Using BEA WebLogic Components

BEA WebLogic Components 1.7
Document Edition 1.0
January 2000

Copyright

Restricted Rights Legend

Trademarks or Service Marks

Using BEA WebLogic Components

Document Edition	Date	Software Version
2.0	January 2000	BEA WebLogic Components 1.7

Contents

1. Overview of WebLogic Components

eCommerce Component Packages 1-1
Customer and Session 1-2
ShoppingAdvisor and Items 1-2
Order Fulfillment 1-2
Features of eCommerce Componenets 1-3
Specifications for eCommerce Components 1-3

2. My BuyBeans.com Example

Beans & Co. Scenario 2-3
The Company 2-3
The Opportunity 2-3
The Project 2-3
The Solution 2-4
Get Started! 2-4
My BuyBeans.com Architecture 2-5
About the My BuyBeans.com Database 2-7
Introduction to the Hands-On Technical Tour 2-9

3. Developing Applications with WebLogic Components

Typical Development Approach 3-2
Component Examples 3-3
Foundation and Axiom 3-4
Package examples.axiom 3-4
Description 3-4
Belongings and EJBs 3-4
The Abstract Factory Pattern 3-5

Remote Iterators 3-5
Axiom Example 3-5
Workflow 3-6
Package examples.workflow Description 3-6
Workflow 3-6
Workflow Example 3-7
BusinessPolicy 3-8
Package examples.businesspolicy 3-8
Package examples.businesspolicy Description 3-8
ItemPriceCalculationPolicy and BusinessPolicy 3-9
BusinessPolicy Example 3-9
Extending 3-10
Package examples.extending 3-10
Package examples.extending Description 3-11
AlphaNumericSequencerExtension Component 3-11
PassByValue 3-13
Package examples.passbyvalue 3-13
Package examples.passbyvalue Description 3-13
Getting and Setting Attributes Using pass-by-value 3-13
Pass By Value Example 3-14
More Information 3-15

4. Deploying WebLogic Components Using Bean-Managed Persistence

Using Bean-Managed Persistence 4-1
The Oracle Reference Implementations 4-2
Additional Requirements 4-3
Deployment Sets Overview 4-3
Deploying on Windows NT 4-4
Deploying on Solaris 4-6

A. References

Sites A-1
EJB Documentation A-1
White Papers A-2

About This Document

This document introduces WebLogic Components from BEA. These Component are built entirely with Enterprise JavaBeans. Using them you will achieve simple and amazingly rapid development cycles.

This document also includes an eBusiness application built with BEA's WebLogic Components. This application was developed for Beans & Co., the Number One fictional bean vendor. First, you will run this application from the point of view of a customer, and then as an employee of the Beans & Co. company. Next you will find out how we built this e-commerce site by taking an inside look at the architecture!

This part of the kit has been designed to get you up and running as quickly and easily as possible, and familiar with the power and possibilities of BEA's WebLogic Components approach.This document explains what Java IDL is and describes how to use the BEA idltojava compiler for developing Java CORBA applications in the BEA WebLogic Components environment.

What You Need to Know

This document is intended mainly for application developers who are interested in using BEA WebLogic application components as software building blocks for eBusiness. It assumes a familiarity with Java programming, Enterprise Java Beans, and the BEA WebLogic Application Server, which serves as the platform for BEA WebLogic Application Components.

e-docs Web Site

BEA product documentation is available on the BEA corporate Web site. From the BEA Home page, click on Product Documentation or go directly to the "e-docs" Product Documentation page at http://e-docs.beasys.com.

How to Print the Document

You can print a copy of this document from a Web browser, one file at a time, by using the File—>Print option on your Web browser.

A PDF version of this document is available on the WebLogic Components documentation Home page on the e-docs Web site (and also on the documentation CD). You can open the PDF in Adobe Acrobat Reader and print the entire document (or a portion of it) in book format. To access the PDFs, open the WebLogic Components documentation Home page, click the PDF files button and select the document you want to print.

If you do not have the Adobe Acrobat Reader, you can get it for free from the Adobe Web site at http://www.adobe.com/.

Related Information

The following BEA WebLogic Enterprise documents contain information that is relevant to using the idltojava compiler and understanding how to implement Java CORBA applications in the WLE system.

For more information in general about Java IDL and Java CORBA applications, refer to the following sources.

- The OMG Web Site at http://www.omg.org/

- The Sun Microsystems, Inc. Java site at http://java.sun.com/

For more reference sites, please see Appendix A.

Contact Us!

Your feedback on the BEA WebLogic Enterprise documentation is important to us. Send us e-mail at **docsupport@beasys.com** if you have questions or comments. Your comments will be reviewed directly by the BEA professionals who create and update the WebLogic Components documentation.

In your e-mail message, please indicate that you are using the documentation for the BEA WebLogic Components 1.7 release.

If you have any questions about this version of BEA WebLogic Components, or if you have problems installing and running BEA WebLogic Components, contact BEA Customer Support through BEA WebSupport at **www.beasys.com**. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes
- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Examples*: LPT1 SIGNON OR

Convention	Item
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

`...`	Indicates one of the following in a command line:
	■ That an argument can be repeated several times in a command line
	■ That the statement omits additional optional arguments
	■ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

1 Overview of WebLogic Components

This topic includes the following sections:

- eCommerce Component Packages
- Features of eCommerce Components
- Specifications for eCommerce Components

eCommerce Component Packages

BEA's WebLogic Components are software building blocks for eBusiness. This family of Enterprise Java Beans helps you bring new business services to your customers quickly and easily, while allowing you to focus precious resources on your unique competitive requirements.

Using novel design patterns, we have modeled, designed, built, and tested our reusable server-side components to work seamlessly in conjunction with your commercial EJB application server.This overview describes these components and how they interact to create a Web presence. You can also refer to the complete component catalog (list only) or the complete API.

