, the BEA WebLogic Workshop “Write and Run” feature enables developers to build and deploy applications quickly, dramatically reducing the time required for iterative development.

Because Web services typically involve application-to-application communication, testing Web services at development time has traditionally been tedious and time consuming. In fact,

normally you would have to write client-side code to test the web service. But in order to simplify testing, BEA WebLogic Workshop provides a browser-based interface to test all applications so functionality can be verified and problems can be quickly identified. The integrated test harness contributes to WebLogic Workshop's model for rapid application development.

- From within the Workshop Test Browser, click on the **Overview** tab.

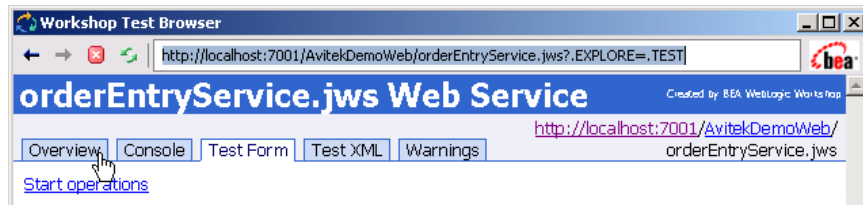


Figure 32. WebLogic Workshop 8.1 generates a test client to quickly test any application

The Overview screen in the Workshop Test Browser automatically provides a basic description of the Web service and the methods it exposes. It also allows clients to easily download a WSDL file, source code for a Java Control to access the *orderEntryService* application, or Java proxy code to use in any Java environment for building a client for this Web service. Moreover, clicking on the Console tab will enable the developer to see what types of services and components this Web service is accessing.

- To test the *orderEntryService* application, select the **Test XML** tab.

The Workshop Test Browser provides a free-text box that is pre-loaded with the skeleton structure of the XML message that the *orderEntryService* expects as incoming order information.

- You can either manually type in values into the Workshop Test Browser, or more conveniently, paste in the sample order information provided for you in the *JavaOrder.xml* file. You can locate this file in the Application Window project tree. Click the *createOrderAsynch* button.

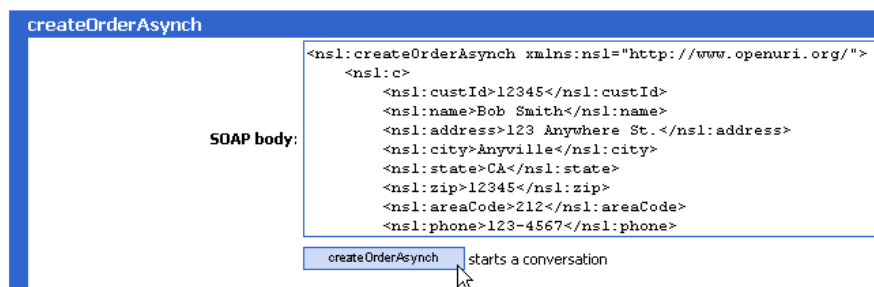


Figure 33. Submit a sample order for a customer named Bob Smith

The test client automatically wraps up the order information into an appropriate SOAP message as shown in the test client. Embedded in the XML is the sample customer and order information:

Service Request createOrderAsynch
Submitted at Sun Feb 23 20:03:15 PST 2003

```

<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header>
    <StartHeader xmlns="http://www.openuri.org/2002/04/soap/conversation/">
      <conversationID>1046059394904</conversationID>
      <callbackLocation>http://TESTUI</callbackLocation>
    </StartHeader>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <ns1:createOrderAsynch xmlns:ns1="http://www.openuri.org/">
      <ns1:c>
        <ns1:custId>12345</ns1:custId>
        <ns1:name>Bob Smith</ns1:name>
        <ns1:address>123 Anywhere St.</ns1:address>
        <ns1:city>Anyville</ns1:city>
        <ns1:state>CA</ns1:state>
        <ns1:zip>12345</ns1:zip>
        <ns1:areaCode>212</ns1:areaCode>
        <ns1:phone>123-4567</ns1:phone>
      </ns1:c>
      <ns1:o>
        <ns1:orderID>558488</ns1:orderID>
        <ns1:custID>12345</ns1:custID>
        <ns1:cameraQty>15</ns1:cameraQty>
        <ns1:flatScreenQty>8</ns1:flatScreenQty>
      </ns1:o>
    </ns1:createOrderAsynch>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

Service Response
Submitted at Sun Feb 23 20:03:16 PST 2003

Figure 34. The Workshop Test Browser helps track all the messages used in testing the Web service

Recall that the *createOrderAsynch* method you are testing is asynchronous, and accordingly, it starts a conversation, as indicated at design-time. Not surprisingly, the Message Log shows that a new conversation has automatically been started. BEA WebLogic Workshop will automatically track all related messages under the corresponding Conversation ID, which is visible in your Message Log as well as in the SOAP headers of the messages being sent.

- Hit the **Refresh** link in the Workshop Test Browser to view any subsequent messages that have been exchanged between the test client and your Web service during this asynchronous conversation. Soon, you will see the order confirmation appear from the *orderResponse* client callback, as shown in **Figure 35**.

Message Log **Refresh**
1046059394904
→ createOrderAsynch
← **callback.orderResponse**
Conversation 1046059394904 is finished.
Clear Log

Client Callback
Submitted at Sun Feb 23 20:03:18 PST 2003

