

# WebLogic User Tasks

Magoo Software Blueprint  
August 2005

## Copyright Notices

Copyright © 2005 Magoo Software. All rights reserved. The document is not intended for production and is furnished “as is” without warranty of any kind. All warranties on this document are hereby disclaimed including the warranties of merchantability and fitness for a particular purpose.

## Trademarks

MagooClient is a trademark of Magoo Software Limited. Sun , Java , J2EE, and all Java-based trademarks or logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both. Oracle is a trademark or registered trademark of Oracle Corporation in the United States, other countries, or both. BEA, WebLogic, and BEA WebLogic Server and WebLogic Workshop are trademarks or registered trademarks of BEA Systems, Inc. IBM and IBM WebSphere are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both. Other company, product, and service names mentioned in this product may be trademarks or service marks of others.

# Contents

1	Introduction.....	4
2	Architectural Overview .....	5
3	Creating the Web Services .....	6
3.1	WebLogic Setup.....	6
3.2	Creating the User Queue .....	6
3.3	Creating the OrderSystem Web Service (OrderSystem.jws).....	6
3.4	Creating the ShippingSystem Web Service (ShippingSystem.jws).....	7
4	Configuring a MagooClient JMS Transport .....	8
5	Adding the EasyPO Schema to MagooClient.....	10
6	Configuring the ShippingSystem Address .....	11
7	Dispatching the Purchase Order back to the Shipping System.....	12

# 1 Introduction

This tutorial is intended to show how interaction with a user via a JMS queue can be used as a mechanism for correcting errors in an order-fulfillment process. This is a common issue within business processes where application logic is incapable of handling particular errors and human intervention is required.

This demo is based on a WebLogic 8.1 Windows installation and some knowledge of WebLogic is assumed. Code samples and instructions are provided to allow you to build the demo yourself. As the JMS API provides a standard means for interacting with messaging providers, the steps for configuring MagooClient will be similar for other JMS-compliant vendors. Additional detailed demos covering other JMS implementations will be added shortly.

The tutorial consists of the following stages:

- [Architectural Overview](#)
- [Creating the Web Services](#)
- [Configuring a MagooClient JMS Transport](#)
- [Adding the EasyPO Schema to MagooClient](#)
- [Configuring the ShippingSystem Address](#)
- [Dispatching the Purchase Order back to the Shipping System](#)

## 2 Architectural Overview

In concrete terms, fault-handling is based on an order processing application ('OrderSystem') placing the invalid document onto a user JMS queue. MagooClient is used to lift the message off the queue, correct any errors and place the document back onto a second shipping queue for processing by a JMS Web Service ('ShippingSystem').

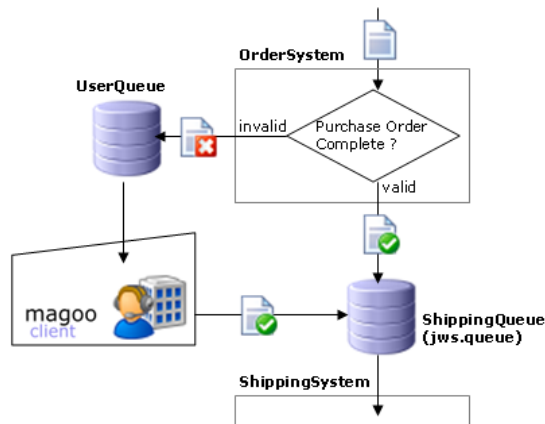
As we'll see, the core advantages of the MagooClient approach are:

- No need to develop or maintain expensive custom GUI or portal code. MagooClient can automatically render and validate messages based on Schema or WSDL information. Using MagooClient also means that the document can be easily extended at a later stage without needing to upgrade client code.
- The model is completely asynchronous - the order processing application is not blocked while the user performs the correction.
- Minimal server side development is required - built-in JMS support means that interaction with MagooClient is achieved via a simple call to a JMS Control using the popular WebLogic XMLBeans model. This can be added with a single line of code.
- Use of XMLBeans allows a Schema-driven development model can be adopted. This means that the full power of Schema can be used to control the document contents thus providing a valuable validation capability for user input.