Customer and Session

The most basic entities for any business are customers and the products sold to them. The Customer is an extension of the Axiom.Person. It provides the ability to store contact, profile, and billing information for your customers. The Session components are used to manage the process of allowing customers to access the system as guests and then to register when they are ready to make a purchase. They also bind customers to the orders that they build.

ShoppingAdvisor and Items

The **Item** is the interface to the products that you are selling. It stores basic product identification and description, and provides a mechanism for pricing, including runtime pluggable pricing policies. The pricing mechanism is designed to allow you to take into account a specific customer's profile. This allows the application of special merchandising discounts and incentives. The **ShoppingAdvisor** is the means by which you organize your products and make them searchable. Its additional features include learning about customer preferences over time and recommending products based on the resulting profile.

Order Fulfillment

The **Order** acts both as a shopping cart and the basis for order fulfillment. It is the mechanism by which a customer keeps track of items that they want to purchase. The list is manipulated through business methods so that overloading can enforce the business rules associated with building an order. There is also an order cost calculation method that can be used to take into account discounts across multiple individual orders.

When an order is completed it is bound to a **PackingList** so that shipping cost can be calculated. The next step is the creation of an **Invoice**, so that the order can be billed. Finally, the **Inventory** is updated.

The **TroubleTicket** components provide customer service issue tracking. These components provide you the ability to accept and track issues submitted by your customers.

Features of eCommerce Components

- **Enterprise JavaBeans** built from the ground up
- **Plug-and-play components** that allow you either to use our out-of-the-box solution, or to integrate with your legacy applications
- Easy-to-use **component APIs** Fully **customizable and extensible** using technologies like pluggable methods and dynamic runtime configuration
- Implemented using **established design and analysis patterns** for ease of use and re-use
- Component**s work with other EJBs**, including third-party and custom-built components
- High-performance features such as **pass-by-value** (PBV)
- Works with leading EJB **Application Servers**
- Works with **leading databases** (Oracle, Sybase, DB2, Cloudscape, etc.)
- Integrate with **legacy systems** (CICS, IMS, legacy databases)
- **You don't need to be an EJB expert to use them!**

Specifications for eCommerce Components

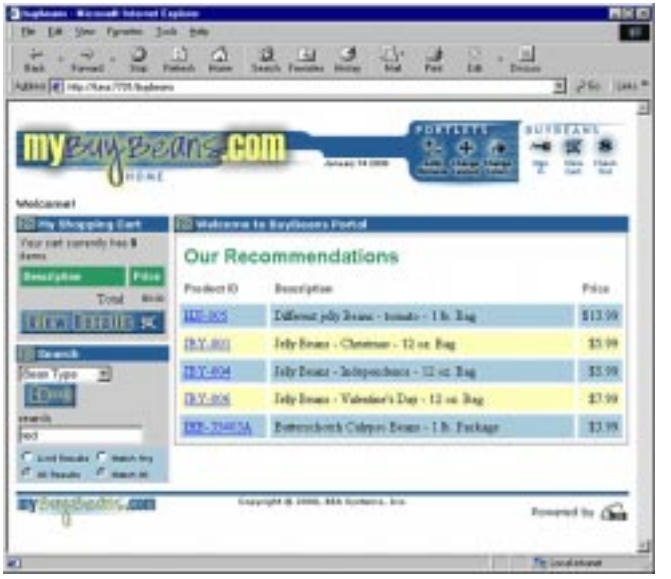
- **100% Pure Java**
- BSCs are Enterprise JavaBeans 1.0
- Support for Java 1.1 (Java 2 forthcoming)
- Support for the advanced Java 2 Collections API

- Support for Enterprise Java APIs including EJB, RMI, JNDI, JTS, and JDBC
- Support for modeling using UML, with roundtrip engineering from Rational Rose
- Component can be invoked from Java Clients, Java Servlets, Java Server Pages (JSP/JHTML), CORBA Clients and Servers, ActiveX/COM, and other clients (Visual Basic, PowerBuilder, etc.)
- Full support for BEA's WebLogic. Support for other leading application servers coming soon.

2 My BuyBeans.com Example

This Demo includes the following sections:

- Beans & Co. Scenario
- Get Started!
- My BuyBeans.com Architecture
- About the My BuyBeans.com Database
- Introduction to the Hands-On Technical Tour



Beans & Co. Scenario

The Company

Beans & Company is a regional powerhouse in beans. For more than a decade they have expanded their catalog business to include virtually every variety of bean. They have accomplished this through acquisition of other bean-related businesses. Aggressive marketing strategies have successfully positioned Beans & Co. to expand into the global marketplace!!!

The Opportunity

It is time for Beans & Company to live up to its sales potential and deliver value to its investors. To reach these goals they need to streamline their operations and quickly reach a worldwide audience. They have leading technology and a skilled technical staff. Now, they must leverage these capabilities across all of their separate business units, and deliver a unified brand image to the consumer.

The Project

Senior management has asked their technical staff to deliver a presence on the Internet. The new site must give customers access to the complete product line. To meet the expected demand, the system must be tightly integrated with the existing logistics and accounting capabilities. To minimize risk and meet the deadline, they need to use their own in-house technical and operations staff.

The Solution

Using enterprise integration, Beans & Co. is able to combine their disparate existing systems with the new technology they need. They are also able to deliver a cohesive presentation to consumers. The technology, Enterprise Java Beans, is being aggressively adopted across all the distinct platforms on which their systems run.

Beans & Co. is using BEA WebLogic Components. BEA is showing Beans & Co. the power of this new technology. New business and presentation logic is being developed using BEA components. Special implementations are being created to interface with the existing shipping, inventory, and billing systems. They are even able to leverage their extensive customer database and product catalogs.

Get Started!

Now that you have installed WebLogic Components, you have everything you need to get going.

