



Delivery Service Blueprint: Integrating StrikeIron Web Services with MagooClient

Magoo Software Blueprint
May 2006

Copyright Notices

Copyright © 2006 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	Integrating StrikeIron Services.....	4
1.1	Introduction	4
1.2	Benefits	4
1.3	About StrikeIron.....	4
1.4	Blueprint Storyboard.....	4
1.5	Resources.....	5
2	Creating the DeliveryService	6
2.1	Defining the Delivery Service WSDL.....	6
2.2	Implementing the Delivery Service.....	6
2.3	Installation Requirements.....	6
2.4	Installing the Blueprint Components.....	6
2.5	Anatomy of an Axis WSDL2Java Build file.....	6
2.5.1	Build Properties.....	7
2.6	Running the Build File.....	7
2.7	Registering the DeliveryService Operations.....	7
2.8	Testing the DeliveryService Operations.....	8
2.8.1	lookupByItemCode Operation	8
2.9	createOrder Operation.....	9
3	Adding StrikeIron Reverse Phone Lookup.....	11
3.1	About Reverse Phone Lookup	11
3.2	Reverse Phone Lookup Test Invocation.....	11
3.3	Integrating Reverse Phone Lookup	13
3.3.1	Creating the ReverseLookup script	13
3.3.2	Adding a ScriptAction.....	15
4	Adding Item Lookup.....	17
4.1	Determining the Selected Order Item	17
5	Appendix: Code Listings.....	18
5.1	DeliveryService.xsd.....	18
5.2	DeliveryService.wsdl.....	18
5.3	DeliveryServiceBindingImpl.java.....	20
5.4	build.xml	20
5.5	itemLookup.js.....	22
5.6	reverseLookup.js.....	23

1 Integrating StrikeIron Services

1.1 Introduction

The MagooClient scripting environment, based on the JavaScript + XML ([E4X](#)) standard, can be used to implement sophisticated business logic to support user input for XML documents. This blueprint describes how Web Services invocations can be easily added through scripting to provide dynamic form-filling and data validation. In particular, this blueprint covers:

- Integrating a StrikeIron Web Service invocation using simple JavaScript
- Configuring Script Actions to provide instant script invocation on form-entry
- Creating a sample Web Service using a WSDL-first design approach
- Implementing form calculations and updating XML structure using JavaScript
- Using XSL stylesheets to customize form look and feel

1.2 Benefits

One of the key benefits of the MagooClient approach is that there is no need for a middle-tier to drive interaction between MagooClient and SOA Services as communication with endpoints is direct. The scripting approach allows for rapid prototyping of service interaction and form-handling from within the client environment. In addition, the use of client-side scripting to perform calculations and to manipulate document content saves on server round-tripping and load.

Once the scripts have been completed, they can be easily deployed across multiple client instances using MagooClient's remote configuration capability.

1.3 About StrikeIron

StrikeIron is the worldwide leader in Web services commercialization with its breakthrough [StrikeIron Web Services Marketplace](#) that greatly simplifies the selling and buying of Web services for a broad audience of providers and users, while simultaneously supporting commercial Web services integration by ISVs.

Overall, the StrikeIron Web Services Marketplace provides an integrated set of capabilities designed to bring together providers, users and partners as part of a community to accelerate the adoption of commercial Web services. Key to this adoption is the StrikeIron Web Services Marketplace as the preferred central location for Web services commerce where publishers and users can come together to sell and buy Web services on top of a powerful technology platform.

Based in Research Triangle Park, North Carolina, StrikeIron was founded by veteran entrepreneurs with more than fifty years of combined experience in building successful technology companies. For more details on StrikeIron, please see www.strikeiron.com.

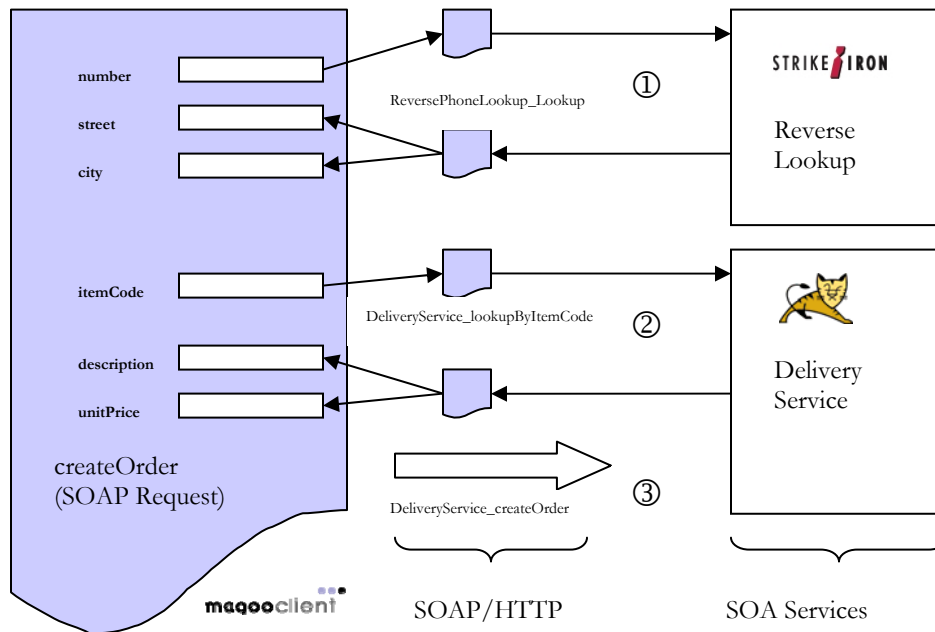
1.4 Blueprint Storyboard

The blueprint scenario involves creating an active form, based on an underlying SOAP/XML document, which reacts to user input during data-entry. The scenario simulates a simple DeliveryService where orders are taken over the phone and logic is required to support accurate data-entry. In terms of actual logic, the form requires the following:

- Automatic checking of contact phone number against address (Reverse Phone Lookup). This capability is available as a StrikeIron Web Service which will be integrated using JavaScript. Invocation should be automatic on entering a phone number into the appropriate form-field.
- Auto-fill of description and unit price details on entering of an order item. This is supported through invocation on a custom order service implemented as a Web Service.
- Dynamic calculation of totals on data-entry.

- Dynamic form behaviour with automatic addition of new blank order item fields on completion of the last item in the list.
- Data validation against XML Schema throughout.