```

<SOAP-ENV:Envelope xmlns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <SOAP-ENV:Header>
    <CallbackHeader xmlns="http://www.openuri.org/2002/04/soap/conversation/">
      <conversationID>1046059394904</conversationID>
    </CallbackHeader>
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    <orderResponse xmlns="http://www.openuri.org/">
      <message><b>Thank you for your order, Bob Smith. Order processed for 8 Flat Screen TV sets and 15 Digital
cameras.</b></message>
    </orderResponse>
  </SOAP-ENV:Body>
</SOAP-ENV:Envelope>

```

No Response
Submitted at Sun Feb 23 20:03:18 PST 2003
The original client is the Test User Interface

Figure 35. Asynchronous messages are correlated according to the Conversation ID

- After confirming the order is acknowledged in the test client, close the Workshop Test Browser and return to the WebLogic Workshop visual development environment.

Loose-Coupling

Another important feature of BEA's architecture for enterprise-class Web services is loose coupling. Loosely-coupled applications are flexible and can accommodate change over time, which is especially important when Web services are used to integrate different applications that are normally built by different teams using different technologies and platforms. The key to loose-coupling is to enable a separation between a Web service's public contract and its private implementation. Since Web services communicate using XML-based documents and WebLogic Workshop application development is done in Java, it is critical to provide easy, intelligent and flexible mapping between XML and Java to ensure true loose-coupling.

To help with loose coupling, BEA WebLogic Workshop 8.1 provides deep support for XML Schema and XQuery, state-of-the-art technologies for working with XML in Java. In the next few steps, you will incorporate a schema definition that provides the structure and constraints on the Purchase Order messages that business partners will use to submit automated orders to Avitek. You will also gain exposure to XQuery technology, which is sometimes referred to as "SQL for XML" because it provides a robust mechanism to access data directly from XML documents, much like SQL provides a mechanism for accessing data in traditional databases. Fortunately, BEA WebLogic Workshop handles for you the complex work of writing custom XQuery expressions.

- You need to first import the appropriate schema file into the *AvitekDemo* application. In the Applications Window, right-click on the Schemas folder, and choose **Import**. Browse to the *purchase-order.xsd* file that you previously saved in the *c:\workshop81demo* folder, and click **Import**.

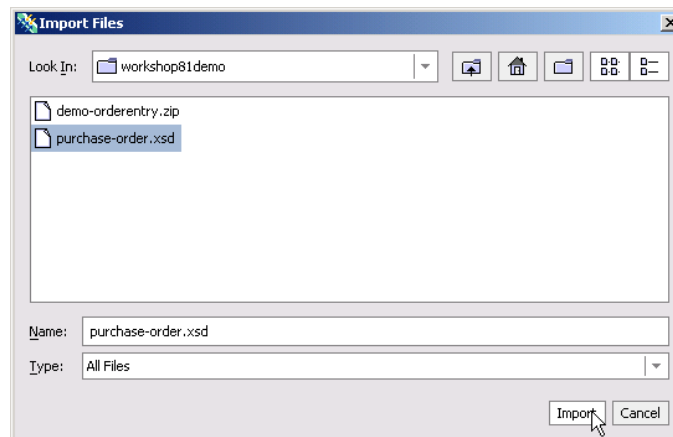


Figure 36. Import the XML Schema Definition (XSD) file to use in XQuery mapping

BEA WebLogic Workshop automatically compiles and interprets the provided schema file to learn the structure of messages expected for the *orderEntryService* application. (You can see the results of this compilation in the Build pane underneath the Design View.)

Next, you will need to provide the "mapping" of the data fields in the incoming XML message to elements in the Java data structures that will be used to hold the data for the application to process.

- From the Design View of *orderEntryService.jws*, click on the *createOrderAsynch* method's arrow icon to make this method active.
- Scroll down in the Property Editor until you see the *XQuery* property under the *Parameter-XML* heading. Click on the ... button to launch the *XQuery Editor* as shown in **Figure 37**.



Figure 37. Use the Property Editor to invoke the XQuery Mapping features

- In the left-side pane of the *XQuery Editor*, expand the tree structure to locate the *createOrderAsynch* schema entry as shown in **Figure 38**. You need to specify this definition as the “source schema” so highlight it and click **Create**.

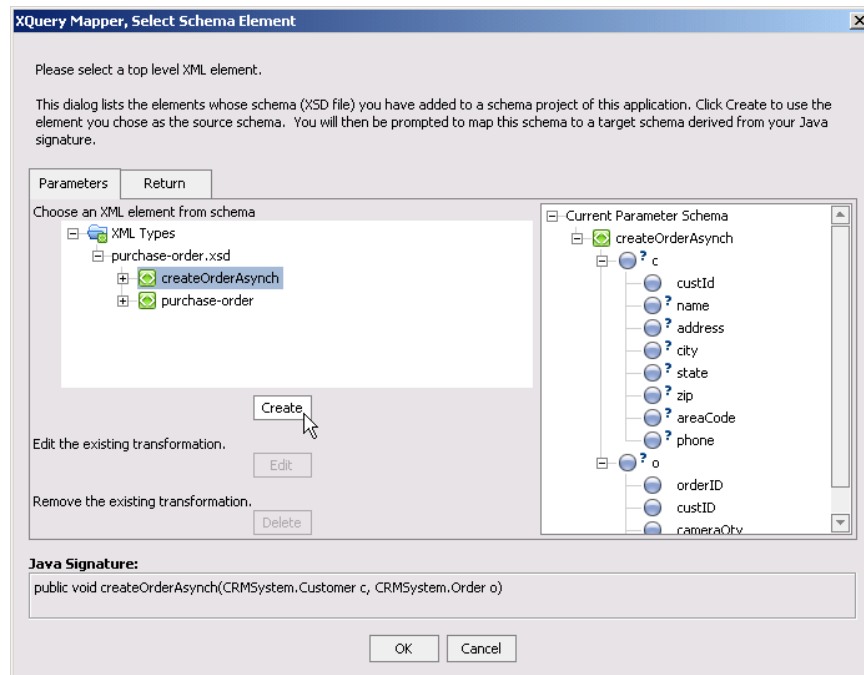


Figure 38. Specify the createOrderAsynch definition as the Source Schema

Next appears a visual mapping tool to visually map fields from the source schema (based on the chosen schema definition) to the target schema (the parameters defined by the *createOrderAsynch* method declaration which are the Customer and Order data structures). Mapping fields between the two schema is as simple as drawing lines connecting source fields to target fields:

- Connect *customer-number* first to *c.custId* and then to *o.custID*
- Connect *firstname* to *c.name* and *lastname* also to *c.name*
- Connect *address*, *city*, *state*, *zip*, and *phone* fields to the matching fields in the *c* data structure

- Connect *order-number* to *o.orderID*
- Connect *tv-order:quantity* to *o.flatScreenQty*
- Connect *camera-order:quantity* to *cameraQty*

When you are finished with these mappings, the XQuery Mapper screen should appear as shown in **Figure 39**.

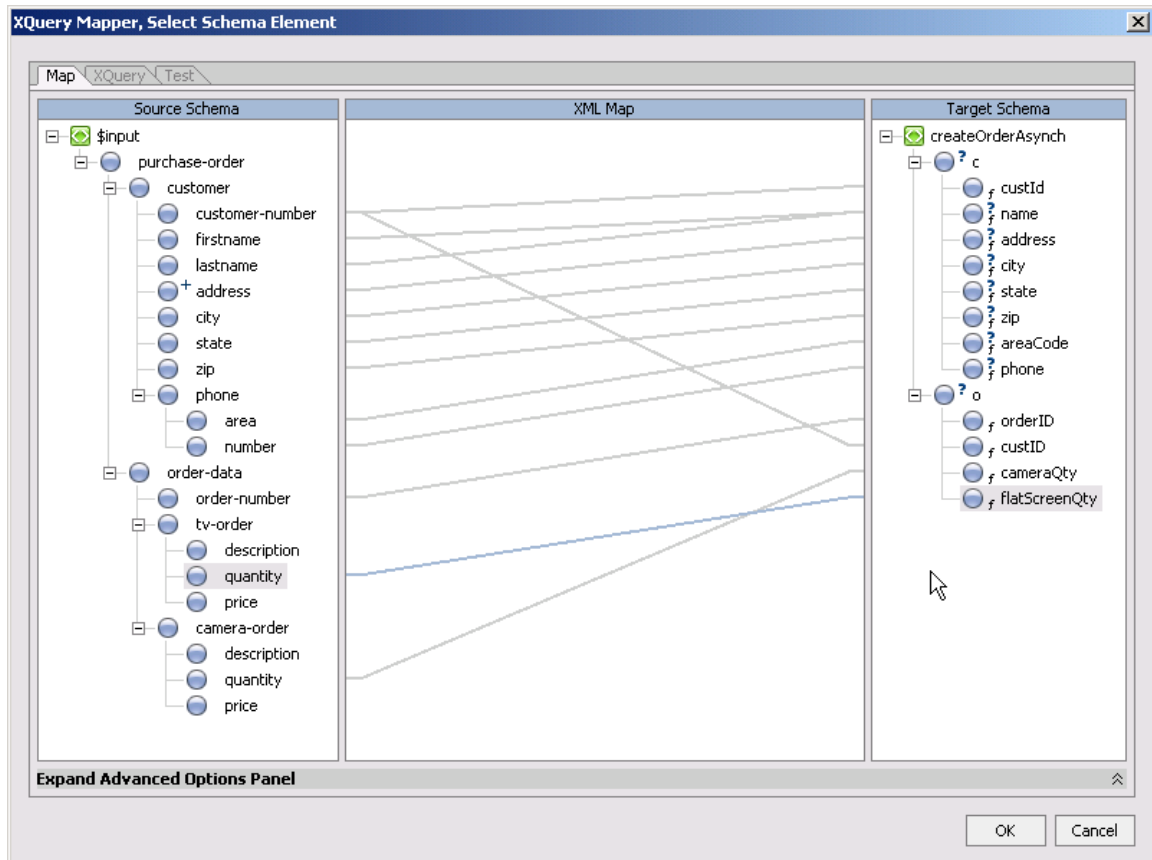


Figure 39. WebLogic Workshop enables visual mapping between Source (XML) and Target (Java)

You can use the visual tool for speed as shown above, or directly access the XQuery source code by clicking on the XQuery tab. Each of the visual mapping actions is translated into the matching XQuery expression.

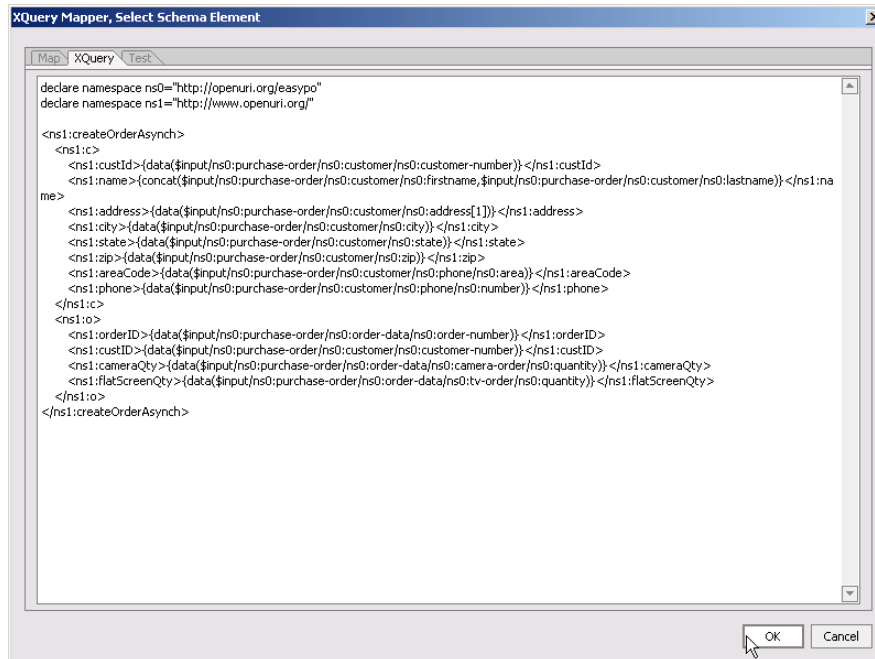


Figure 40. Workshop provides a code-view of the XQuery expressions underlying the mapping

- Select the *XQuery* tab at the top of the *XQuery Mapper* window to view the mapping in its native XQuery format. For aesthetic completeness, insert some whitespace into the *name* mapping by specifying a space (" ") between the first name and last name. The modified line should read as follows:

```
<ns1:name>{concat($input/ns0:purchase-
order/ns0:customer/ns0:firstname," ",$input/ns0:purchase-
order/ns0:customer/ns0:lastname)}</ns1:name>
```

- Click **OK** in the *XQuery Mapper* tool, and **OK** in the *XQuery Editor* to return to the Design View.

With these additions for ensuring a loosely-coupled Web service, you can now test your revised *orderEntryService* Web service application:

- As before, uses the menus to select **Debug** → **Start**, or hit the **Start** icon on the menu bar.
- Click on the **Test XML** tab. Notice that the XML skeleton provided now is slightly different from last time. This is because the *orderEntryService* application is now expecting an XML message that conforms to the *createOrderAsynch* schema definition you specified.
- For convenience, an XML message conforming to the expected schema has been provided in the *PurchaseOrderMessage.xml* file, accessible in the Application Window. Paste the full contents of this XML message into the SOAP body field of the test client and click *CreateOrderAsynch*.

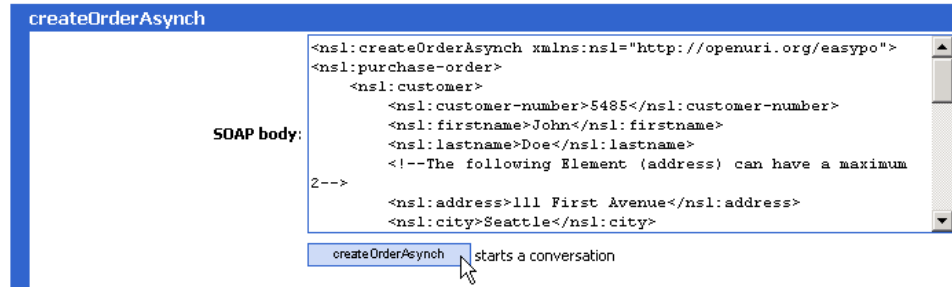


Figure 41. Submit a sample order for a customer named John Doe

- After invoking the asynchronous method, you will need to click the **Refresh** link in the Workshop Test Browser to collect subsequent messages in the conversation. Soon, you should receive the client callback in the Message Log which confirms the order.

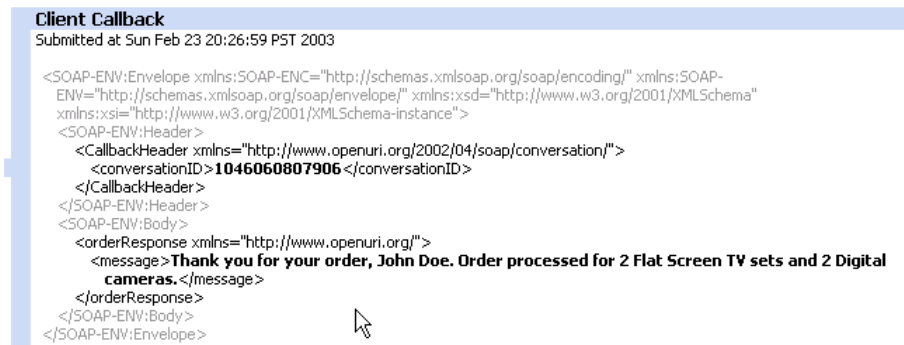


Figure 42. The order for John Doe is acknowledged by the *orderEntryService* application

Congratulations! You just built, deployed, and tested an enterprise-class Web service in a matter of minutes using BEA WebLogic Workshop 8.1!

Exercise #4. Creating an Enterprise-Class Web Application

In helping Avitek make its Order Entry System accessible to business partners, you've experienced WebLogic Workshop's capabilities for building powerful, enterprise-class Web services. Now, turn your attention towards creating a Web application to target the unlimited number of potential e-commerce customers. Fortunately, thanks to BEA WebLogic Workshop's unified development architecture, creating a Web application utilizes the same simplified programming model as you have already become familiar with from building the custom control and Web services application in the previous exercises. To get started with this exercise:

- If it is still active from the previous exercise, close the Workshop Test Browser to return to the WebLogic Workshop development environment. Then save and close all open files including *orderEntryService.jws* to clear your workspace.
- In the Application Window, locate the *Controller.jpf* file and double-click on its icon to open the file.

Java Page Flow Technology

The key requirement for using BEA WebLogic Workshop to build server-side applications with dynamic JSP/HTML user interfaces is understanding a new Java innovation introduced with WebLogic Workshop 8.1 – Java Page Flow. A Java Page Flow is simply a set of related JavaServer Pages and a companion Java controller class. A JavaServer Page (JSP) gives you a convenient way to present dynamically generated content via a web browser. JSPs are able to provide dynamic content because they are not purely HTML; instead, JSP pages include Java code that provides the dynamic content. And the Java controller class (denoted by a file's .JPF suffix) provides the navigational and behavioral control that spans across the related JSP pages. These components work together to help build enterprise-class, standards-based Web applications.

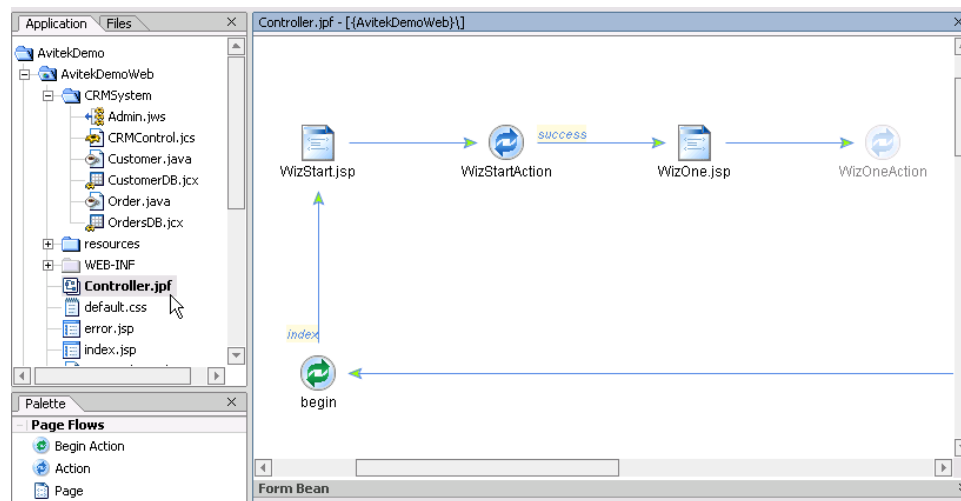


Figure 43. The Flow View provides a visual depiction of the Web application

You should see in front of you the *Controller.jpf* file being displayed via the default Flow View. The *Controller.jpf* file is the Java class that contains the business logic for the Web application

you need to complete. This file will include the necessary methods and code annotations to define the intended behavior of the Web application, including the navigational flow between pages, the actions (i.e. business logic) executed by pages, and the data shared between pages in the application. Page Flows can also leverage controls to access existing business logic and back-end resources. Importantly, the Java Page Flow programming approach provides a way to elegantly separate presentation code from navigational control logic. Presentation code can be placed where it belongs – in the JSP files. Navigational control logic can be implemented in each Java Page Flow's single configuration file – the JPF file. This separation of functionality offers a big advantage to development team members because they can make business logic revisions in a single JPF file, instead of having to search through every JSP file to make multiple updates.

For Avitek's needs, your goal is to build a web-based Order Entry System that allows customers to submit product orders. Of course, you will also need to collect the requisite customer information, handling the cases where a user could be a new customer or an existing customer.

Figure 44 diagrams the expected user experience for the Avitek Order Entry Web application:

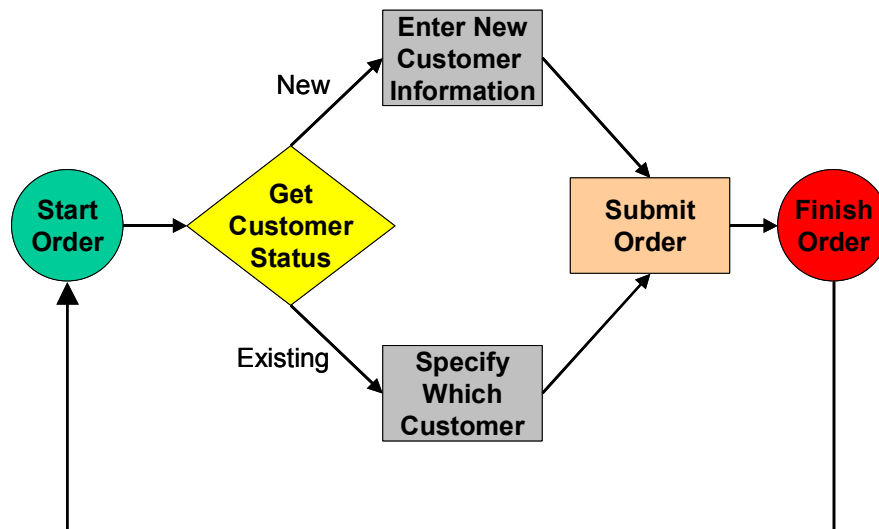


Figure 44. Process flow for the Avitek Web application

To create this Web application, you will benefit from the ability to re-use the *CRMControl* you built-in Exercise #2 to handle interfacing with Avitek's databases. Moreover, several components of the Web application have been pre-defined for you. In fact, the Flow View of the *Controller.jspf* provides a visual depiction of the pre-defined pages and actions, and should resemble the diagram in **Figure 44**.

To complete the Order Entry Web application, you will need to add some missing actions and provide the necessary business logic therein. In particular, notice in the Flow View the missing links between the *WizOne.jsp* icon and the *WizTwoNew.jsp/WizTwoExisting.jsp* icons.

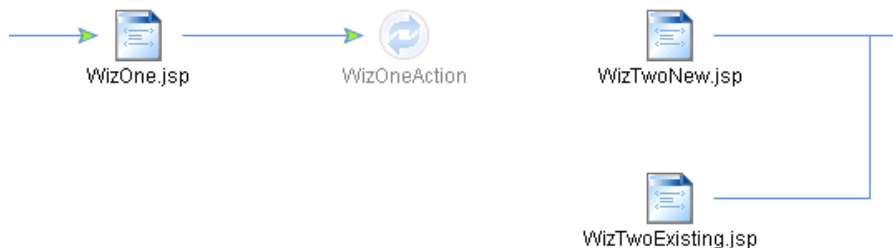


Figure 45. Notice the gaps remaining in the Java Page Flow application

These missing links represent the area of the Web application where you will need to focus your attention. As defined, the *WizOne.jsp* page allows the user to indicate whether they are a new or existing Avitek customer:

- Locate the *WizOne.jsp* in the Flow View and double-click on its icon to see how the page is defined. When finished, close the page by clicking on the X in the top-right corner.



Figure 46. *WizOne.jsp* allows a user to specify their customer status

Based on the user response provided by the radio button selection, the Web application needs to process some business logic and determine in which path the user should be taken. This functionality will be implemented in *WizOneAction*, which is a method in *Controller.jspf* that gets automatically invoked after the user submits a response in the *WizOne.jsp* page.

- Locate the *WizOneAction* icon in the Flow View, it is adjacent to the *WizOne.jsp* icon and should appear fainter than the other icons. To instantiate this action, right-click on the *WizOneAction* icon and choose **Create**.

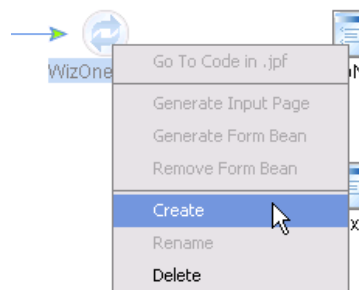


Figure 47. Activate the *WizOneAction* element using convenient pop-up menu commands

The next step is to define the Form Bean associated with *WizOneAction*. A Form Bean is the mechanism that communicates data between a page (i.e. *WizOne.jsp*) and its associated Action (i.e. *WizOneAction*). Since the *WizOne.jsp* page allows a user to identify oneself as a “new” or “existing” customer, you need the Form Bean associated with *WizOneAction* to hold this piece of information.

- Right-click on the *WizOneAction* icon, and select **Generate Form Bean**. This automatically brings up the *Form Bean Editor*.

- In the *Form Bean Editor*, click the **New Field** icon or otherwise click in the first row of the table to create a new field. Name the new field *customerType* and leave the type as the default String value. Click **OK**.

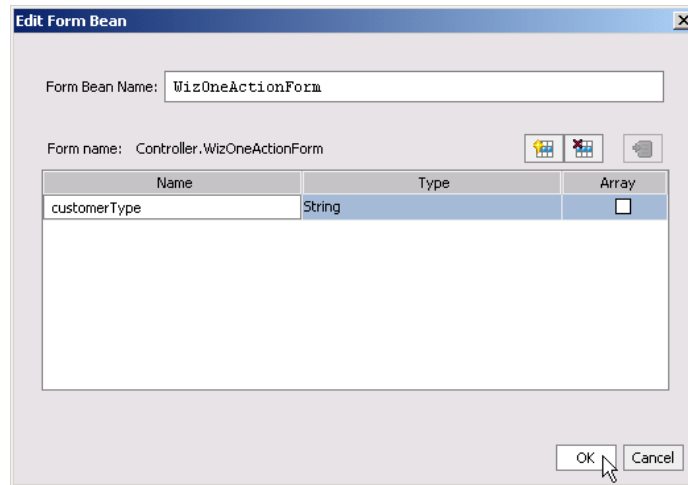


Figure 48. Add a String field to the Form Bean to capture the user's status

The Form Bean is now complete and will convey the *customerType* value to *WizOneAction*. Depending on the value (i.e. new or existing customer), *WizOneAction* will determine which page a particular user is taken to next. Accordingly, you need to visually connect the newly created *WizOneAction* with the subsequent pages in the application flow.

- Draw a link between *WizOneAction* and *WizTwoNew.jsp*. This is done by simply clicking just to the right of the *WizOneAction* icon (the mouse pointer should change to have a link underneath the arrow as shown in **Figure 49**) and drag a link (keep the mouse button depressed) towards *WizTwoNew.jsp*. Change the link's label (in the yellow box) to **new**.



Figure 49. Visually drag links to connect pages in the Flow View

- Similarly, drag a link from *WizOneAction* to *WizTwoExisting.jsp*, and label its link as **existing**.

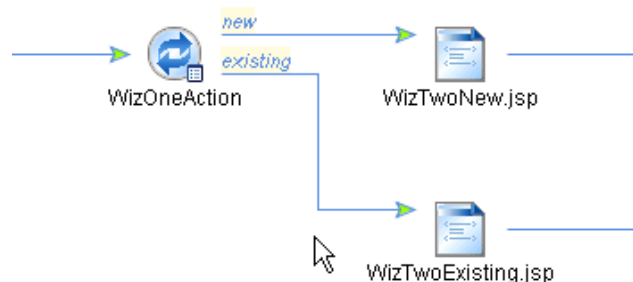


Figure 50. The Flow View displays the specified navigational logic

As you can see, the Flow View provides an easy and intuitive interface for specifying and viewing the overall navigational flow of your Order Entry Web application.

- With Web applications in WebLogic Workshop 8.1, you can also take advantage of the Action View by clicking on the corresponding tab near the bottom of the canvas.

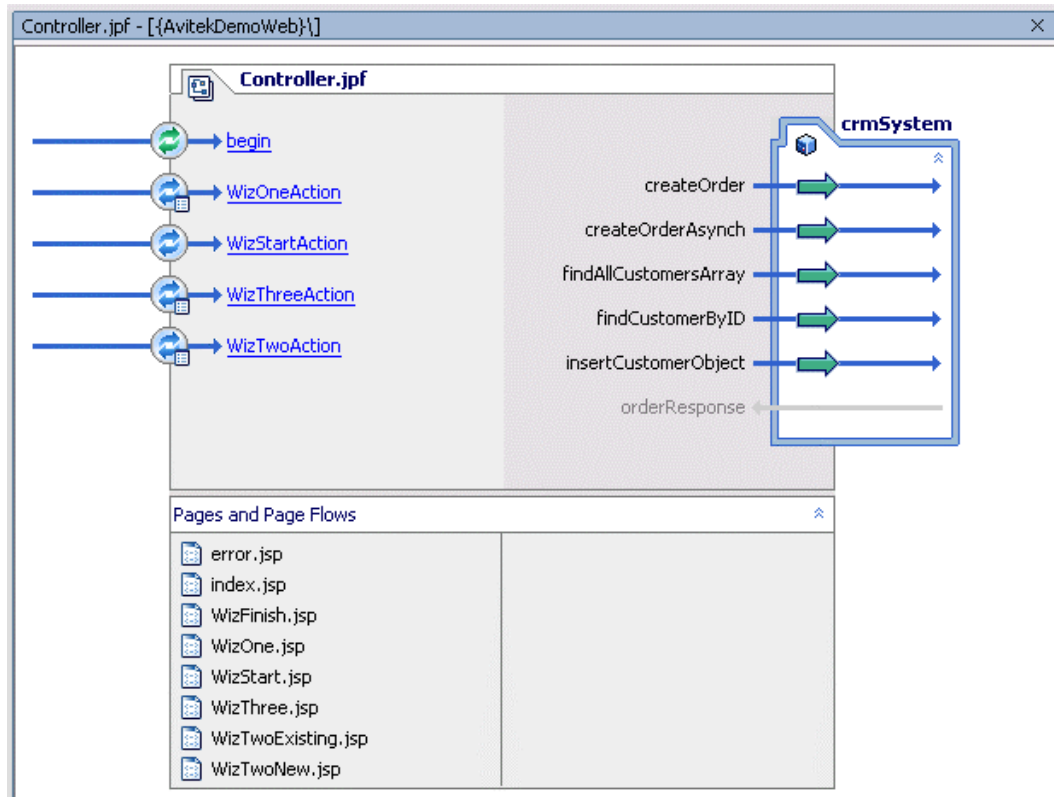


Figure 51. The Action View displays the methods and resources in Controller.jspf

The Action View should look very familiar to the Design View you experienced while building the custom control and the Web service: there are methods on the left-side and resources on the right-side. With a Web application, the methods represent the Actions that are defined to control the application's behavior. Clicking on any Action's name will take you to the synchronized code in the Source View:

- From the Action View, click on the *WizOneAction* hyperlink to switch over to the Source View and view the underlying Java code for this Action.