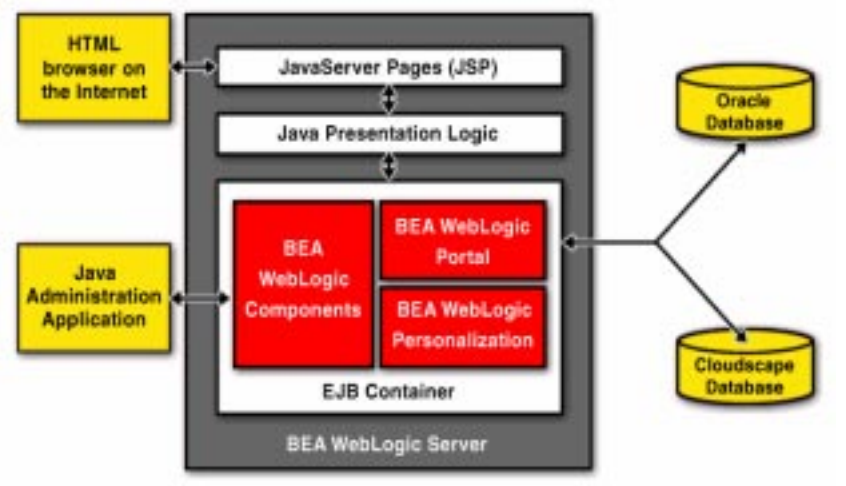
- First you will want to start the WebLogic application server.
- Next you will want to connect to the site. (You can login with username cool and password bean, or register a new user and shop!)
- You can run our Java based Admin Client to manage the site and get a behind the scenes look. **(Login and Password fields should be empty.)**

Now that you are really impressed, you'll want to find out how we did it. Of course this site was made possible by our WebLogic Components! The site also uses industry-leading technology like the WebLogic application server and Cloudscape database. We have installed evaluation copies of these products to support this demo and to show you how powerful they are.

We also include Deployment Sets for leading relational databases using BEA own Bean Managed Persistence. See Chapter 4, "Deploying WebLogic Components Using Bean-Managed Persistence," for information on how they work for Oracle and the Solaris platform.

To learn more about the technology and the tools, please see Appendix A, "References."

My BuyBeans.com Architecture

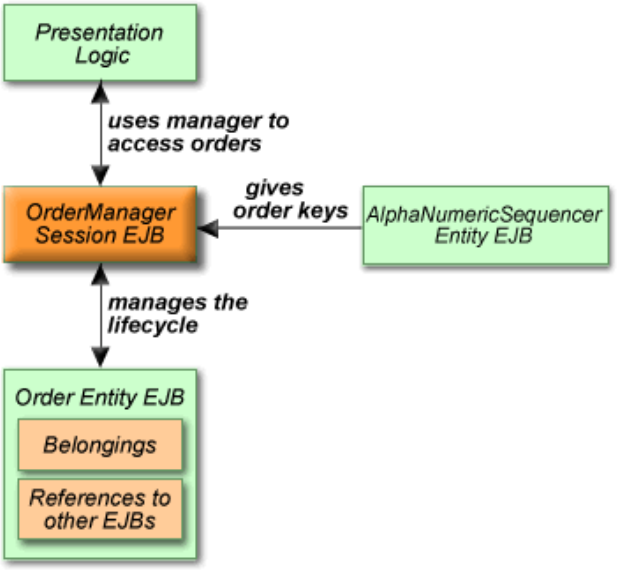


My BuyBeans.com is an n-tier distributed system entirely built using BEA WebLogic technology. The system consits of a client front-end, a middle tier of WebLogic Enterprise JavaBeans™ (EJB) components, and a back-end database. The middle tier also contains the BEA WebLogic Portal framework and Personalization engine. All components of the middle tier run within an instance of the BEA WebLogic Application Server.

Customers access My BuyBeans.com using a HTML Web browser. The Web browser uses HTTP to connect to a set of Java Server Pages (JSP) on the application server. A JSP consists of static HTML and embedded Java code that generates dynamic HTML contents. When accessed for the first time, each JSP is compiled by the WebLogic Application server into a Java Servlet. The output of the Java Servlet is the static HTML contained in the JSP along with the dynamic HTML that is generated by the execution of the embedded Java code.

The use of JSPs both simplifies and streamlines the process of dynamic HTML generation. It allows a Web designer to design and implement a Web page using the normal HTML design tools that he or she is familiar with, while at the same time, freeing an Application developer to embed the necessary Java code for the dynamic content of the page. The JSP source is provided so that you can get a feel for just how easy it is to use and modify.

Using the embeded Java code found in the JSP, the generated Servlets make calls to presentation logic written in Java. The My BuyBeans.com presentation logic is written as a set of Java classes. These classes encapsulate data access to the middle tier of WebLogic Components, thus making the JSPs simpler and easier to maintain. The presentation logic objects use the Java Naming and Directory Interface (JNDI) to locate the appropriate WebLogic Components. Once a reference is obtained to a WebLogic Component, it can be used as if it were a local resource.