The key elements of the demo architecture are:

- **EasyPO.xml:** The schema document for the Purchase Order.
- **UserQueue:** A JMS Queue specifically configured to queue Purchase Order documents for processing by users.
- **OrderSystem Web Service:** An application which detects errors in Purchase Order documents and places them onto the UserQueue for correction.
- **ShippingSystem Web Service:** This accepts completed, valid orders for processing via JMS. The Web Service implementation is based on a standard WebLogic Web Service with the protocol tag set to 'jms-xml' to indicate that it accepts requests on a JMS queue.
- **Web Service JMS Queue (Shipping Queue) :** This is the default queue ('jws.queue') on which WebLogic accepts JMS Web Service requests. Logically this can be considered as the ShippingQueue - the user places corrected documents onto this queue once editing is completed.
- **MagooClient:** Used to receive, edit, validate and send Purchase Orders.

The message flow is summarized below:



## 3 Creating the Web Services

### 3.1 WebLogic Setup

For convenience, this tutorial and sample code is based on the Workshop Sample Applications which ship with WebLogic 8.1. This means that the required Web Service input queue 'jws.queue' and required Queue Connection Factory ('weblogic.jws.jms.QueueConnectionFactory') are already configured for the associated workshop domain. Also the Purchase Order Schema (EasyPO.xsd) is already available and compiled to an XMLBean in the Schema project under SamplesApp.

From the Windows start menu, start Workshop using:  
BEA WebLogic Platform 8.1->Examples->WebLogic Workshop->Start Workshop with Sample Applications.

### 3.2 Creating the User Queue

Start the WebLogic Server Console from the Windows start menu using:  
BEA WebLogic Platform 8.1->Examples->WebLogic Workshop->Server Admin Console

This will start the console for the workshop domain. Select 'Services'-'>'JMS'-'>'Servers'-'>'cgJmsServer'-'>'Destinations' and add a new queue. **The physical Queue name and JNDI name should be set to 'UserQueue'.**

### 3.3 Creating the OrderSystem Web Service (OrderSystem.jws)

The OrderSystem is based on a simple Web Service (which we just use for starting the process). It creates an incomplete Purchase Order and places it on the UserQueue using a JMS Control. The OrderSystem therefore consists of two parts: a Web Service with a simple 'start' method which creates the EasyPO Purchase Order document and a JMS Control (UserQueueControl.jcx) to place it onto the UserQueue.

First create a WebLogic Web Service (OrderSystem.jws) in the 'xmlBeans' folder of the 'WebServices' project under the SampleApp application folder. In design view, insert a new JMS Control using the 'Insert Control - JMS' wizard. The JMS control should have the following properties:

- Variable Name: **userQueueControl**
- New JCXName: **UserQueueControl**
- Message Type: **Text/XMLBean**
- JMS send Destination type: **Queue**
- send-jndi-name: **UserQueue**
- connection-factory: **weblogic.jws.jms.QueueConnectionFactory**

The control should appear on the left of the OrderSystem Web Service in the Design View. Using the Design View now add a 'start' method to OrderSystem.jws. Switch to Source View and fill out the implementation as follows:

```
package xmlBeans;
import org.openuri.easypo.*;
import org.openuri.easypo.PurchaseOrderDocument.PurchaseOrder;
public class OrderSystem implements com.bea.jws.WebService
{
    /**
     * @common:control
     */
    private xmlBeans.UserQueueControl userQueueControl;
    static final long serialVersionUID = 1L;
    /**
     * @common:operation
     */
    public void start() {
```

```

PurchaseOrderDocument poDoc = PurchaseOrderDocument.Factory.newInstance();
PurchaseOrder po = poDoc.getPurchaseOrder();
if(po == null) {
    po = poDoc.addNewPurchaseOrder();
    Customer customer = po.addNewCustomer();
    customer.setName( "jbloggs" );
}

// Dispatch to UserQueue
userQueueControl.sendMessage(poDoc.toString());
}
}

```

This start() method is purely for demo purposes and provides a means for initiating a message drop onto to the UserQueue. Normally the decision logic for initiating message dispatch would be part of a much broader OrderSystem application.

### 3.4 Creating the ShippingSystem Web Service (ShippingSystem.jws)

Create a WebLogic Web Service (ShippingSystem.jws) in the 'xmlBeans' folder of the 'WebServices' project under the SampleApp application folder. In design view, add a new method 'acceptPurchaseOrder'. Click on this to enter the Source View.