The following diagram shows how the SOA Services, DeliveryService and StrikeIron Reverse Lookup are used via the MagooClient Scripting capability to populate a SOAP message. The completed SOAP message is in turn dispatched to a SOA Service (DeliveryService).



1.5 Resources

Full code listings for all of the items required to build the DeliveryService are provided in the [appendix](#) at the end of this document. For convenience, a zip file containing the WSDL, build script and completed code is provided for [download](#).

2 Creating the DeliveryService

The core Web Service used to process orders in this blueprint is the Delivery Service. For this scenario, the Delivery Service needs to implement two operations:

- Create Order (createOrder): this accepts a completed order and processes it. An order confirmation is returned to the user. The createOrder request message is the underlying XML basis for data entry.
- Lookup Item Code (lookupByItemCode): this supports order creation by performing a lookup for the entered item code e.g. 'T08', returning the item description and unit price.

2.1 Defining the Delivery Service WSDL

The implementation of the Delivery Service follows the WSDL design view in that the Service schema and WSDL are defined first and then automatic code-generation is used to create a template server. The WSDL file ([DeliveryService.wsdl](#)) references a schema definition ([DeliveryService.xsd](#)) which includes all of the necessary types for the SOA service.

2.2 Implementing the Delivery Service

As a WSDL-first design is being adopted and the provided WSDL is compatible with leading Web Service implementations, it should be possible to build and deploy the Service using your preferred Web Service platform (e.g. BEA WebLogic, Cape Clear ESB or Microsoft .NET). For this blueprint, the freely-available Apache Axis 1.3 is used. If you use a different SOAP stack then you will need to change the Service endpoint URLs accordingly.

2.3 Installation Requirements

To build and run the sample Delivery Service using Axis, you will need to install the following on your machine:

- JDK Version 1.4 or above
- [Apache Tomcat 5.0](#) – to use the blueprint build defaults on Windows, this should be installed into the Tomcat default C:\Program Files\Apache Software Foundation\Tomcat 5.0
- [Apache Axis V1.3](#) – to use the blueprint build defaults on Windows, this should be installed into the Axis default C:\axis-1_3
- [Apache Ant](#). The Ant bin directory should be added to the PATH so that ant is executable from the command line.

2.4 Installing the Blueprint Components

Download the [DeliveryService.zip](#) file and extract the contents to your development directory. Note that this includes an impl/deliveryservice sub-directory containing a sample Service implementation file. This file is used during the build process to overwrite the empty implementation file created by the wsdl2java step.

2.5 Anatomy of an Axis WSDL2Java Build file

An Ant build file ([build.xml](#)) is provided which will:

- Generate the Server stub code and deployment descriptor from WSDL
- Merge in the completed Service implementation class
- Compile the entire set of Java classes
- Copy the Service classes to the target Axis deployment directory
- Use the Axis Admin Service to deploy the Service.

2.5.1 Build Properties

All of the properties used throughout the build process are located at the top of the `build.xml`. This should allow the build file to be easily re-used for other Axis Web Services. Details on changing these to suit your particular development environment are provided in the next sections.

2.5.1.1 Axis/Tomcat Properties

The `build.xml` is pre-configured for a default Tomcat 5.0/Axis 1.3 installation on Windows. For other installation directories or operating systems, change the following lines within the `build.xml`:

```
<!-- INSTALLATION SPECIFIC PROPERTIES - CHANGE AS REQUIRED -->
<property name="axis.home" value="c:/axis-1_3"/>
<property name="tomcat.home" value="c:/Program Files/Apache Software Foundation/Tomcat 5.0"/>
<property name="axis.deploy" value="\${tomcat.home}/webapps/axis/WEB-INF/classes"/>
```

2.5.1.2 Package/Namespace Properties

These properties allow the namespace to package mapping to be controlled during the generation of Java service class from WSDL (wsdl2java target). In this case all of the namespaces are mapped into a common package for convenience. To change the package names or mapping details (including re-use for other projects), change the following lines:

```
<!-- SERVICE/PROJECT SPECIFIC PROPERTIES - CHANGE AS REQUIRED -->
<property name="wsdl.url" value="DeliveryService.wsdl"/>
<property name="package.name" value="deliveryservice"/>
<property name="package.path" value="deliveryservice"/>
<property name="xsd.namespace"
  value="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/xsd"/>
<property name="types.namespace"
  value="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/types"/>
<property name="wsdl.namespace"
  value="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/wsdl"/>
```

2.6 Running the Build File

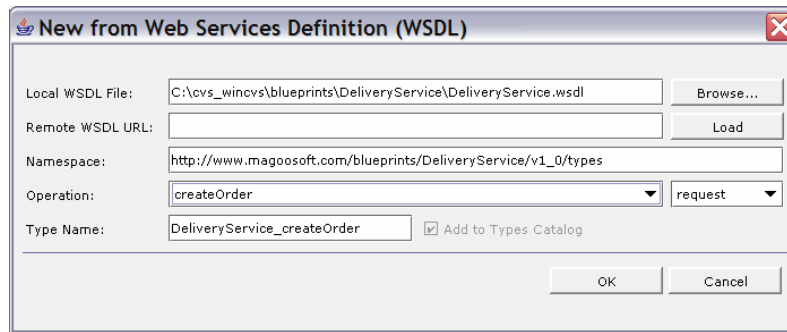
Ensure that your Tomcat/Axis installation is running. On a local machine this can be quickly checked using the 'Axis Happiness Page' – simply point your browser at <http://localhost:8080/axis/happyaxis.jsp>.

An important point to note is that Tomcat/Axis must be re-started whenever the Service implementation classes are updated.

Run the Ant build by typing `ant` at the command prompt - check beforehand that you have added the ant bin directory to your Windows PATH. You will see warnings from Axis indicating that `DataHandler` and `MimeMultipart` classes are not found and attachment support is disabled – these can be ignored as attachment support is not required for the `DeliveryService`.

2.7 Registering the DeliveryService Operations

In order for `MagooClient` to be able to create and validate SOAP message for `DeliveryService`, the WSDL details must first be configured. Start `MagooClient`, select **New from WSDL...** from the **Message** menu in the main `MagooClient` window, browse to the development directory where you unzipped the `DeliveryService.zip` and select `DeliveryService.wsdl`. On selection, the dialog box should be updated with details of the available operations. Select **OK** to register `DeliveryService_createOrder`. Repeat this step to also configure `DeliveryService_lookupByItemCode`.