Component Interaction Example

At the heart of the system are WebLogic Components, a set of 80 EJBs that provide most of the functionality of essential e-business. These beans run inside an "EJB Container" on the application server and expose the My BuyBeans back-end systems. Since the components can be found using the Java Naming and Directory Interface (JNDI), the can be accessed from anywhere on the network. To simplify delivery of

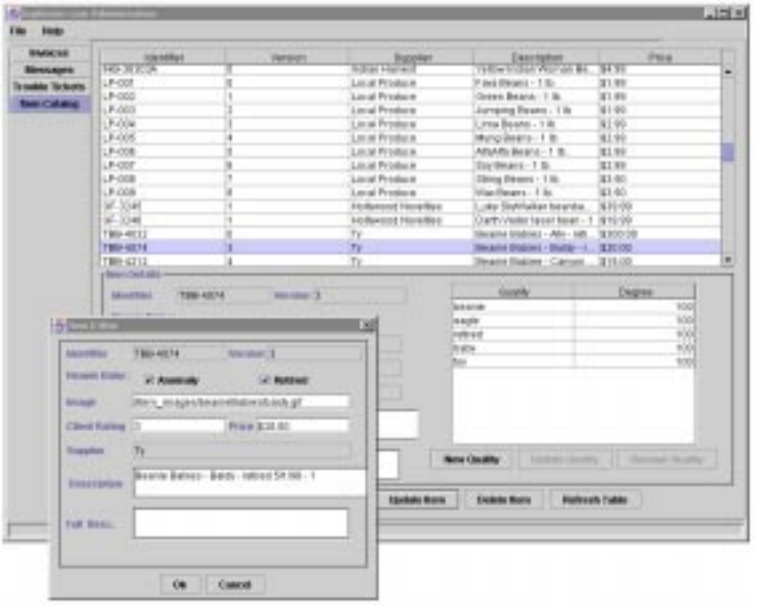
this demonstration, all of the components and servlets are running together in one instance of WebLogic Server. When deploying this application on an enterprise level, the various components will run on different machines and, potentially, on different architectures.

My BuyBeans.com uses both Session and Entity WebLogic Components. Component access to the My BuyBeans back-end is facilitated by the WebLogic Application Server which provides a JDBC connection pool to the My BuyBeans database. The Entity beans in this application use "container-managed" persistence. This means that WebLogic Server manages storing the contents of an Entity EJB to the database when the bean's persistable attributes change. The demo My BuyBeans.com application uses a Cloudscape database. The database is lightweight, and runs embedded within the WebLogic Server, so we don't need to start it separately. We have provided documentation on the database schema and how it is used by My BuyBeans.com.

My BuyBeans.com uses the concept of pluggable methods to set up pricing policies for its products and to calculate the prices based on these policies. Three ItemPriceCalculationPolicies are used, each holding business logic specific to the pricing of a different class of Items. We have included the UML diagram and documentation for these items and their pricing policies.

In addition to the WebLogic Components, My BuyBeans.com takes advantage of the features of the WebLogic Portal framework and WebLogic Personalization engine. The WebLogic Portal framework allows an Internet portal site to be quickly assembled from a collection of *portlet applicaitons*. Additionally, the Portal framework provides mechanisms for portal administration and user personalization. Portlet applications (also referred to as Portlets) are JSP and HTML pages that present dynamic content in a portal application. The content is laid out in the portal page by the Portal framework according to a user's personalized information as stored by the WebLogic Personalization engine. My BuyBeans.com is a portal application consisting of 15 portlets.

Just so you don't think that this is browser-only technology, we have also created a back-office administration application implemented as a standalone Java application. This application provides a unified interface to commonly performed tasks at My BuyBeans.com (such as managing the item catalog, or dealing with trouble tickets from customers). This application accesses remote objects directly to accomplish its goals.



About the My BuyBeans.com Database

The My BuyBeans.com application uses a Cloudscape database. The database is lightweight, and runs embedded in the application server, so you don't need to start it separately. As you run the application, it is important to keep in mind that because this evaluation version of Cloudscape supports table-level locks only, it handles only a single user. While a table is locked by one user, another user does not have access to the same table. **Concurrent access by multiple users may cause deadlocks and corrupt the Cloudscape database.**

The My BuyBeans.com database contains the following tables corresponding to the beans deployed for the My BuyBeans application:

TABLENAMES	
ALPHANUMERICSEQUENCER	ITEM
BASICBEAN	ITEMINVENTORY
BEANIEBABY	ITEMQUALITIES
BUSINESSSMARTWORKAREA	ITEMSBYQUALITY
COFFEEBEAN	JELLYBEAN
CUSTOMER	MAILBOX
CUSTOMERPROFILE	PACKINGLIST
EBUSINESSSESSION	SHIPPINGMETHOD
INVENTORYRECORD	TORDER
INVOICE	TROUBLETICKET

Run the `itemloader.sh` and `shippingmethodloader.sh` to load the data into database.

Note: Examples assume all directory strings are relative to the Weblogic Components install directory (`WEBLOGICAC_HOME`).

To run `itemloader.sh`, use:

```
bin/solaris2>sh itemloader.sh -url http://hostname:7501
-tabdelimitedfile
WEBLOGICAC_HOME/deploy/weblogic/oracle8x/misc/BuyBeans.txt
```

To run shippingmethodloader, use:

```
bin/solaris2>sh shippingmethodloader.sh -url http://hostname:7501
-commadelimitedfile
WEBLOGICAC_HOME/deploy/weblogic/oracle8x/misc/shippingmethods.txt
```

Note: If you make any structural changes to the My BuyBeans database, you will need to make the appropriate changes in the deployment descriptor of the particular beans. These files reside under the src directory for each bean in a deployment folder and are named `<BeanName>DD.txt`.

Note: The instructions on this page are for Solaris. For a Windows-based system, use the same commands from the command line (i.e., the same syntax):

instead of

`bin/solaris2> sh XXX.sh`,

run

`bin/win32> XXX.bat`.

Introduction to the Hands-On Technical Tour

The ultimate learning tool is a working application -- and we've provided a way for you to see how it all fits together. BEA WebLogic Components Technical Tour takes you through the entire development process of extending the My BuyBeans.com site with a custom item.

The tour contains detailed information on each of the tools that you need to create your own WebLogic Components extensions, including the SmartGenerator WebLogic Components generation tool. For more information about the SmartGenerator and UML modeling, you can also refer to Modeling with WebLogic Components.

To find out more about our packages and descriptions for each of the components, see the Bean Catalog or the complete API.

Any more questions? Try the Frequently Asked Questions (FAQ).