By default, new Web Services accept requests using the form-post, form-get and http-soap protocol settings. In this case however we want the ShippingSystem to read from a JMS Queue. Therefore in the Property Editor, **set the jms-xml protocol to true** and all others to false. Note - you must **use the jms-xml and NOT the jms-soap or jms-soap12 protocols** as the document being exchanged is plain XML based on XMLBeans rather than a SOAP message. Switch to the source view and add in some simple implementation code to write the received document to the console (System.out) so we can see the completed document.

```

package xmlBeans;
import org.openuri.easypo.*;
import org.openuri.easypo.PurchaseOrderDocument.PurchaseOrder;
/**
 * @jws:protocol http-soap="false" form-get="false" form-post="false" jms-xml="true"
 */
public class ShippingSystem implements com.bea.jws.WebService {
    static final long serialVersionUID = 1L;
    /**
     * @common:operation
     */
    public void acceptPurchaseOrder(PurchaseOrderDocument poDoc) {
        // Print to console so we can see the completed document.
        System.out.println("Completed PO received:\n" + poDoc.toString());
    }
}

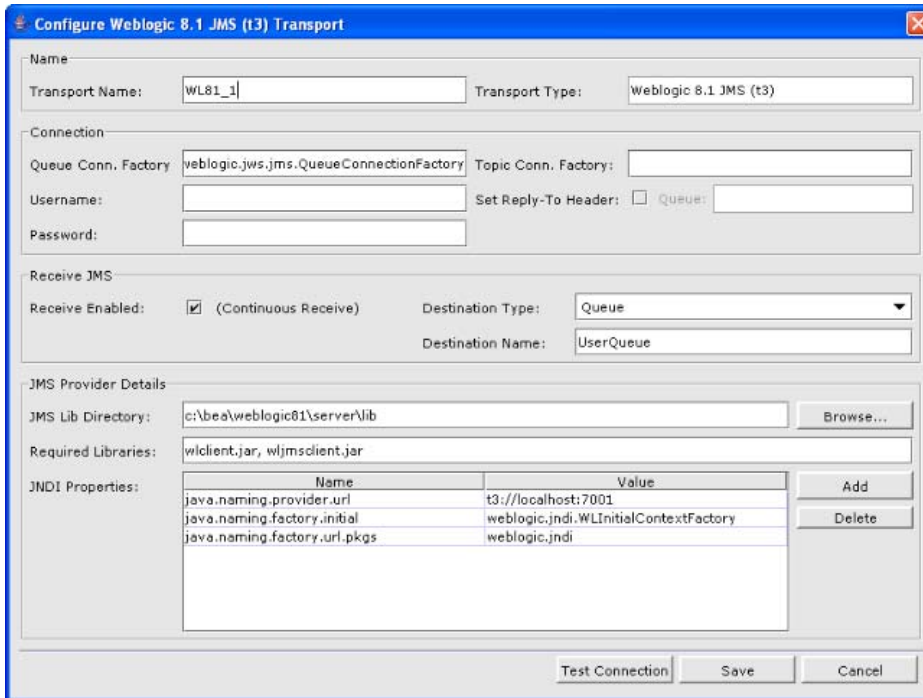
```

Note in the code above that the jws:protocol annotation has all of the default protocols set to false and only jms-xml set to true. Also note that it is not necessary to create an inbound JMS Queue for the ShippingSystem. By default, WebLogic Server listens for JMS Web Service requests on the 'jws.queue' Queue and routes them to the Web Service implementation automatically based on properties within the JMS message. We'll cover how a particular Web Service is addressed later in 'Configuring a MagooClient JMS Address'.

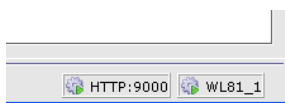
Press the Start button (Ctrl + F5) to compile and deploy the new Web Services and the UserQueueControl to the WebLogic Server. This completes the WebLogic setup.

## 4 Configuring a MagooClient JMS Transport

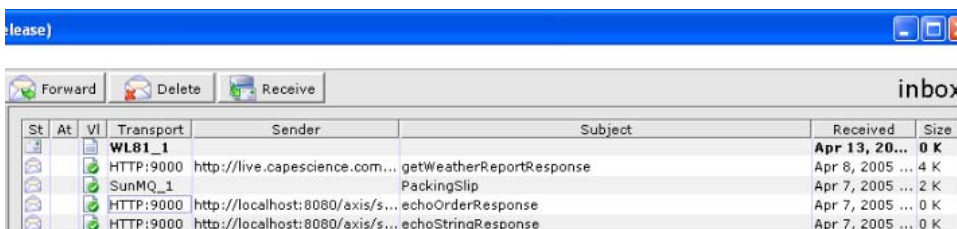
In order for MagooClient to communicate with a JMS Queue, it is necessary to configure an appropriate JMS transport. This allows messages to be sent and received using a particular protocol or provider. In the MagooClient main menu select 'Configure'-'>'Communications...'. This will provide a list of configured transports. Click on the Add button and select 'WebLogic 8.1 JMS (t3)' as the required transport type. The JMS Configuration Dialog will now appear:



If you have installed WebLogic in the default install directory then the only setting you will need to enter is the Queue name on which the transport will listen for messages. This should be set to 'UserQueue'. The Queue Connection Factory is by default set to the Workshop setting. Click on the 'Test Connection' button and you should see a 'Connection successfully established' message. Save the JMS transport settings. The status bar in the main MagooClient window should now display an enabled icon for the WL81\_1 transport:



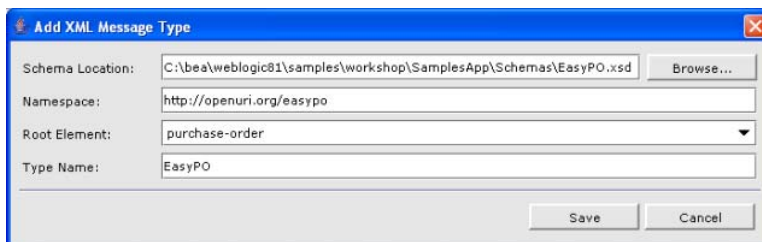
You can now test that MagooClient is receiving messages by starting the Workshop Test Browser and invoking the start method. This creates a Purchase Order and places it on the UserQueue. This will be picked up by MagooClient and should now appear in the MagooClient inbox.



The status icon for the new message indicates that it is an XML document but that it is not validated i.e. MagooClient cannot identify the document type. Double-click on the document to open it - the XML content will be rendered as a form however the pop-up functionality on the tree is not enabled. The reason for this is that MagooClient has not been told where to find schema information relating to this XML document. We will do this in the next stage.

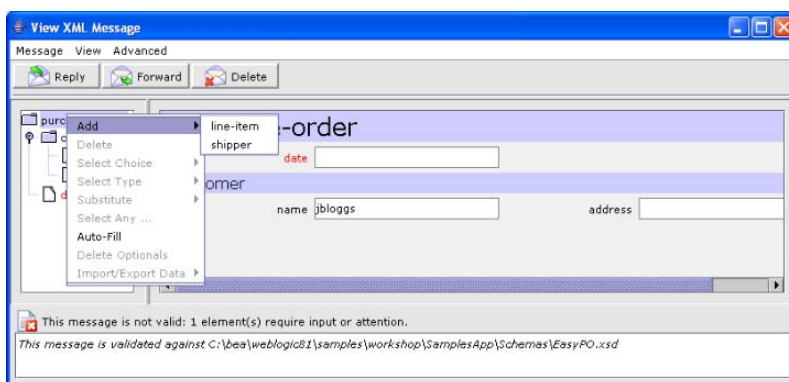
## 5 Adding the EasyPO Schema to MagooClient

In order for MagooClient to recognize and validate documents, it must first be configured with the Schema or WSDL information. In this case the schema is EasyPO.xsd located in the WebLogic Samples/Workshop/SamplesApp/Schemas directory. In the MagooClient main menu select Configure->Message Types to view the Message Types Catalogue. Select 'Add' and 'XML Schema (XSD)' as the New Message Type. Browse to the Schema directory and select EasyPO.xsd. Enter 'EasyPO' as the Type Name.

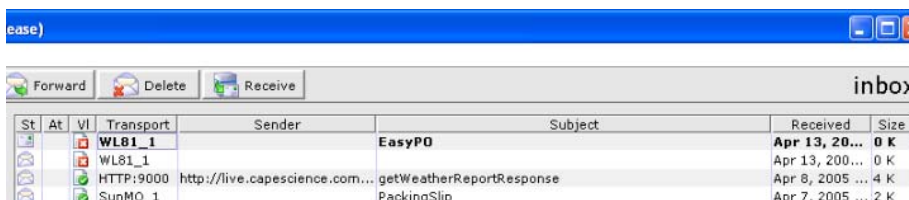


Save the new XML Message Type - it should now appear in the list of configured message types. To check that EasyPO documents are now being recognized, double-click on the new message in the inbox once again. This time the message will appear with an error as it is now being validated against EasyPO.xsd. The error is flagged because the date field is of the schema type dateTime which must be of the form CCYY-MM-DDThh:mm:ss e.g. 2004-12-03T14:54:03 - it cannot be left empty. Also the pop-up menu on the navigation tree now offers the option to add a line-item or shipper element to the purchase-order root node.

In addition the Auto-Fill option appears which will populate fields with valid values. In this case it can be used to automatically insert the current date and time into the date field.