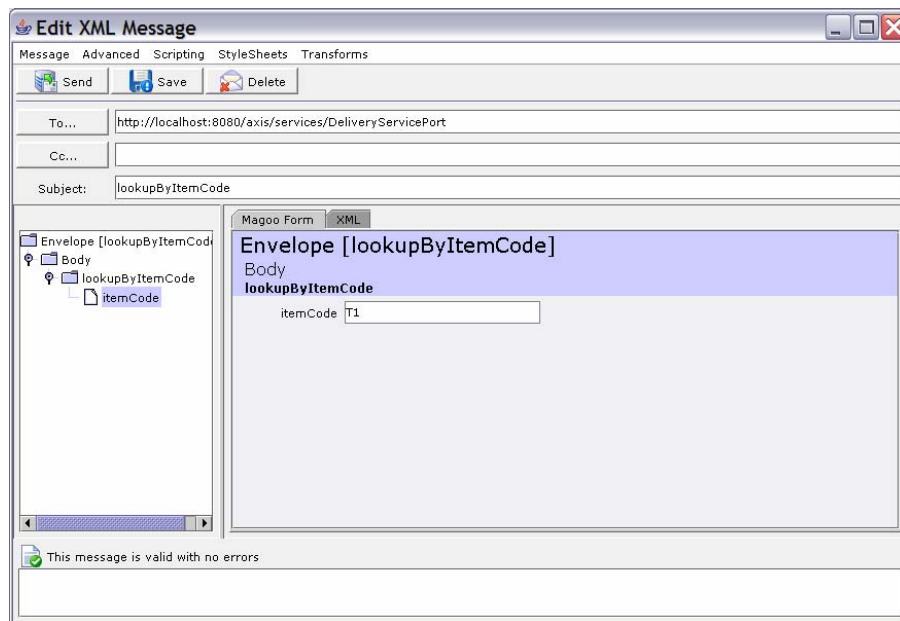


2.8 Testing the DeliveryService Operations

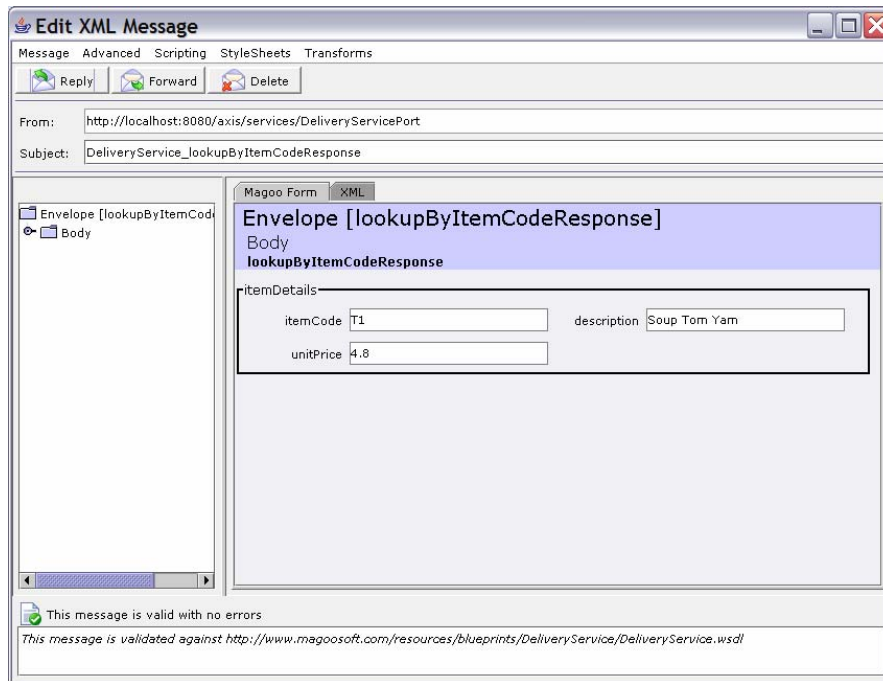
The following invocations test that the DeliveryService is functioning correctly before proceeding with script development.

2.8.1 lookupByItemCode Operation

Click on the **New** button within the MagooClient main window to open the Message Type Catalog. Double-click on the DeliveryService_lookupByItemCode type. A new message will be created – enter 'T1' for the itemCode and click on **Send**. As can be seen from the Service implementation code, this demo implementation only provides lookups for item codes T1 through T10 – feel free to extend this as required!



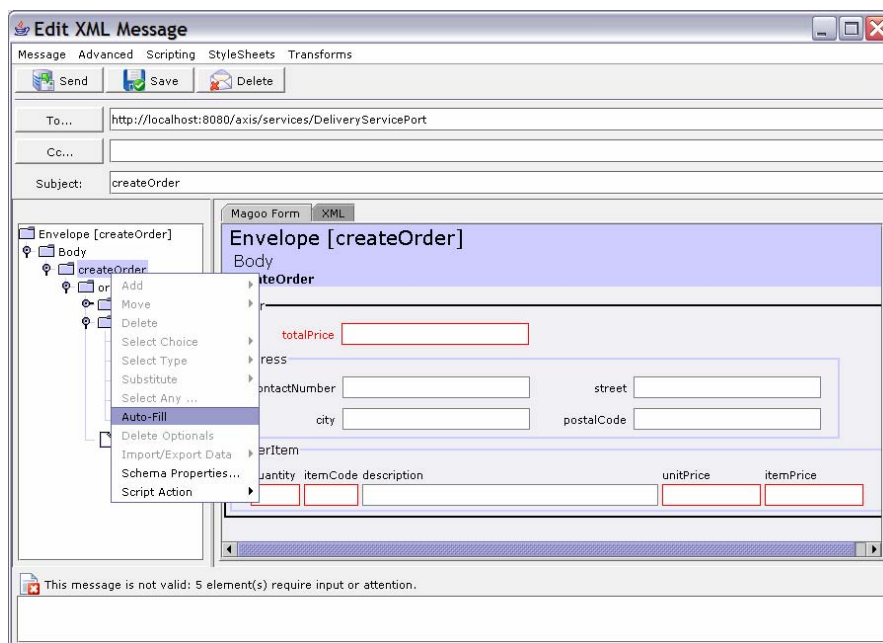
If the DeliveryService is operating correctly then a SOAP response should appear in the MagooClient inbox. Double-click on it to open – the item's description and unit price should be returned correctly.