If you would like more information about Enterprise Java Beans and their inner workings, the best place to go is straight to the EJB Specifications on Sun's web site.

CHAPTER

3 Developing Applications with WebLogic Components

This topic includes the following sections:

- Typical Development Approach
- Component Examples, including:
  - Foundation and Axiom
  - Workflow
  - BusinessPolicy
  - Extending
  - PassByValue
- More Information

Typical Development Approach

To create a distributed client-server application in Java using BEA's WebLogic Components technology, you must perform the following programming steps:

1. Model the interfaces.

   Note: Refer to our white paper on Modeling with eBSC's.

2. Generate the classes that implement the interfaces.

3. Implement the interfaces by adding the business logic.

4. Compile the implementation files.

   Note: For aspects of steps 2, 3, and 4, refer to the hands-on technical tutorial.

5. Write the client-side software application (components).

6. Compile the client application (components).

7. Deploy the EJBs.

   Note: Refer to information about our BMP deployment sets.

8. Run the client.

Component Examples

To become better acquainted with the workings of an EJB-based application built using our WebLogic Components, follow these examples:

- Foundation and Axiom
- Workflow
- BusinessPolicy
- Extending
- PassByValue

Note: To build and run any of these examples, you must have the following in your CLASSPATH:

- `theory-smart-generator.jar, theory-axiom-foundation.jar, theory-ebusiness.jar, theory-examples.jar` Each of these jar files can be found in the lib directory under the WebLogic Components installation directory.
- Application Server classes (default classpath required by WebLogic Server)

The fastest way to run any of the examples is by using the scripts provided in `...\bin\win32\*.bat or ..\bin\solaris2\*.sh` (Found under the WebLogic Components installation directory)

Foundation and Axiom

This example demonstrates the core technology: *Belongings*, *Entity Beans*, *Collections* and *RemoteIterators*.

Package examples.axiom

The Axiom example shows the use of BEA Axiom package of WebLogic Components.

Table 3-1 Axiom Package Summary

Class	Description
AxiomExample	Shows how to use BEA's Axiom package of WebLogic Components.

Description

The Axiom example shows the use of BEA Axiom package of WebLogic Components.

This example demonstrates:

- The Abstract Factory Pattern
- How to use EJB WebLogic Components
- Usage of Belonging WebLogic Components
- Remote Iterators

Belongings and EJBs

The Axiom package contains light weight components known as belongings, as well as Entity and Session EJB components. Belongings can be aggregated to other components by value. EJBs are used alone or aggregated to other components by reference or value.

The Abstract Factory Pattern

All WebLogic Components use the abstract factory pattern. The principle is very simple: Don't use `new()` to create an object, instead, you use `Home.create()`. The Abstract factory pattern is implemented as the "Home" for EJBs and as a Java class with static methods for Belongings.

Remote Iterators

This example also shows the use of Remote Iterators, one of BEA's collection APIs. You can iterate through a collection both locally and remotely. Locally, the whole collection is sent to you from the server and you can traverse it. This is good for small collections where you need to iterate a lot. For large collections, you can use one or more remote iterators. Remote iterators traverse the collection in the server. That way, you only * bring over the net the values that you need saving bandwidth. They also serve as "bookmarks" on a collection, since they "remember" in what position you leave them.

Axiom Example

The example application performs the following steps:

1. Find or create or a Customer component
2. Create belongings
3. Add belongings to the Customer
4. Use a Remote Iterator to iterate through the belongings
5. Remove the Belongings

To get the most out of this example, first read through `AxiomExample.java on our web site`. Then you can build it and run it.

See above note:"To build and run any of these examples, you must have the following in your CLASSPATH:" on page 3-3

Workflow

Workfow, *eBusinessSession*, and *eBusinessSessionManager* components. The *Workflow* maintains state for the session and guides the user through the process.

Package examples.workflow

Shows the use of BEA WebLogic Workflow Components.

This example demonstrates:

- How to use WebLogic Components.
- Usage of the Workflow component.
- Usage of `eBusinessSession` and `eBusinessSessionManager`.

Table 3-2 Workflow Package Summary

Class	Description
WorkflowExample	Workflow example.

Package examples.workflow Description

Shows the use of BEA WebLogic Workflow Components.

This example demonstrates:

- How to use WebLogic Components.
- Usage of the Workflow component.
- Usage of `eBusinessSession` and `eBusinessSessionManager`.

Workflow

This example shows the use of a WebLogic Components that has a workflow associated to it The workflow states and transitions are modeled with Rational Rose. For this examples, we'll use the `EBusinessSession` Component. This component has a workflow that guides it through the different stages of an online e-business session. If you look at the Rose model file for the ebusiness.session package, you will find that

`EBusinessSessionWorkflow` has a state diagram associated to it. The workflow logic can be implemented in any way you want; however, BEA provides a reference implementation. For the reference implementation, for each component with the BSC.Workflow stereotype, all the states and transitions in the Rose model are generated into a complete state machine by the SmartGenerator, so you can use it immediately, without any hand-coding of the workflow states or transitions. Please note that the SmartGenerator is NOT included in the WebLogic Components Kit. In this example, we also use the EBusinessSessionManager and show how a "manager" session bean can simplify the usage of an entity bean

Workflow Example

The workflow example application performs the following steps:

1. Create a Guest Session component using the `EBusinessSessionManager`.
2. Try options such as `enroll`, `cancellEnrollment`, `becomeGuest`, and `disableAuthentication`. (You can find these transitions in the `EBusinessSessionWorkflow` state diagram)
3. Register as a new or Login as an existing Customer
4. Perform more options (`authenticate`, and `disableAuthentication`)

To get the most out of this example, first read through `WorkflowExample.java on` our web site then you can build it and run it.