```
/**
 * @jpf:action
 * @jpf:forward name="new" path="WizTwoNew.jsp"
 * @jpf:forward name="existing" path="WizTwoExisting.jsp"
 */
protected Forward WizOneAction(WizOneActionForm form)
{
    return new Forward("success");
}
```

Figure 52. The code annotations are color-coded for enhanced visibility

Notice the code annotations, denoted with *@jpf* tags, above the *WizOneAction* method definition shown in **Figure 52**. These were automatically inserted by the development environment when you drew the visual links in the Flow View.

- Within the *WizOneAction* method, you need to replace the single line of default code with the following lines to determine the type of user being handled and then execute the appropriate section of the if/else logic branch. When you are finished, the method will appear as follows:

```
/**
 * @jpf:action
 * @jpf:forward name="existing" path="WizTwoExisting.jsp"
 * @jpf:forward name="new" path="WizTwoNew.jsp"
 */

public Forward WizOneAction(Controller.WizOneActionForm
    form)
{
    if (form.getCustomerType().equals("existing"))
    {
        custInfo = crmSystem.findAllCustomersArray();
        return new Forward("existing");
    }
    else
    {
        return new Forward( "new" );
    }
}
```

Automated Data-Binding

WizOneAction is now complete. You are almost ready to test the Web application, however there is one more important task remaining:

- Return to the Flow View of *Controller.jpf*, locate the *WizTwoNew.jsp* file, and double-click on it to open it.

You will now see the *WizTwoNew.jsp* loaded in WebLogic Workshop's JSP editor. The editor enables developers to create complex JSP forms using tags and properties, and supports a Design View that provides drag-and-drop, WYSIWYG functionality. As provided to you, the *WizTwoNew* page is nearly empty, so you need to make the necessary additions to the page so that it can collect information from a new customer; this data will then need to be stored in Avitek's databases. Fortunately, WebLogic Workshop makes this easy with drag-and-drop data-bound forms that can automatically connect user interface elements with back-end resources:

- Locate the *WizTwoAction* icon in the Data Palette and drag it on to the *WizTwoNew* page in the Design View.

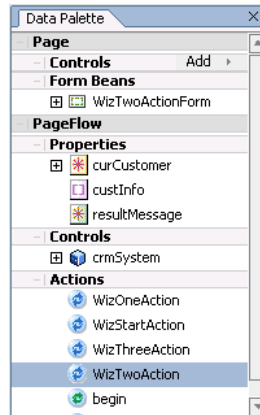
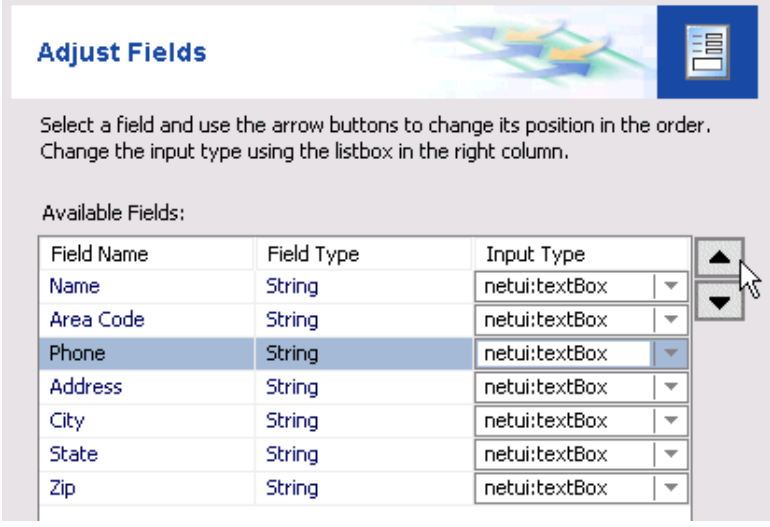


Figure 53. The Data Palette aggregates elements associated with the Java Page Flow

- This action brings up the Form Wizard.
 - In Step 1, leave the default Action as *WizTwoAction*.
 - In Step 3, you will see the data fields which were pre-defined in the Form Bean for *WizTwoAction*. Un-check the *custID* field to indicate that a new customer should not be able to enter in their own customer number – rather the system will generate this automatically.
 - Click **Next**.

Figure 54. Select the appropriate fields to include on the form using the Form Wizard

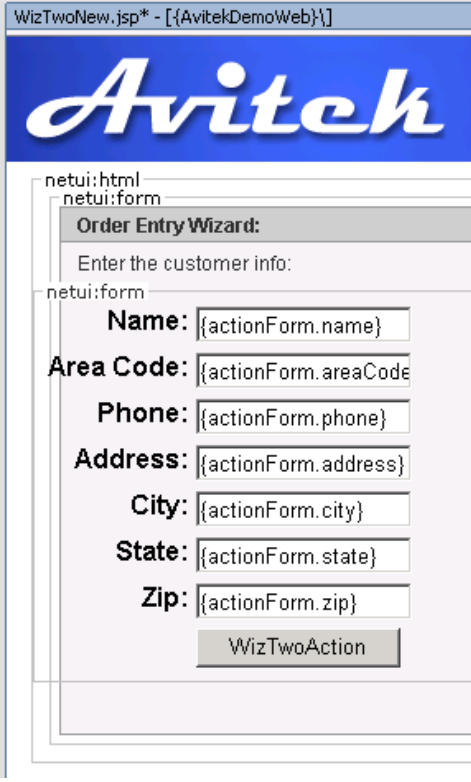
- The *Adjust Fields* dialog box allows you to modify the position in which the fields will appear, as well as the type of user interface element used to collect the data (i.e. textbox, checkbox, etc.). In this case, to make the order of fields more logical, simply use the arrow icons to move the *Area Code* and *Phone* fields above the *Address* field. Click **Create**.

The 'Adjust Fields' dialog box has a title bar with the text 'Adjust Fields' and a blue icon of a document with a list. Below the title bar is a text area with the instruction: 'Select a field and use the arrow buttons to change its position in the order. Change the input type using the listbox in the right column.' Below this is a section labeled 'Available Fields:' containing a table with three columns: 'Field Name', 'Field Type', and 'Input Type'. The table lists fields: Name, Area Code, Phone, Address, City, State, and Zip. The 'Phone' row is currently selected. To the right of the table are two arrow buttons (up and down) and a mouse cursor pointing at them.

Field Name	Field Type	Input Type
Name	String	netui:textBox
Area Code	String	netui:textBox
Phone	String	netui:textBox
Address	String	netui:textBox
City	String	netui:textBox
State	String	netui:textBox
Zip	String	netui:textBox

Figure 55. Adjust the fields to the desired position for the form

The JSP editor will update the *WizTwoNew.jsp* page to display the fields you just added, as shown in **Figure 56**.

The image shows a screenshot of a JSP editor window titled 'WizTwoNew.jsp* - [{AvitekDemoWeb}]'. The editor displays a WYSIWYG representation of a web form. At the top is the 'Avitek' logo. Below it is a section titled 'Order Entry Wizard:' with the instruction 'Enter the customer info:'. The form contains several text input fields labeled 'Name:', 'Area Code:', 'Phone:', 'Address:', 'City:', 'State:', and 'Zip:'. Each label is followed by a text box containing a placeholder value like '{actionForm.name}'. At the bottom of the form is a button labeled 'WizTwoAction'. The editor's background is light blue, and the form elements are rendered in a realistic style.

```
WizTwoNew.jsp* - [{AvitekDemoWeb}]  
  
Avitek  
  
Order Entry Wizard:  
Enter the customer info:  
  
Name: {actionForm.name}  
Area Code: {actionForm.areaCode}  
Phone: {actionForm.phone}  
Address: {actionForm.address}  
City: {actionForm.city}  
State: {actionForm.state}  
Zip: {actionForm.zip}  
  