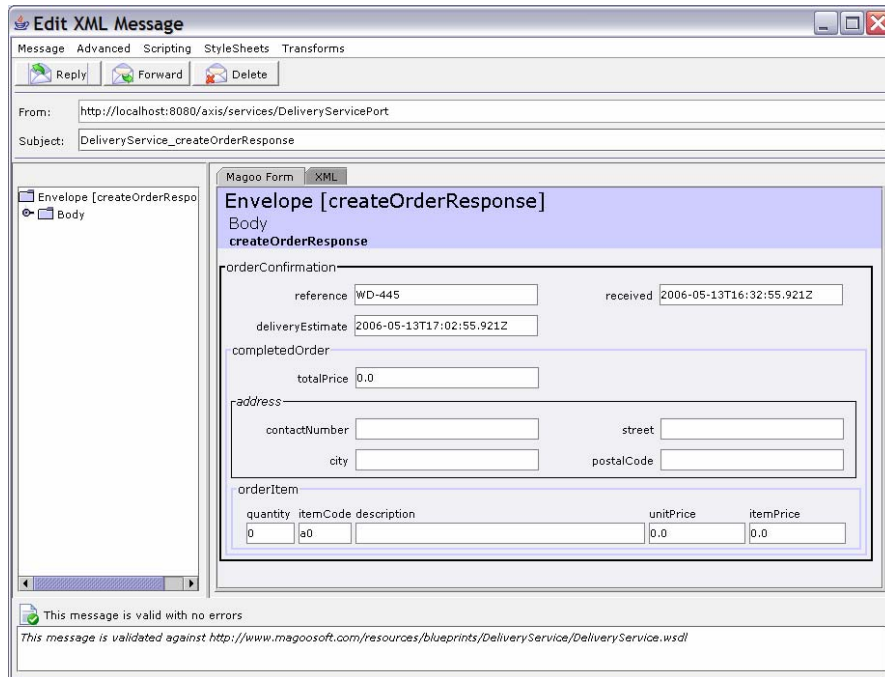
2.9 createOrder Operation

To create a new createOrder SOAP request, select the DeliveryService_createOrder type from the Message Type Catalog. The new message will be automatically opened for editing.

Note that a number of the fields are marked as having validation errors. This is because the underlying WSDL/Schema does not allow some of the elements to be left blank e.g. numerical fields such as totalPrice must contain a valid numerical value. For testing purposes, these errors can be simply cleared by selecting the createOrder node in the left-hand navigation tree and right-clicking to open the pop-up menu. Select **Auto-Fill** and confirm – all of the elements in the message will be populated with schema-valid values.



Click on **Send** to dispatch the request to the DeliveryService. It should respond with a SOAP response message as follows:



The screenshot shows the 'Edit XML Message' window with the following details:

- From:** http://localhost:8080/axis/services/DeliveryServicePort
- Subject:** DeliveryService_createOrderResponse
- Envelope [createOrderResponse]**
 - Body**
 - createOrderResponse**
 - orderConfirmation**
 - reference: WD-445
 - received: 2006-05-13T16:32:55.921Z
 - deliveryEstimate: 2006-05-13T17:02:55.921Z
 - completedOrder**
 - totalPrice: 0.0
 - address**
 - contactNumber: [input field]
 - street: [input field]
 - city: [input field]
 - postalCode: [input field]
 - orderItem**

quantity	itemCode	description	unitPrice	itemPrice
0	a0	[input field]	0.0	0.0

Validation message: This message is valid with no errors. This message is validated against <http://www.magoosoft.com/resources/blueprints/DeliveryService/DeliveryService.wsdl>

3 Adding StrikeIron Reverse Phone Lookup

3.1 About Reverse Phone Lookup

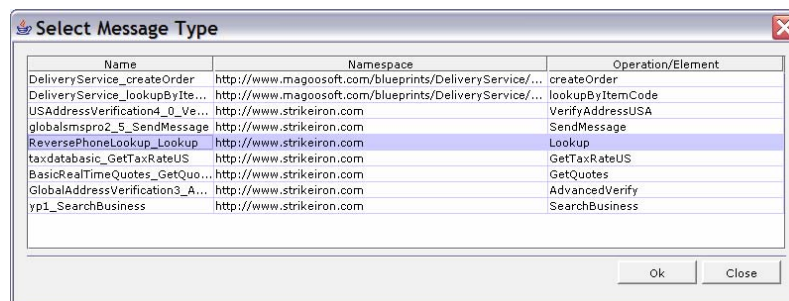
The StrikeIron 24-hour Accurate Reverse Phone Lookup Web Service provides a programmatic interface to name and address data associated to any telephone number, including residential, business, and government numbers in the U.S. and Puerto Rico. This data is updated nightly making them the most accurate and up to date resource of their kind. More details are available from:

<http://www.strikeiron.com/ProductDetail.aspx?p=103>

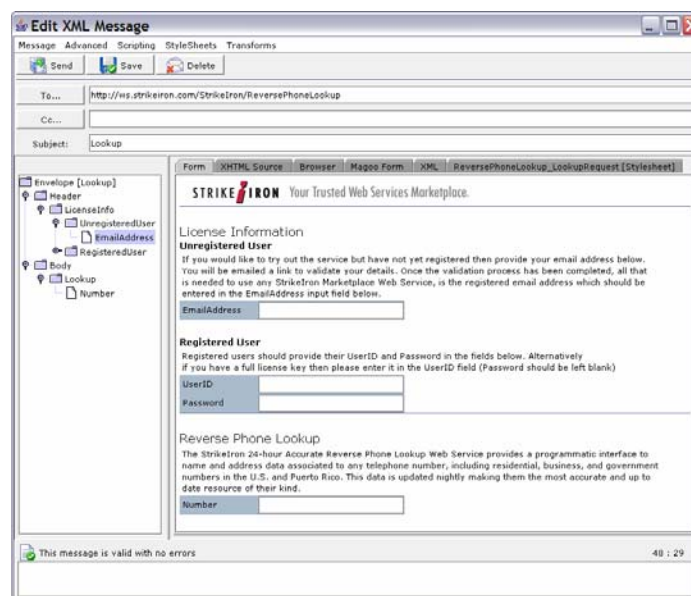
3.2 Reverse Phone Lookup Test Invocation

In this section we will look at how data-entry into the number field within the createOrder SOAP message can be used to drive a reverse phone lookup and corresponding population of the address fields (city, postcode etc). The reverse lookup is achieved using the commercial StrikeIron Service.