See above note:“To build and run any of these examples, you must have the following in your CLASSPATH:” on page 3-3

BusinessPolicy

Pluggable Methods, Strategy Pattern, or *Individual Instance Method.* No matter what you call it, it is a powerful design tool. This example demonstrates how *ConfigurableEntity* beans and **BusinessPolicy** work together to create very flexible solutions.

Package examples.businesspolicy

BusinessPolicy Example shows the use of BEA WebLogic BusinessPolicy Components.

Table 3-3 BusinessPolicy Package Summary

Class	Description
AprilFoolsDiscountPolicy	This class is a custom item pricing calculation policy.
BusinessPolicyExample	This example demonstrates the concept of "Pluggable Methods", better known as policies.
SeniorCitizenDiscountPolicy	This class is a custom item pricing calculation policy.

Package examples.businesspolicy Description

BusinessPolicy Example shows the use of BEA BusinessPolicy WebLogic Components.

This example demonstrates:

- How to use WebLogic Components
- How to add a default Policy to an Item using WebLogic Components BusinessPolic
- How to use a non-default policy to change the price of an item

ItemPriceCalculationPolicy and BusinessPolicy

This example shows the use of `theory.smart.ebusiness.item.ItemPriceCalculationPolicy` which is an extension to `theory.smart.foundation.BusinessPolicy`. A `BusinessPolicy` consists of rules and regulations, specific to your business. These rules can be encapsulated into a component and then added to a Component such as an Item.

This example demonstrates the concept of "Pluggable Methods", better known as policies. When you create your components, you will realize that many times you want to alter the component behaviour based on external conditions that you can not evaluate at development time. Reusability, extensibility and rapid development and enhancement are typical problems that can be solved using policies. `BusinessPolicy` is BEA's implementation of the Policy and Strategy design patterns. Using this concepts allows you to replace the default policy at runtime. The policy is stored as a property for the item. In this example we will use an item component. The item component has a pricing policy. The item's price is calculated based on a given quantity and the pricing policy. You can replace the pricing policy to alter the way the price is calculated for the item. This means that you can modify the behaviour of the item by plugging in a method that calculates the price the way you want If you do not provide a pricing policy, a default policy will be used. The example creates an item. It Then, sets the `SeniorCitizenDiscountPolicy` as the default pricing policy for the item. Then, the item's price is calculated using the default policy. Finally, it modifies the item's quantity and once again, calculates the price; this time using the `AprilFoolsDiscountPolicy` policy. To better understand this example it would be great if you go through the Axiom example first.

The concept is also used in our BuyBeans.com online store where different pricing policies of BuyBeans are used for calculating the prices of `examples.buybeans.item.BeanieBaby`, `examples.buybeans.item.CoffeeBean`, and `examples.buybeans.item.JellyBean` components. They use `BeanieBabyPricePolicy`, `CoffeeBeanPricePolicy`, and `JellyBeanPricePolicy` respectively.

BusinessPolicy Example

The BusinessPolicy example application performs the following steps:

1. Find or create or an Item component
2. Set the Item's `Quantity`.

3. Add the `SeniorCitizenDiscountPolicy` to the Item as the default pricing policy and change the Item's price.
4. Change the Item's `Quantity`.
5. Change the item's price using the `AprilFoolsDiscountPolicy`.

To get the most out of this example, first read through `BusinessPolicyExample.java` on our web site. Then you can build it and run it.

See above note:"To build and run any of these examples, you must have the following in your CLASSPATH:" on page 3-3

Extending

Part of the power of our component suite is that all components are extensible. Through inheritance you can add specialized methods and attributes to WebLogic Components without having to 'reinvent the wheel.'

Package examples.extending

The Extending Example shows you the use of BEA WebLogic AlphaNumericSequencer Components and how to extend WebLogic Components. This example demonstrates:

- How to use WebLogic Components
- How to use a sequencer
- How to extend a component.

Table 3-4 Extending Interface Summary

Interface	Description
AlphaNumericSequencerExtension	Primary Key = `examples.extending.AlphaNumericSequencerExtensionPk`

Table 3-4 Extending Interface Summary

Interface	Description
AlphaNumericSequencerExtensionHome	The home interface for the `AlphaNumericSequencerExtension` entity bean.

Table 3-5 Extending Class Summary

Class	Description
AlphaNumericSequencerExtensionImpl	Primary Key = `examples.extending.AlphaNumericSequencerExtensionPk`
AlphaNumericSequencerExtensionPk	This is a PrimaryKey for managing the life cycle of a BSC Configurable/Entity bean.
AlphaNumericSequencerExtensionValue	
ExtendingExample	This example demonstrates extending a WebLogic Components.

Package examples.extending Description

The Extending Example shows you the use of BEA AlphaNumericSequencer Component and how to extend an Component.

This example demonstrates:

- How to use WebLogic Components
- Usage of a sequencer
- How to extend a component.

AlphaNumericSequencerExtension Component

The `AlphaNumericSequencerExtension` extends `AlphaNumericSequencer`. The `AlphaNumericSequencer` generates sequential identifiers in a user prescribed format. The user of the class can configure the prefix, suffix, step, and width of these

components. The current sequence number is persisted and access to the counter is controlled so that uniqueness can be guaranteed across all users of a given sequencer. It is often used to generate unique keys for entities such as accounts, users, and sessions. The `AlphanumericSequencerExtension` has a `getValue()` method that converts returns the counter value in hexadecimal notation

The extending example application performs the following steps:

1. Find or create or an `AlphaNumericSequencerExtension` component

2. Iterate through 10 sequencer strings.

To get the most out of this example, first read through `ExtendingExample.java` on our web site. Then you can build it and run it. Other files in this example are:
`AlphaNumericSequencerExtension.java,`
`AlphaNumericSequencerExtensionHome.java,`
`AlphaNumericSequencerExtensionImpl.java,`
`AlphaNumericSequencerExtensionPk.java`

See above note:"To build and run any of these examples, you must have the following in your CLASSPATH:" on page 3-3

PassByValue

Sometimes it is useful to get or set all of the attributes of an object with a single method call. When dealing with remote objects that are persisted in the database, this results in a tremendous performance gain. BEA's WebLogic Components give you that flexibility.