To test that the MagooClient Message Processor is also validating automatically, start the WorkShop Test Browser and invoke the start method again on the OrderSystem.jws. This time the new message will appear with the validation status set to invalid because MagooClient can now recognize the document. Also, to provide the user with some context, the message type name is displayed in the Subject column.



## 6 Configuring the ShippingSystem Address

The final part of the scenario involves sending EasyPO documents to the ShippingSystem by placing them on the appropriate queue. As indicated above, WebLogic Server automatically polls the 'jws.queue' Queue for inbound JMS requests.

However in addition to using the correct Queue, it is vital to let WebLogic know which Web Service should receive the message. This is done for WebLogic 8.1 by setting a String Property in the message header which identifies the target Web Service (this is similar to setting a soapAction header on a HTTP Web Service request). The property in question is the **URI** property and this should be set to **/WebServices/xmlBeans/ShippingSystem.jws**

To do this, it is necessary to set up an appropriate address in the MagooClient address book (this also spares the user from having to remember the details all the time!). In the MagooClient main window, select Configure->Address Book. Select Add and then JMS as the required Address Type. Fill out the Address details as follows:

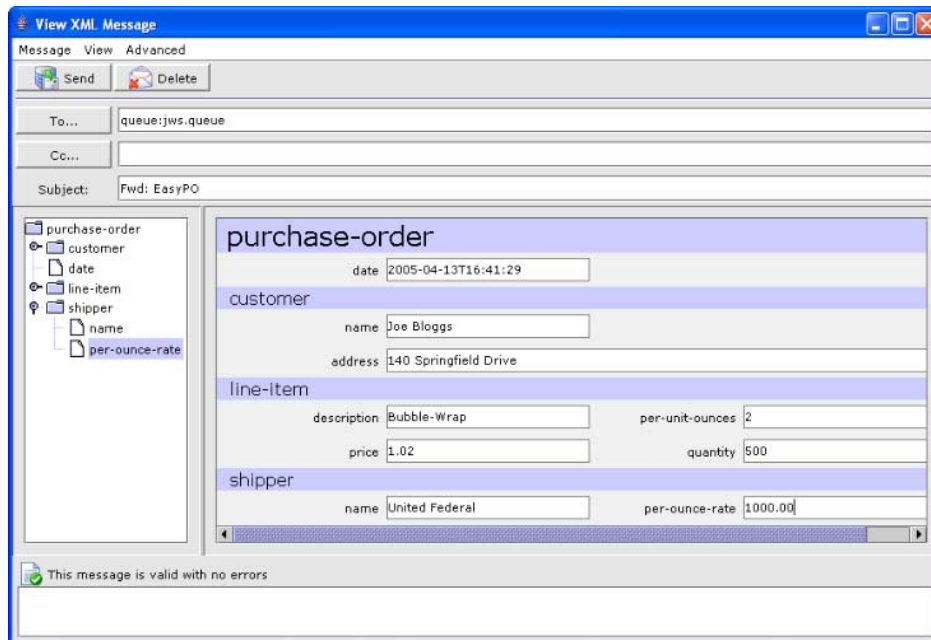
Name	Value
URI	/WebServices/xmlBeans/ShippingSystem.jws

The Display Name is used as a shortcut when entering addresses in the Message Viewer - it will be automatically expanded out if entered by the user. As indicated above, the URI property must be added and set correctly. The 'Send Using' field should be set to WL81\_1 which is the JMS Transport we configured earlier. Note that although a JMS Transport can only listen to one particular Queue or Topic, it can be used to send to any Queue or Topic as long as the QueueConnectionFactory or TopicConnectionFactory have been set-up (to listen on multiple Queues or Topics, simply configure multiple JMS Transports, one per Queue or Topic). Save the JMS Address details - the new address should now appear in the Address Book.

## 7 Dispatching the Purchase Order back to the Shipping System

To place the EasyPO back on the ShippingQueue, select the message in the inbox and click on the Forward button. This will open the XML Editor with the draft to be forwarded. Edit the document as required - using the pop-up menu on the left hand XML navigation tree allows you to add and delete elements such as 'line-item' and 'shipper' as required.

Then click on the 'To:' button and select the 'shipq' JMS Address from the Address Book. The completed document in the XML Editor should now look something like this:



Click on the send button. MagooClient will put the document onto the ShippingQueue and you should see the output from the ShippingService.jws appearing in the WebLogic console indicating that the message has been successfully processed. The message flow from OrderSystem to the user and back to the ShippingSystem has been completed.