WizTwoAction
```

Figure 56. WebLogic Workshop 8.1 provides a JSP editor with WYSIWYG functionality

WebLogic Workshop 8.1 automatically created the data-bound form that you specified in the Form Wizard, which demonstrates how easy it is to build Web applications in WebLogic Workshop incorporating data in databases, Web services, or any custom business logic. The data binding is enabled by an out-of-the-box tag library and even supports complex data like grids, lists, repeating elements, etc. Beyond the data management, WebLogic Workshop 8.1 will automatically manage the session, state management, and other complex details so the developer can maintain focus on the business problem.

Automated Deployment and Testing

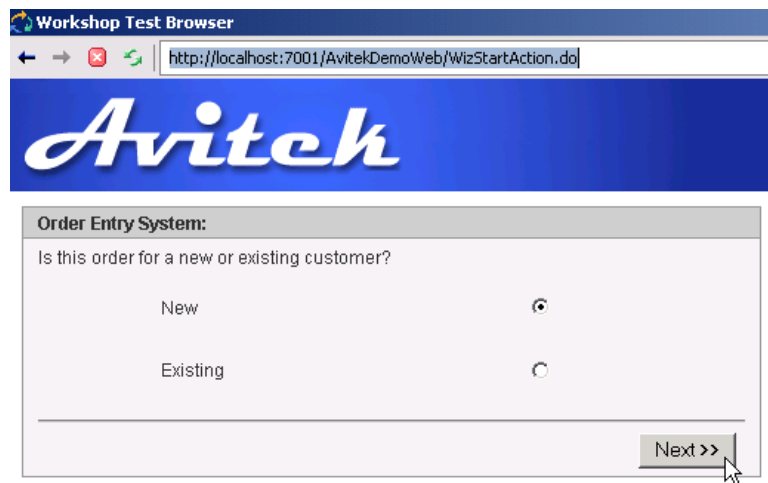
This completes the development required for the Avitek Order Entry System Web application so now you can test the application. Once again, you can take advantage of WebLogic Workshop's automated deployment and integrated test harness that makes iterative development a breeze:

- Save and close the *WizTwoNew.jsp* file and return to the *Controller.jspf* file. Click the **Start** icon (or **Debug** → **Start without Debugging**) to automatically deploy the Order Entry Web application and bring up the Workshop Test Browser. Note that, as with any Web application, loading the first view of a JavaServer Page (JSP) takes longer than all future views because the presentation code is being compiled on the first pass.
- From the starting page in the Workshop Test Browser, click **Start Entering An Order!** to begin the Avitek Order Entry System.



Figure 57. The entry-page for the Avitek Order Entry System

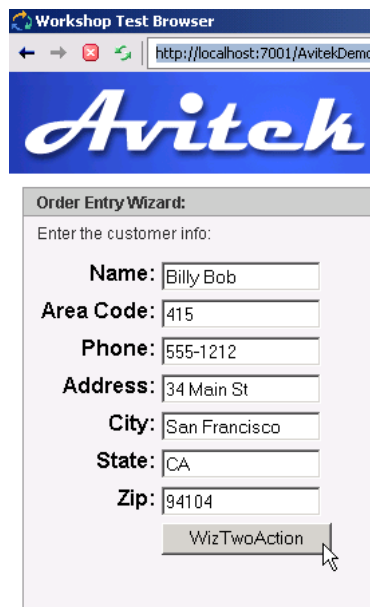
- On the first pass-through, identify yourself as a **New** customer using the radio buttons and click **Next**.



The screenshot shows a web browser window titled "Workshop Test Browser" with the address bar displaying "http://localhost:7001/AvitekDemoWeb/WizStartAction.do". The page features the "Avitek" logo in a blue banner. Below the banner is a form titled "Order Entry System:". The form contains the question "Is this order for a new or existing customer?". There are two radio button options: "New" (which is selected) and "Existing". At the bottom right of the form is a button labeled "Next >>". A mouse cursor is pointing at the "Next >>" button.

Figure 58. The application uses simple UI elements to collect information on the customer's status

- Thanks to the business logic you entered in the *WizOneAction* method, you will be properly navigated towards the "new customer" path and will have a chance to enter in your information. Here you will see the data-bound form that you created on the *WizTwoNew* page. Fill in the fields as indicated in **Figure 59** and click the **WizTwoAction** button.



The screenshot shows a web browser window titled "Workshop Test Browser" with the address bar displaying "http://localhost:7001/AvitekDemo". The page features the "Avitek" logo in a blue banner. Below the banner is a form titled "Order Entry Wizard:". The form contains the instruction "Enter the customer info:". There are several text input fields with labels: "Name:" (containing "Billy Bob"), "Area Code:" (containing "415"), "Phone:" (containing "555-1212"), "Address:" (containing "34 Main St"), "City:" (containing "San Francisco"), "State:" (containing "CA"), and "Zip:" (containing "94104"). At the bottom right of the form is a button labeled "WizTwoAction". A mouse cursor is pointing at the "WizTwoAction" button.

Figure 59. The form collects the relevant information for the new customer in this example

- Next you will have a chance to specify the quantity of products to order, hit **Next**, and you will receive a confirmation page.



Workshop Test Browser

http://localhost:7001/AvitekDemoWeb/WizTwoAction.do

Avitek

Order Entry Wizard: Customer Billy Bob

Enter Item Quantities:

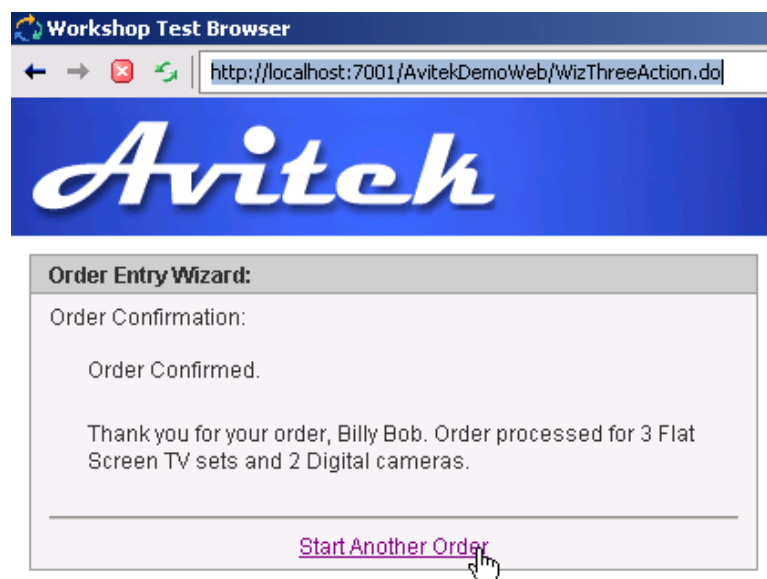
Digital Cameras	2
Flat Screen TVs	3

Buy

Figure 60. The application now collects the appropriate order information

It is that simple for any new customer to come to Avitek's web-site and start making orders for televisions and digital cameras! Just to make sure that the customer's information was handled properly, let's walk through the Order Entry System one more time:

- Click **Start Another Order** from the confirmation page.



Workshop Test Browser

http://localhost:7001/AvitekDemoWeb/WizThreeAction.do

Avitek

Order Entry Wizard:

Order Confirmation:

Order Confirmed.

Thank you for your order, Billy Bob. Order processed for 3 Flat Screen TV sets and 2 Digital cameras.

[Start Another Order](#)

Figure 61. The application confirms the initial order, and allows the user to start another order

- This takes you back to the starting page. Click on **Start Entering An Order!** as before.

- This time, identify yourself as an **Existing** customer, and click **Next**.

Workshop Test Browser

http://localhost:7001/AvitekDemoWeb/WizStartAction.do

Avitek

Order Entry System:

Is this order for a new or existing customer?

New ☐

Existing ☒

Next >>

Figure 62. In this pass, specify customer status as Existing

- You will now see a list of all the customers that are currently in the Avitek Customer database. You should see a line item for “Billy Bob” (or other name) you added in the previous steps. Also note that you will see entries for Bob Smith and John Doe, which correspond to the orders in *JavaOrder.xml* and *PurchaseOrderMessage.xml*, respectively, that were submitted in Exercise #3.

Workshop Test Browser

http://localhost:7001/AvitekDemoWeb/WizOneAction.do

Avitek

Order Entry Wizard:

Choose the customer from the table below:

Name	Address	Phone	
John Smith	100 Monroe st.	570-3353	<input type="radio"/>
Jane Doe	100 somewhere st.	555-1232	<input type="radio"/>
Bob Smith	123 Anywhere St.	123-4567	<input type="radio"/>
John Doe	111 First Avenue	555-1234	<input type="radio"/>
Billy Bob	34 Main St.	555-1212	<input checked="" type="radio"/>

Next >>

Figure 63. The application displays all the records in the Customer Database

Congratulations – the Order Entry System Web application is complete! BEA WebLogic Workshop 8.1 makes user-interface development easy because all you needed to do was specify logic, navigation, and data – the WebLogic Workshop 8.1 run-time framework handles all the low-level details and helps to boost your productivity.

In just these four simple exercises, you have successfully built standards-based, enterprise-class applications that will enable Avitek to be more agile and compete more effectively. Importantly, you were able to accomplish this impressive objective without needing any knowledge of J2EE or object-oriented programming disciplines, and still completed the job in one-tenth the time (or less!) than it would have required without the benefit of BEA WebLogic Workshop 8.1.

(Bonus) Exercise #5: Creating a Web Application From a Java Control