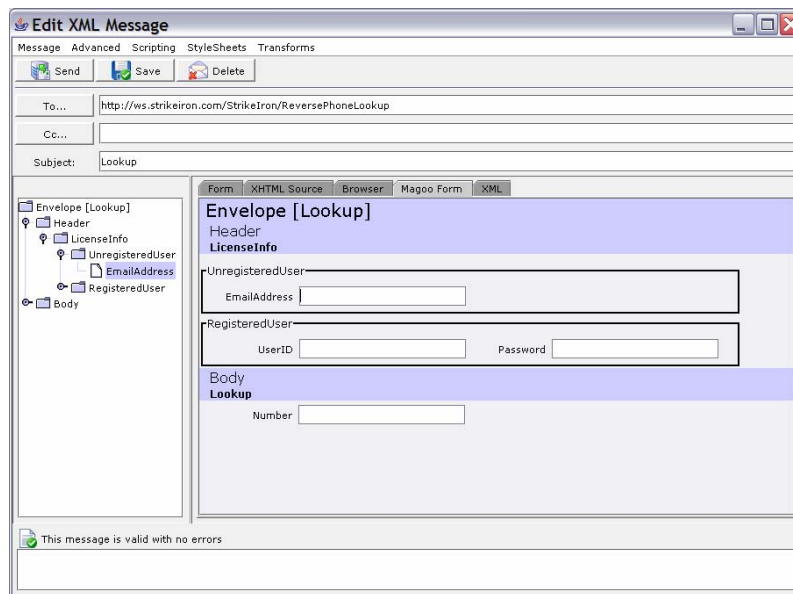
A number of StrikeIron Services, including Reverse Lookup are pre-configured within the MagooClient Message Type Catalog. Click on the **New** button or select **New from Types Catalog...** from the **Message** menu in the main MagooClient window. The Message Type Catalog browser will appear with the list of configured message types. This should include the ReversePhoneLookup_Lookup type.



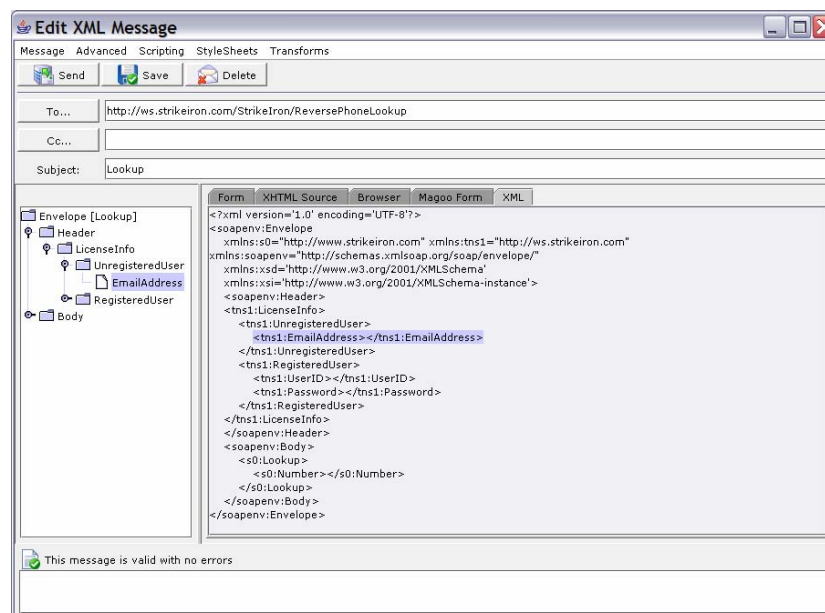
Select the ReversePhoneLookup_Lookup and click on Ok. A new message draft will be created and opened for editing:



Note that this message type has been pre-configured with a stylesheet which allows the look and feel of the form to be customized. The stylesheet hides details such as the SOAP message structure which are not essential for user input. Also a logo and explanatory text have been added to explain Service operation including licensing. More information on using stylesheets to customize form appearance is provided in the Stylesheets section of the [Magoosoft User Guide](#). For comparison the default Magoosoft view is shown below:

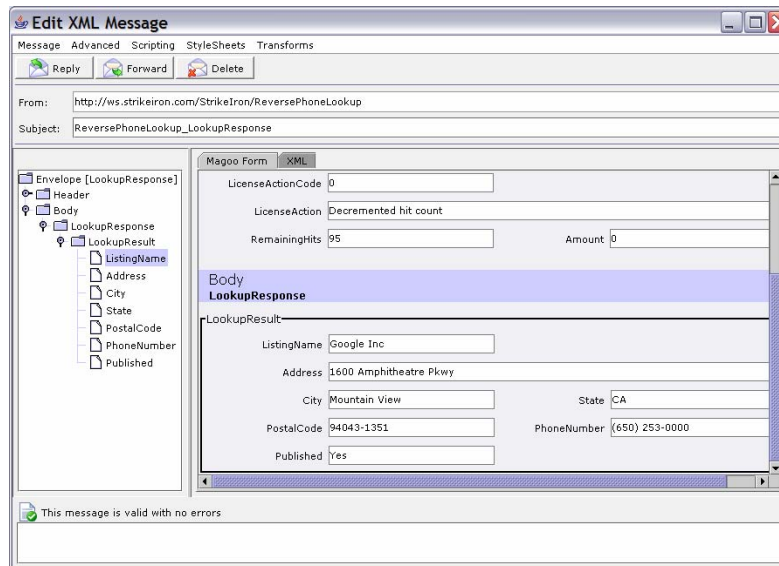


Both the customized stylesheet view and the default Magoosoft view are both based on the same XML model. To view the 'raw' XML, select the XML tab within the Editor pane.



To send a test message, select the Form tab and enter a phone number e.g. (650) 253-0000 into the Number field. Depending on whether or not you have already registered with StrikeIron, enter either your email address and password or your UserId. Click on the **Send** button.

A response from the StrikeIron Service should appear shortly in the MagooClient inbox. Double-click on the message to view it:



It is recommended that you retain this received message as it will be used in the next section to develop the JavaScript required to copy the Lookup Response into the DeliveryService createOrder message.