Package examples.passbyvalue

Shows the use of BEA WebLogic Components' *pass-by-value* feature. This example demonstrates:

- How to use WebLogic Components.

Table 3-6 PassByValue Class Summary

Class	Description
Pass-by-value example.	Pass-by-value example.

Package examples.passbyvalue Description

Shows the use of BEA WebLogic Components' pass-by-value feature.This example demonstrates:

- How to use WebLogic Components.
- How to get/set values for a WebLogic Components using pass-by-value.

Getting and Setting Attributes Using pass-by-value

This example shows how you can get and set the attributes of an Entity Component by value. What this means is that instead of getting/setting one attribute at a time, you can request that a local copy of all attributes be sent to you directly, in one remote call. You can then read and modify this "Value object" or local copy, and send it back in one remote call. This has tremendous performance advantages compared to accessing one attribute at a time It is also important to be able to set many attributes within a single transaction without having to begin/commit a JTS User Transaction from the client. In short, pass-by-value is really handy! Our implementation is tightly-coupled: that

means that at compile time, we enforce type consistency for getting/setting attributes in the value objects. Thishas an advantage over "parameter sets"and "late-binding" implementations where you pass around a set of name/value pairs: with these approaches, if you change the type of an attribute your client will still compile but crash at runtime. This won't happen using WebLogic Components, since the value object will change accordingly and the client would not compile if an assignment was illegal. In addition, our value objects are generated by BEA SmartGenerator (based on a UML model), so they don't add any maintenance costs.

Pass By Value Example

The PassByValue example application performs the following steps:

- Find or create or a Customer component
- Create a value object and get the CustomerValue.
- Change Customer information
- Set the CustomerValues

To get the most out of this example, first read through `PassByValueExample.java` on our web site. Then you can build it and run it.

See above note:"To build and run any of these examples, you must have the following in your CLASSPATH:" on page 3-3

More Information

For further information please see the following:

- Appendix A of this document: References
- Frequently Asked Questions (F.A.Q.)
- The complete API javadoc

4 Deploying WebLogic Components Using Bean-Managed Persistence

This topic includes the following sections:

- Using Bean-Managed Persistence
- The Oracle Reference Implementations
- Deployment Sets Overview
- Deploying on Windows NT
- Deploying on Solaris

Using Bean-Managed Persistence

You can deploy the WebLogic Components examples using Bean-Managed Persistence.

It is now possible to map WebLogic Components to any database available through JDBC. BEA provides **two reference implementations** (*deployment sets*) for bean-managed persistence. These are available as deployment options in the WebLogic Components release. For more technical background, read about Deploying WebLogic Components with Enterprise Data Sources.

Here, we describe how to deploy the WebLogic Components software in the following environments:

- Deploying on Windows NT
- Deploying on Solaris

The example Oracle deployment set allows you to map the **My BuyBeans.com** components to Oracle 8.0.5. The Oracle 8.1.5 reference implementation uses serialization to persist most of the attributes of Enterprise Java Beans.

BEA's **Smart Generator** has basic object/relational mapping features that generate up to 100% of the mapping from beans to a relational model using serialization. For complex databases the generated code serves as a starting point for reliable and scalable Bean-Managed Persistence that you can modify and fine-tune.

The Oracle Reference Implementations

The Oracle reference implementation uses Oracle's object-relational features including database object types and nested tables. WebLogic Components are stored with a maximum transparency. You can query on every field in the example BuyBeans object model. This reference implementation is made available to help gauge the effort and complexity of more-detailed object-relational persistence.

If you have any comments, questions, or need help with a WebLogic-Oracle 805 deployment or a WebLogic-Oracle 815 deployment, please contact support@theorycenter.com.

- **Deploying Oracle 8.1.5.** The instructions in this document are specific to the WebLogic-Oracle 805 deployment. If you wish to deploy using WebLogic-Oracle 815, the instructions are exactly the same, except that file names and directory names contain the string '815' instead of '805'. For example, the filename 'BuyBeansOracle805.sql' becomes 'BuyBeansOracle815.sql'.

- **Oracle OCI Driver.** The WebLogic-Oracle 805 deployment uses the Oracle Thin JDBC driver. If you want to use the Oracle OCI driver instead, you must re-run the ejb compiler and redeploy your beans using isolationLevel TRANSACTION_READ_COMMITTED in the deployment descriptors.

Additional Requirements

- BuyBeans.com examples successfully deployed using default container-managed persistence on the Windows NT or the Solaris operating systems.
- Oracle Thin driver for JDBC. As of this writing, these drivers can be downloaded for free from Oracle

Deployment Sets Overview

Deployment Sets give you the freedom to develop your business application independently from the application server or database vendors.

This separates business logic from the development environment and gives you the freedom to choose to develop in one environment and deploy in another. Each Deployment Set pertains to a particular combination of an application server and database.

There is a deployment set available for each of BEA's certified implementations on an application server and database. Deployment sets are available for:

- **WebLogic**
  - Using Container-Managed Persistence to a Cloudscape database
  - Using Bean-Managed Persistence to an Oracle 805 Database
  - Using Bean-Managed Persistence to an Oracle 815 Database

The following table illustrates the matrix of servers and databases.

Table 4-1 Servers and Databases

Databases	Application Servers	
Databases	WebLogic	NAS
Cloudscape	CMP	n/a
Oracle 8.0.5	BMP	under development
Oracle 8.1.5	BMP	under development

Deploying on Windows NT

This reference example assumes you have installed the WebLogic Components™ CD or the download under `c:\weblogicac`.