In the previous exercise, you completed the Order Entry System Web application by providing the missing business logic, flow control, and data-binding information. But what if you wanted to create a Web application from scratch? More specifically, suppose Avitek would like to create a Web application that a database administrator could use to access the Orders Database and perform basic database management tasks such as creating, retrieving, updating, or deleting records. BEA WebLogic Workshop 8.1 makes it incredibly easy to build a “starter” Web application with this “CRUD” functionality from any Java Control or table in only a few clicks.

- To start, you need to create a new Web application. After saving and closing all open files, right-click on the *AvitekDemoWeb* folder and select **New** → **Page Flow**.
- The Page Flow Wizard will appear. Provide *orderAdmin* as the Page Flow Name. The Controller File Name will be automatically adjusted. Click **Next**.

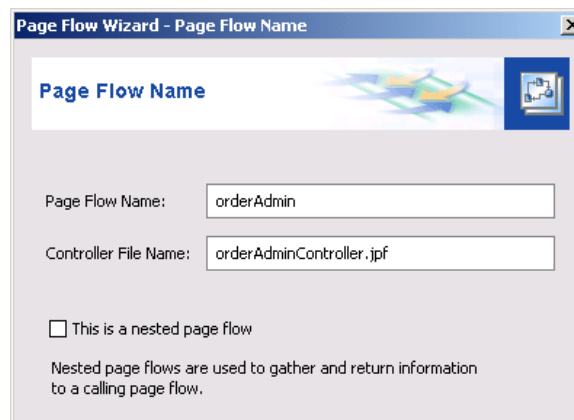


Figure 64. Create a new Page Flow named *orderAdmin*

- The following step allows you to specify if this new Page Flow is to be built from scratch or if it is to be based on a Java Control. You will want to use a Database Control (that still needs to be defined) so select the Database Control radio box and click the **New** button to launch the Database Control Wizard.
- Provide *OrdersAdminDB* as the Name and leave the default Data Source *cgSampleDataSource* to indicate that you will create a new table in the database that is embedded with WebLogic Platform. Click **Next**.

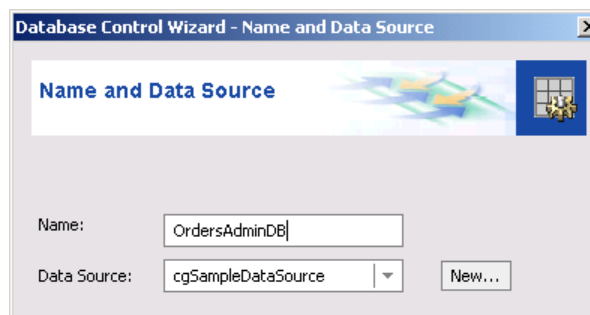


Figure 65. Use the Database Control Wizard to create and configure the new control

- You want to be able to query and update a data table, so leave the default setting in the top radio group. And below, select **CAJUN** as the Schema type, and select **ORDERS** from the Table drop-down box.

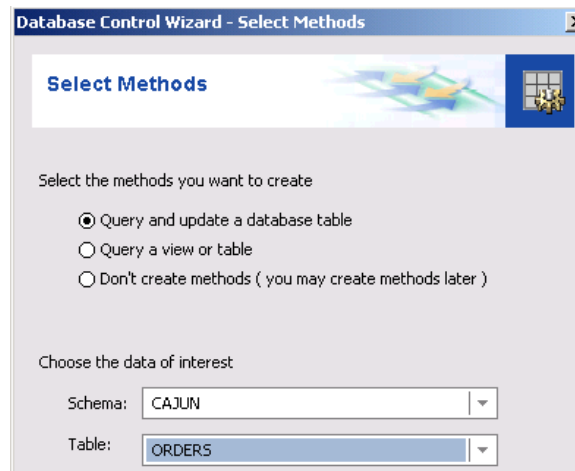


Figure 66. Specify the CAJUN schema and the ORDERS table as the data of interest

- The next step in the Database Control Wizard lets you see the available columns in the Orders table. These default settings are correct so just select **Next**.
- You would like the new primary keys to be automatically generated by the database (the default selection) so just click **Create** to complete the Database Control Wizard.
- This returns you to the original Page Flow Wizard dialog box. But now that you have successfully created the Database Control upon which the new Web application will be based, you can specify the *OrdersAdminDB* as the Database Control and click **Next**.

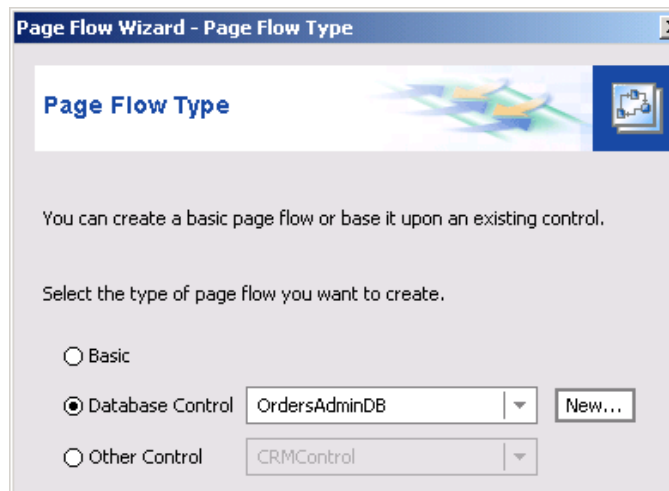


Figure 67. Return to the Page Flow Wizard, and select OrdersAdminDB as the basis for the new Page Flow

- Now you have a chance to select the actions that will be permitted in the new *OrdersAdmin* application. To reflect the fact that a new order entry would only be generated via the Order Entry System interfaces you built earlier, the database administrator in this case shouldn't have this particular ability so you should disable the *Insert New Rows* checkbox as shown in **Figure 68**. Finally, click **Create**.

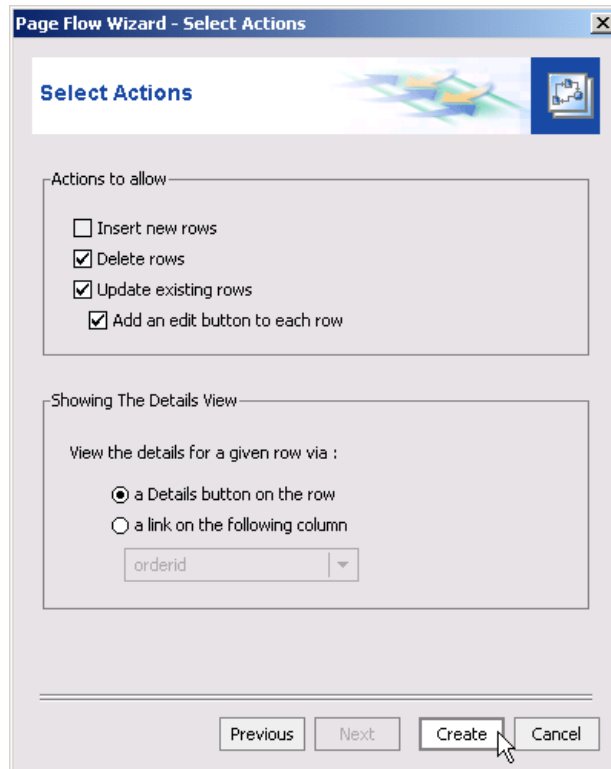


Figure 68. Customize the actions permitted in the new Page Flow application

You will now see the new *orderAdminController* Page Flow appear in the Flow View:

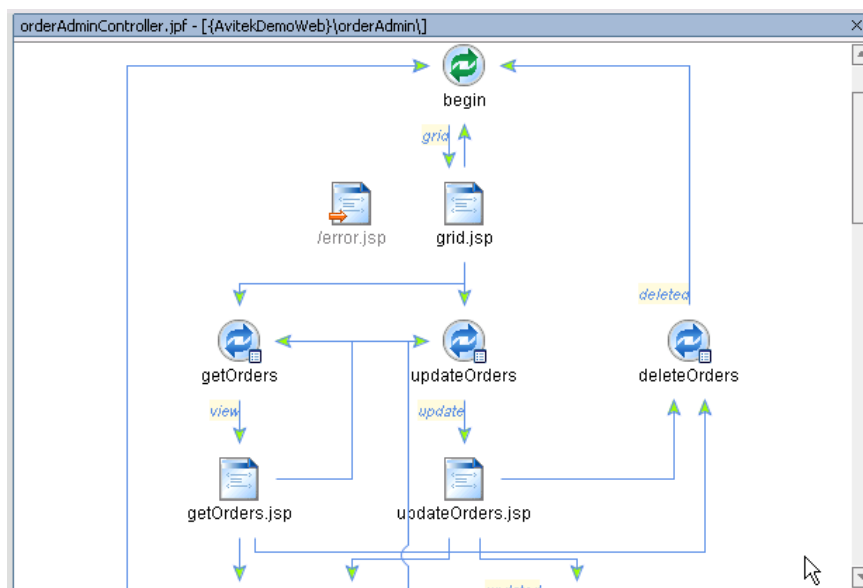


Figure 69. WebLogic Workshop 8.1 automatically defines the appropriate pages, flows, and actions required

As you can see in **Figure 69**, BEA WebLogic Workshop automatically created the necessary JSP pages, navigation instructions, and associated actions to provide the Orders Database administrative functions you specified. Of course, a developer could easily extend this application as appropriate, but the product of the Page Flows Wizard you just completed is a fully-functioning Web application that any authorized Avitek user around the world could use to perform database administration activities on the Orders database.

Not only that, you have created a truly enterprise-class application that natively supports MVC (model-view-controller) architecture and leverages Struts, but Workshop made it easy enough for any developer – you don't have to be an expert in any of these technologies.

- To see the new Web application in action, click the **Start** icon to deploy the application and invoke the integrated test harness. The first screen to appear will display all the records currently in the Orders database as shown in **Figure 70**.

OrdersAdminDB - Grid View:

Page 1 of 1

Details	Edit	Orderid ▾	Custid ▾	Cameraqty ▾	Flatscreenqty ▾
Details	Edit	1	1	5	15
Details	Edit	2	1	1	0
Details	Edit	3	2	2	0
Details	Edit	26633	44947	15	8
Details	Edit	32591	5983	2	2
Details	Edit	28025	82491	2	3

Page 1 of 1

Figure 70. The Grid View displays all the order records in the Avitek Orders database

- If you select the **Edit** link on a particular record, you are taken to the Edit View and have the ability to modify or delete a record.

OrdersAdminDB - Edit View:

Orderid	26633
Custid	<input type="text" value="44947"/>
Cameraqty	<input type="text" value="15"/>
Flatscreenqty	<input type="text" value="8"/>

Figure 71. The application provides administrators the ability to easily edit or delete existing order records

Congratulations! You have built a full-fledged Web application with core functionality from an underlying, standard database table in only a few minutes!

(Bonus) Exercise #6: Using XMLBeans For Handling XML in Java

While building the Order Entry Web service in Exercise #3, you leveraged Workshop's XQuery Mapping tools to handle the transformation between XML messages and the Java data objects that are used to process the information. The XQuery mapping approach empowers genuine loose-coupling – separation of the service's public contract (i.e. schema definition) from the internal Java implementation. XQuery mapping is complemented by another important BEA feature for loose-coupling named XMLBeans.

XMLBeans is an innovative technology that makes it very easy for developers to access and manipulate XML data and documents in Java by offering convenient Java object-based view of XML data without losing access to the richness of the original, native XML structure. Access to the Java class “views” are automatically generated just by submitting the relevant schema definition, as you already did in a prior exercise. To see XMLBeans in action:

- Clear the workspace and then re-open the *orderEntryService.jws* file in the Design View.
- Since you will be replacing the previously implemented XQuery mapping, you need to delete the previous transformation. The easiest way is to make *CreateOrderAsynch* the active method, and click on the *XQuery* property under the *Parameter-XML* in the Property Editor to launch the *XQuery Editor*.



Figure 72. Use the Property Editor to re-launch the XQuery Mapping Editor

- Next, click **Delete** to remove the existing transformation you created in Exercise #3. Click OK to return to the Design View.
- To incorporate XMLBeans in the *createOrderAsynch* method, just click on its hyperlinked name to switch to the Source View.

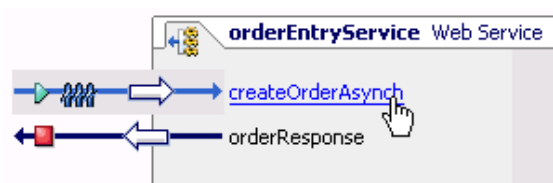


Figure 73. Click on the hyperlinked method name to switch to Source View

- Modify the *createOrderAsynch* method declaration so that instead of taking a *Customer* and *Order* parameter, it takes only a *PurchaseOrderDocument* parameter. The revised method definition line should appear as follows:

```
public void createOrderAsynch(PurchaseOrderDocument podoc)
```

Notice that when you are typing in this new variable type, you should see a pop-up box indicating that support for this variable requires importing a new class:

```
/**
 * @common:operation
 * @common:message-buffer enable="true"
 * @jws:conversation phase="start"
 */
public void createOrderAsynch(PurchaseOrderDocument podoc)
{
    cRMControl.createOrderAsynch(c,o);
}
```

org.openuri.easypo.PurchaseOrderDocument? Alt+Enter

Figure 74. WebLogic Workshop 8.1 includes a convenient Auto-Import feature in the Source View

This is due to Workshop's auto-import feature to enhance productivity in building modular applications. Press **Alt+Enter** to automatically import the appropriate *org.openuri.easypo* package (this name was defined in the schema file you imported in Exercise #3). The *PurchaseOrderDocument* is a Java type automatically created by Workshop when the *Purchase Order* schema is imported and concurrently processed by the embedded XMLBeans compiler. This Java class is typical in that it includes a number of variables and methods, that together provide full access to the underlying XML-based purchase order document.

At this point you have two options. One approach would be to create a new control that takes a *PurchaseOrderDocument* as its parameter for processing. Or alternatively, you could simply create and populate one *Customer* and one *Order* data structure from the information in the *PurchaseOrderDocument*, such that you could re-use the existing *createOrder* method with the existing *CRMControl*. Take this latter approach here for the sake of simplicity.

- Insert the following lines of code to use XMLBeans to extract the necessary pieces of information from the underlying XML to fill-in the important fields of the *Customer* and *Order* data structures. When finished, the method should appear as follows:

```
/**
 * @common:operation
 * @common:message-buffer enable="true"
 * @jws:conversation phase="start"
 */
public void createOrderAsynch(PurchaseOrderDocument podoc)
{
    Customer c = new Customer();
    Order o = new Order();
    PurchaseOrder po = podoc.getPurchaseOrder();
    c.name = po.getCustomer().getFirstname() + " " +
        po.getCustomer().getLastname();
    c.custId =
        po.getCustomer().getCustomerNumber().intValue();
}
```