3.3 Integrating Reverse Phone Lookup

There are 2 steps involved in providing automatic reverse lookup within the createOrder document. First, a script must be created which will:

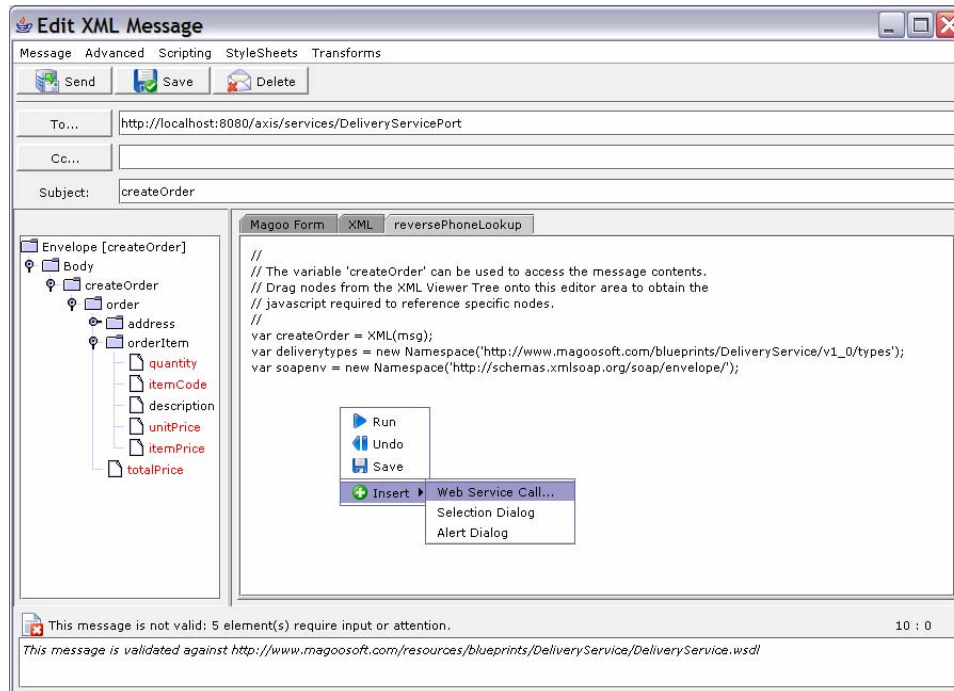
- Create the required SOAP request (ReversePhoneLookup_Lookup)
- Populate this request with the number from the createOrder document
- Dispatch the request to StrikeIron
- Extract the required city, postcode and address details from the response (ReversePhoneLookup_LookupResponse) and populate the createOrder document accordingly.

Secondly a 'ScriptAction' must be configured which will automatically invoke the script whenever the contents of the number field changes.

3.3.1 Creating the ReverseLookup script

Open a new createOrder document by selecting **New** in the main MagooClient window – from the Message Type Catalog select DeliveryService_createOrder. Once the new document instance has been opened for editing, select **Create New...** from the **Scripting** menu.

When prompted enter reversePhoneLookup for the script name. A new script will be created and opened for editing. Right click within the script editor area and select **Insert** and **Web Service Call...**



Select ReversePhoneLookup_Lookup from the Message Type Catalog. MagooClient will automatically insert a JavaScript function `fn_Lookup()`. This function creates a blank SOAP Reverse Lookup request, dispatches it to the StrikeIron Service and sets up a response XML variable to hold the returned SOAP response data.

The first step is to enter your StrikeIron Licensing information into the SOAP content e.g. for an unregistered trial then modify the SOAP header content to include your email address as follows:

```
<tns1:LicenseInfo>
  <tns1:UnregisteredUser>
    <tns1:EmailAddress>my_email@my_company.com</tns1:EmailAddress>
  </tns1:UnregisteredUser>
  <tns1:RegisteredUser>
    <tns1:UserID></tns1:UserID>
    <tns1>Password></tns1>Password>
  </tns1:RegisteredUser>
</tns1:LicenseInfo>
```

If you have a full or registered user trial subscription then add your details to the UserID/Password elements accordingly.

The function now requires only 4 lines of JavaScript in order to integrate it with the createOrder document. Firstly, prior to dispatching the request to the StrikeIron service, set the phone number in the ReversePhoneLookup_Lookup request to the value entered in the createOrder document:

```
Lookup.soapenv::Body.s0::Lookup.s0::Number =
createOrder.soapenv::Body.deliverytypes::createOrder.order.address.contactNumber.toString
();
```

N.B: Note the syntax used to obtain the value – the `toString()` operator must be used on the right hand side of the assignment. This is a common gotcha when attempting to copy element content (as opposed to the elements themselves) from one XML variable to another. Also be sure to use correct namespace prefixing as E4X (JavaScript + XML) is namespace-aware.

Note also that a useful shortcut for filling out element names is to simply drag the required element node from the left-hand navigation tree of an opened message onto the editor area. This will ensure that the full accessor path is always expanded out correctly.

The second set of modifications relate to copying the returned lookup data back into the createOrder document. After the response has been received and the contents stored in the LookupResponse variable, the following 3 lines are used to set the street, city and postcode elements to the values returned from the StrikeIron service. Again note the required use of namespace prefixes and the toString() operator.

```
createOrder.soapenv::Body.deliverytypes::createOrder.order.address.street =
LookupResponse.soapenv::Body.s0::LookupResponse.s0::LookupResult.s0::Address.toString();

createOrder.soapenv::Body.deliverytypes::createOrder.order.address.city =
LookupResponse.soapenv::Body.s0::LookupResponse.s0::LookupResult.s0::City.toString();

createOrder.soapenv::Body.deliverytypes::createOrder.order.address.postalCode =
LookupResponse.soapenv::Body.s0::LookupResponse.s0::LookupResult.s0::PostalCode.toString();
);
```

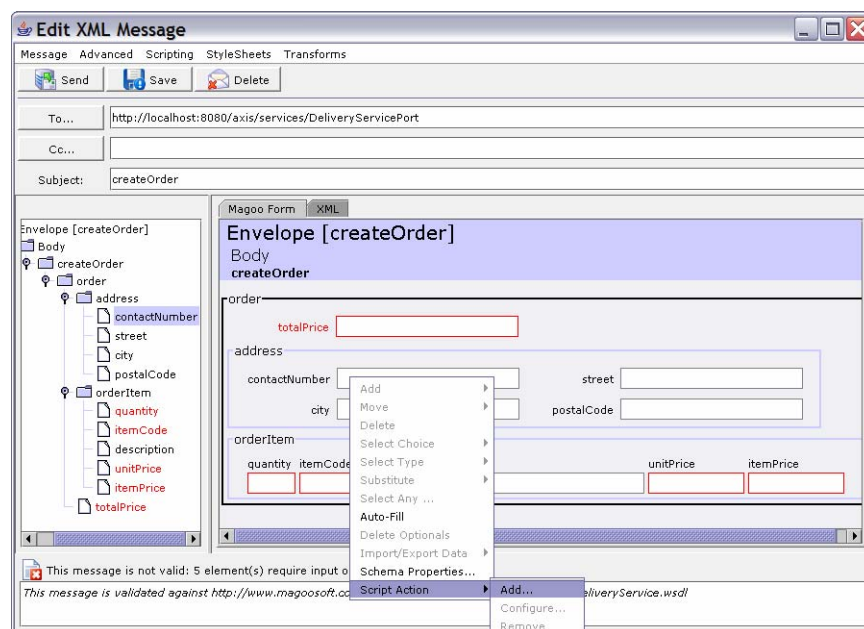
The completed script is provided in [reverseLookup.js](#) with required modifications marked accordingly.