1. Edit `c:\weblogicac\bin\win32\sethome.bat`
   - Change `DEPLOYMENT_SET` to `%WEBLOGICAC_HOME%\deploy\weblogic\oracle805\classes`
   - Change `DB_CLASSPATH` to your Oracle thin driver classpath.
   - Remove the `DEPLOYMENT_SET` and `DB_CLASSPATH` definitions for the Cloudscape deployment set.
2. Create the *BuyBeans schema* in your Oracle 8.0.5 or higher database.
   - Use `c:\weblogicac\deploy\weblogic\oracle805\misc\BuyBeansOracle805.sql`
3. Change the **`weblogic.properties`** file.
   - Rename the file `c:\weblogicac\weblogic.properties` to `cloudscape-weblogic.properties`. This saves the file so you can return to the CMP version at a later time.

- Copy the file `c:\weblogicac\deploy\weblogic\oracle805\misc\weblogic.properties` to `c:\weblogicac\weblogic.properties`.

4. Replace the `@WEBLOGICAC_HOME@` marker with c:\weblogicac (assuming you installed in c:\weblogicac).

5. Edit the file `c:\weblogicac\weblogic.properties`.

   - Search for the properties named `weblogic.jdbc.connectionPool.portal` and `weblogic.jdbc.connectionPool.jdbcPool`. These are examples for using either the OCI or Thin driver. Modify the connection pools to use your database and password. See http://www.weblogic.com/docs/classdocs/API_jdbct3.html#connpools for further help.

6. **Start the WebLogic server** using `c:\weblogicac\StartPortal.bat`.

7. Verify the following in the server log:

   - Verify that the portal and jdbcPool connection pools were successfully created.
   - Verify that all EJBs successfully deployed.

8. Load the **database**

   - Use `c:\weblogicac\bin\win32\itemloader.bat` with the following qualifiers:
     - **-url** use the name and port of the machine where you deployed the weblogic server. Example: `http://localhost:7601/`
     - **-tabdelimitedfile** `c:\weblogicac\deploy\weblogic\oracle805\misc\BuyBeans.txt`.
   - Use `c:\weblogicac\bin\win32\shippingmethodsloader.bat` with the following qualifiers:
     - **-url** use the name and port of the machine where you deployed the weblogic server. Example: `http://localhost:7601/`
     - **-commadelimitedfile** `c:\weblogicac\deploy\weblogic\oracle805\misc\shippingmethods.txt`

9. **Start your web browser**. Enter the url with the name and port of the machine where you deployed the WebLogic server. You will have to register a new user with **username** "*cool*" and **password** "*bean*".

10. You are now using BEA's Bean-Managed Persistence!

Deploying on Solaris

This example assumes you have installed the WebLogic Components™ download into `/weblogicac`.

1. Edit `/weblogicac/bin/solaris2/sethome.sh`

   - Change `DEPLOYMENT_SET` to `$WEBLOGICAC_HOME/deploy/weblogic/oracle805/classes`
   - Change `DB_CLASSPATH` to your Oracle thin driver classpath.
   - Remove the `DEPLOYMENT_SET` and `DB_CLASSPATH` definitions for the Cloudscape deployment set.

2. **Create the BuyBeans schema** in your Oracle 8.0.5 or higher database.

   - Use `/weblogicac/deploy/weblogic/oracle805/misc/BuyBeansOracle805.sql`

3. Change the **`weblogic.properties`** file.

   - Rename the file `/weblogicac/weblogic.properties` to `cloudscape-weblogic.properties`. This saves the file so you can return to the CMP version at a later time.
   - Copy the file `/weblogicac/deploy/weblogic/oracle805/misc/weblogic.properties` to `/weblogicac/weblogic.properties`.

4. Edit the file `/weblogicac/weblogic.properties`.

   - Search for the properties named `weblogic.jdbc.connectionPool.portal` and `weblogic.jdbc.connectionPool.jdbcPool.` These are examples for using either the OCI or Thin driver. Modify the connection pools to use

your database and password. See http://www.weblogic.com/docs/classdocs/API_jdbct3.html#connpools for further help.

5. **Start the WebLogic server** using `/weblogicac/StartPortal.sh`.
6. Verify the following in the server log:
   - Verify that the portal and jdbcPool connection pools were successfully created.
   - Verify that all ejb's successfully deployed.
7. **Load the database**
   - Use `/weblogicac/bin/solaris2/itemloader.sh` with the following qualifiers:
     - **-url** use the name and port of the machine where you deployed the weblogic server. Example: `http://localhost:7601/`
     - **-tabdelimitedfile** `/weblogicac/deploy/weblogic/oracle805/misc/BuyBeans.txt`.
   - Use `/weblogicac/bin/solaris2/shippingmethodsloader.sh` with the following qualifiers:
     - **-url** use the name and port of the machine where you deployed the WebLogic server. Example: `http://localhost:7601/`
     - **-commadelimitedfile** `/weblogicac/deploy/weblogic/oracle805/misc/shippingmethods.txt`
8. **Start your web browser**. Enter the url with the name and port of the machine where you deployed the WebLogic server. You will have to register a new user with **username** "*cool*" and **password** "*bean*".
9. You are now using BEA's Bean-Managed Persistence!

A References

Here are a few links that will help you learn more about these exciting technologies:

Sites

- Enterprise JavaBeans - http://www.java.sun.com/products/ejb/index.html
- BEA Weblogic - http://weblogic.beasys.com
- Cloudscape - http://www.cloudscape.com

EJB Documentation

- EJB Specifications - http://www.java.sun.com/products/ejb/docs.html

White Papers

- Deploying eBusiness Smart™ Components with Enterprise Data Sources”
- Enterprise JavaBeans™: Server Component Model for Java” - http://www.javasoft.com/products/ejb/white_paper.html

- Business Smart™ Components: A family of Enterprise JavaBeans for Mission-Critical Applications”