```

o.custID = c.custId;
o.orderID =
    po.getOrderData().getOrderNumber().intValue();
o.cameraQty =
    po.getOrderData().getCameraOrder().getQuantity();
o.flatScreenQty =
    po.getOrderData().getTvOrder().getQuantity();
cRMControl.createOrderAsynch(c,o);
}

```

While typing the lines above, you will likely get Auto-Import messages and you may need to specify for Workshop to import *CRMSystem.Customer* and *CRMSystem.Order*.

- To test this revised Web service, click the **Start** icon.
- Click the **Test XML** tab in the Workshop Test Browser.
- Get the text from the *PurchaseOrderXBean.xml* file in the Application Window and copy it into the SOAP body text-box in the test browser. Click the **createOrderAsynch** button to submit the order message.
- As before, you should click **Refresh** to capture the rest of the messages involved in the asynchronous communication. When the *callback.orderResponse* item appears in the Message Log you can see the order confirmation message in the Client Callback.

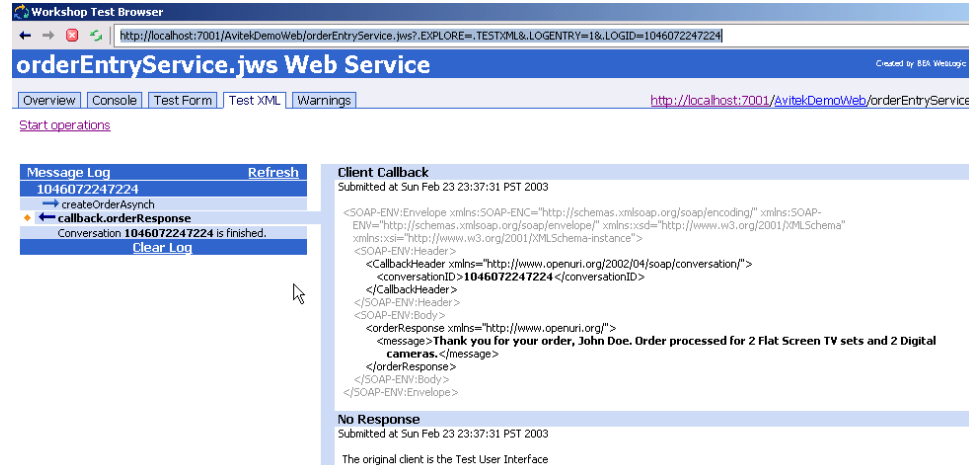


Figure 75. The test client tracks the asynchronous conversation and shows the order acknowledgement

As you can see, WebLogic Workshop 8.1 makes it easy to use XMLBeans to manipulate the XML. While this scenario only demonstrated extracting data from the underlying XML document using the Java interfaces, you could just as easily have traversed the raw XML using a Cursor API, or have used the suite of “setter” functions to extend or modify the XML message. Importantly, Workshop automatically ensures that any changes to the XML document is in compliance with the submitted schema that defines the expected structure. This preservation of XML fidelity combined with its ease-of-use is the key benefit of using XMLBeans to create loosely-coupled applications.

To help determine whether XQuery mapping or the XMLBeans approach is correct for your application, an important consideration is the nature of processing: end-point or way-point. In

end-point processing, the program is an end point for the XML. Its job is either to extract from the XML the information it needs to do its job or else to create an XML document from scratch. The application can do complex procedural things with the document, and it is acceptable for the application to discard information in the incoming XML document that it does not need. On the other hand, in way-point processing, the program acts on an XML document as that document flows through a process. In this case the program shouldn't be "lossy," since it doesn't know what functions precede or follow it in the business process. In these cases, preservation of the full XML fidelity is critical. As you might guess, it is these way-point applications that are particularly well-suited to XMLBeans whereas end-point applications may only require XQuery mapping. Often both are acceptable alternatives and the issue becomes one of developer choice.

Congratulations! You have now completed all the exercises in the JumpStart Guide.

Behind the Scenes

The ease of creating enterprise-class applications in BEA WebLogic Workshop 8.1™ is only the beginning. Although you haven't written a single line of J2EE code, the applications you have written are running on top of the industry-leading BEA WebLogic Server and taking advantage of sophisticated J2EE features like entity beans, stateless session beans, JDBC connections, and JMS queues.

In this product walkthrough you've hopefully seen how easy it is to create reasonably sophisticated applications in BEA WebLogic Workshop 8.1, including custom Java Controls, Web services, and Web applications. But there is additional power is what you haven't seen – all the machinery and J2EE platform features involved in making these applications run. The applications each takes advantage of EJBs, JMS Queues and other advanced features without exposing them to the developer at any time. If you need to, you can see many of these underlying components through the WebLogic Server Administrator's Console.

After completing these six exercises, it should be apparent that BEA WebLogic Workshop 8.1 makes it easy for all developers to build powerful, standards-based applications, while at the same time, taking advantage of leading application server features like clustering, caching, pooling, and message queuing that are otherwise accessible only to a relatively few, highly trained J2EE developers. With BEA WebLogic Workshop 8.1, you only needed to write the applications as if everything was a simple Java class.

To do what you just accomplished in this product demonstration without the help of BEA WebLogic Workshop's programming model would require years worth of knowledge of APIs like JNDI, JMS, and JDBC. It would require detailed knowledge of XML standards such as SOAP and WSDL, the Struts framework, MVC paradigms, state management, message correlation, data persistence, XML-to-Java mapping, and much more. Fortunately BEA WebLogic Workshop abstracts away this infrastructure complexity to let developers solve pressing business problems and achieve a faster time-to-value.

Conclusion

BEA WebLogic Workshop 8.1™ is an exciting evolution in the J2EE development arena. By combining the ability to create powerful, standards-based applications with a simplified programming model, WebLogic Workshop 8.1 delivers unprecedented productivity to IT development organizations and accelerates their time-to-value.

In the course of exploring the product demonstration outlined in this JumpStart Guide, you will have experienced a number of BEA WebLogic Workshop's important benefits:

Making J2EE accessible and productive: The WebLogic Workshop visual development environment and run-time framework insulate developers from the inherent complexity of J2EE. Instead of writing to low-level J2EE APIs, which require years of domain expertise, the WebLogic Workshop provides simplifying concepts such as controls, properties, and events that let developers focus on the business logic, not the infrastructure details. This means the enterprise-proven strength of J2EE is finally available to the majority of developers with skills in visual or procedural languages, at ten times the productivity of working directly with the low-level J2EE APIs.

Reducing IT complexity: BEA WebLogic Workshop 8.1 provides developers with a single tool and a single, simplified programming model for building and integrating any type of application on BEA WebLogic Enterprise Platform™ 8.1— including Web applications, Web services, Portals, and Integration applications. This approach streamlines the development organization and lets developers work on any platform project using the skills they already know. WebLogic Workshop 8.1 also provides an extensible Java Controls methodology to make it easy to leverage existing IT infrastructure such as databases, legacy applications, and other back-end resources — empowering developers to accomplish more in less time.

Enhancing project success with enterprise-class applications: BEA WebLogic Workshop 8.1 reduces development risk by implementing applications on scalable, reliable and secure enterprise-class architecture. The Workshop 8.1 runtime framework automatically creates applications using standard J2EE components to protect your technology investments and maintain maximum flexibility.

BEA WebLogic Workshop 8.1 is a unified, simplified, and extensible development environment that empowers all developers, not just J2EE experts, to build, test, and deploy enterprise-class applications on the BEA WebLogic Enterprise Platform.

Appendix: BEA WebLogic Workshop 8.1™ Specifications

Supported Platforms:

Microsoft Windows XP

Microsoft Windows 2000

Linux – Red Hat Advanced Server 2.1

Sun Microsystems 8 and 9 (run-time only)

HP-UX 11.0 and 11i (run-time only)

Application Server:

BEA WebLogic Workshop 8.1 *Application Developer Edition* requires BEA WebLogic Server 8.1

BEA WebLogic Workshop 8.1 *Platform Edition* also requires BEA WebLogic Portal 8.1 and/or BEA WebLogic Integration 8.1

Next Steps: Product Information and Developer Resources

For a full range of developer resources, including product subscriptions, up-to-the minute technical articles, demos and code samples, and more, please visit BEA's dev2dev developer portal at <http://dev2dev.bea.com>

BEA dev2dev is a complete developer program designed to ensure developer success and complement the productivity and rich development features found in BEA WebLogic Workshop 8.1™.

BEA dev2dev delivers a wealth of expert technical information, tools, training and support for developers of all levels of technical knowledge and experience. BEA dev2dev Online provides a single entry point for developers to download software plus a wide range of technical articles and whitepapers on BEA WebLogic Workshop™ and other BEA technologies, forums and peer networking, training events and seminars, user group connections, code samples, publications and more.

Other important developer resources:

- BEA Development Support; <http://www.bea.com/support/>
- BEA Education Services; <http://www.bea.com/education/index.shtml>
- BEA Developer Newsgroups; <http://www.bea.com/support/newsgroup.shtml>
- BEA Certified Developer Program;
<http://education.bea.com/training/CertificationProgram.jsp>

About BEA

BEA Systems is the world's leading application infrastructure software company, with more than 13,000 customers around the world including the majority of the Fortune Global 500. The BEA WebLogic® Enterprise Platform provides the application infrastructure foundation that simplifies the flow of information, decreases the costs of managing applications, and makes an enterprise more agile, productive, and connected. BEA's platform is also the de facto standard for more than 2,100 systems integrators, independent software vendors and application service providers who partner with BEA to ensure the successful deployment of customer solutions. Headquartered in San Jose, Calif., BEA has 96 offices in 34 countries and is on the Web at www.bea.com.

Copyright © 2003 BEA Systems, Inc.

BEA and WebLogic are registered trademarks, and BEA WebLogic E-Business Platform, BEA WebLogic Integration, BEA WebLogic Personalization Server, BEA WebLogic Portal, and BEA WebLogic Server are trademarks of BEA Systems, Inc. All other trademarks are the property of their respective owners.



BEA Systems, Inc.

2315 North First Street

San Jose, CA 95131 U.S.A.

Telephone: +1.408.570.8000

Facsimile: +1.408.570.8901

www.bea.com