3.3.2 Adding a ScriptAction

There are a number of ways in which scripts can be activated from within MagooClient. These “ScriptActions”, introduced in V3.1, provide a configurable mapping between user interaction and script invocation. The three available ScriptActions are:

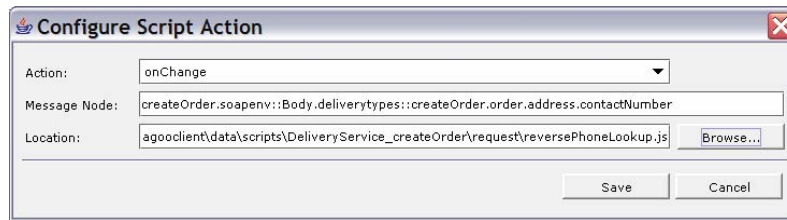
- onOpen – run a script on document opening. This is typically used to mark error conditions within a document for attention on open.
- onPopup – scripts can be activated by right-clicking anywhere within the opened document. By default, all created scripts are configured to be invoked in this way.
- onChange – run a script whenever the configured form field (i.e. document element or attribute) value changes.

In this case, it is required that the reverseLookup script is invoked whenever the value of the contactNumber element changes. To configure a ScriptAction, right click on the element in either the form view or the tree navigation panel.

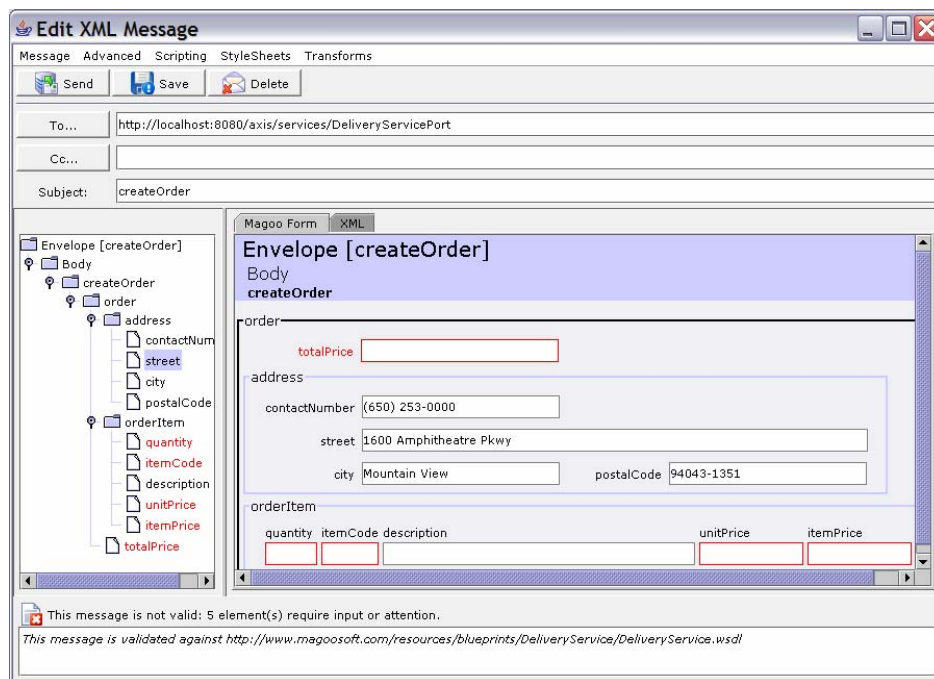


In the dialog that follows, the path for the required node will be automatically populated. Now browse to the required script. Scripts are by default created under:

[MagooClient Install Dir]/data/scripts/<MessageType>/[request|response]



Select Save and test the ScriptAction by entering a sample phone number e.g. (650) 253-0000 into the contactNumber field and hit tab or click outside the field in order to confirm data entry. If script execution has been successful then the street, city and postalCode fields should be populated correctly using the results of the StrikeIron Reverse Lookup call.



That's it! The StrikeIron Reverse Phone Lookup service has been successfully integrated. If you would like additional details on using the MagooClient Scripting Environment, including more advanced user interaction and error-handling capability then please see the [MagooClient User Guide](#).

In the next section, we will see how a second SOA invocation can be used to populate the order item details. This will also include more advanced form-handling, calculations and manipulating document structure.

4 Adding Item Lookup

Note: this section requires that you have built and deployed the `DeliveryService` as described above.

There are a number of requirements relating to the entering of order items which we would like to address using the Scripting Environment:

- The description and unit price fields should be automatically populated on entering an `itemCode`. This should be achieved using a dynamic callout to a Web Service rather than relying on a static mapping within the document.
- For convenience, new blank order items should be added to the form automatically once a order item has been completed
- Unit total prices and overall total price should be automatically calculated.

The process for creating a script, adding a Web Service callout (`DeliveryService_lookupByItemCode`) and configuring a `ScriptAction` are similar to the steps performed above when integrating Reverse Phone Lookup. In this case, an `onChange()` `ScriptAction` should be added to the `itemCode` field under `orderItem` which will automatically invoke the [itemLookup.js](#) script whenever the value of an `itemCode` field changes. This will automatically configure `ScriptActions` for `itemCode` fields under all `orderItem` elements.

In the following section we will focus on some of the more interesting aspects of form-handling.

4.1 Determining the Selected Order Item

The `fn_lookupByItemCode()` function handles invocation on the `DeliveryService` in order to perform a lookup on the item code and return the item description and price. However, the `createOrder` document may have several order items – so the issue therefore is how to determine which item currently has focus as that should be used to provide the `itemCode` for the `lookupByItemCode` request. Likewise, its description and `unitPrice` sub-elements should be populated using the response.

The solution is to use the inbuilt `dialog` object and it's `getSelectionPath()` method. This provides a connection between the scripting environment and the `MagooClient` GUI and returns a string containing the path to the currently selected node. Using some simple javascript, it is then possible to determine the index of the selected `orderItem`.

```
function fn_determineSelectedItemIndex() {
    var path = "" + dialog.getSelectionPath();
    var parts = new Array();
    parts = path.split("\.");
    for (partsIndex = 0; partsIndex < parts.length; partsIndex++) {
        var part = parts[partsIndex];
        if(part.indexOf("orderItem") >= 0) {
            var openBracketIndex = part.indexOf("[");
            if(openBracketIndex < 0) {
                return 0;
            } else {
                return part.substring(openBracketIndex + 1, part.length - 1);
            }
        }
    }
    return -1;
}
```

5 Appendix: Code Listings

Note that all of the following source files are [downloadable as a zip file](#).

5.1 DeliveryService.xsd

```
<!-- (c) Magoo Software Limited 2005-2006 (www.magoosoft.com) -->
<!-- Delivery Service Blueprint -->

<xsd:schema
  targetNamespace="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/xsd"
  xmlns:deliveryxsd="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/xsd"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema">

  <xsd:complexType name="Order">
    <xsd:sequence
      <!-- string types -->
      <xsd:element name="address" type="deliveryxsd:Address"/>
      <xsd:element name="orderItem" type="deliveryxsd:OrderItem" minOccurs="1"
maxOccurs="unbounded"/>
      <xsd:element name="totalPrice" type="xsd:float"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="OrderItem">
    <xsd:sequence
      <!-- string types -->
      <xsd:element name="quantity" type="xsd:int"/>
      <xsd:element name="itemCode" type="deliveryxsd:ItemCode"/>
      <xsd:element name="description" type="deliveryxsd:Description"/>
      <xsd:element name="unitPrice" type="xsd:float"/>
      <xsd:element name="itemPrice" type="xsd:float"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:simpleType name="Description">
    <xsd:restriction base="xsd:string">
      <xsd:maxLength value="30"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:simpleType name="ItemCode">
    <xsd:restriction base="xsd:string">
      <xsd:pattern value="([a-z][A-Z])[0-9]{1,2}"/>
      <xsd:maxLength value="3"/>
    </xsd:restriction>
  </xsd:simpleType>

  <xsd:complexType name="Address">
    <xsd:sequence
      <xsd:element name="contactNumber" type="xsd:string"/>
      <xsd:element name="street" type="xsd:string"/>
      <xsd:element name="city" type="xsd:string"/>
      <xsd:element name="postalCode" type="xsd:string"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="ItemInfo">
    <xsd:sequence
      <xsd:element name="itemCode" type="deliveryxsd:ItemCode"/>
      <xsd:element name="description" type="xsd:string"/>
      <xsd:element name="unitPrice" type="xsd:float"/>
    </xsd:sequence>
  </xsd:complexType>

  <xsd:complexType name="OrderConfirmation">
    <xsd:sequence
      <xsd:element name="reference" type="xsd:string"/>
      <xsd:element name="received" type="xsd:dateTime"/>
      <xsd:element name="deliveryEstimate" type="xsd:dateTime"/>
      <xsd:element name="completedOrder" type="deliveryxsd:Order"/>
    </xsd:sequence>
  </xsd:complexType>
</xsd:schema>
```

5.2 DeliveryService.wsdl

```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions
  name="DeliveryService"
  targetNamespace="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/wsdl">
```

```

xmlns:tns="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/wsd1"
xmlns:deliverytypes="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/types"
xmlns:deliveryxsd="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/xsd"

xmlns:wsd1="http://schemas.xmlsoap.org/wsd1/"
xmlns:soap="http://schemas.xmlsoap.org/wsd1/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
<wsdl:documentation xmlns:wsd1="http://schemas.xmlsoap.org/wsd1/">Delivery Service Demo</wsdl:documentation>
<wsdl:types>
  <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/types">
    <xsd:import namespace="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/xsd"
      schemaLocation="DeliveryService.xsd"/>

    <xsd:element name="createOrder">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="order" type="deliveryxsd:Order"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="createOrderResponse">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="orderConfirmation" type="deliveryxsd:OrderConfirmation"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="lookupByItemCode">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="itemCode" type="deliveryxsd:ItemCode"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>

    <xsd:element name="lookupByItemCodeResponse">
      <xsd:complexType>
        <xsd:sequence>
          <xsd:element name="itemDetails" type="deliveryxsd:ItemInfo"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:element>
  </xsd:schema>
</wsdl:types>
<wsdl:message name="createOrderRequest">
  <wsdl:part element="deliverytypes:createOrder" name="parameters"/>
</wsdl:message>
<wsdl:message name="createOrderResponse">
  <wsdl:part element="deliverytypes:createOrderResponse" name="parameters"/>
</wsdl:message>

  <wsdl:message name="lookupByItemCodeRequest">
    <wsdl:part element="deliverytypes:lookupByItemCode" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="lookupByItemCodeResponse">
    <wsdl:part element="deliverytypes:lookupByItemCodeResponse" name="parameters"/>
  </wsdl:message>

  <wsdl:portType name="DeliveryServicePortType">
    <wsdl:operation name="createOrder">
      <wsdl:input message="tns:createOrderRequest"/>
      <wsdl:output message="tns:createOrderResponse"/>
    </wsdl:operation>
    <wsdl:operation name="lookupByItemCode">
      <wsdl:input message="tns:lookupByItemCodeRequest"/>
      <wsdl:output message="tns:lookupByItemCodeResponse"/>
    </wsdl:operation>
  </wsdl:portType>

  <wsdl:binding name="DeliveryServiceBinding" type="tns:DeliveryServicePortType">
    <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>

    <wsdl:operation name="createOrder">
      <soap:operation soapAction="http://www.magoosoft.com/DeliveryService/createOrder"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>

    <wsdl:operation name="lookupByItemCode">
      <soap:operation soapAction="http://www.magoosoft.com/DeliveryService/lookupByItemCode"/>
      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>

```

```

    </wsdl:binding>

    <wsdl:service name="DeliveryService">
      <wsdl:port binding="tns:DeliveryServiceBinding" name="DeliveryServicePort">
        <soap:address location="http://localhost:8080/axis/services/DeliveryServicePort"/>
      </wsdl:port>
    </wsdl:service>
  </wsdl:definitions>

```

5.3 DeliveryServiceBindingImpl.java

```

/**
 * DeliveryServiceBindingImpl.java
 *
 */

package deliveryservice;

import java.util.*;

public class DeliveryServiceBindingImpl implements deliveryservice.DeliveryServicePortType{

    public deliveryservice.OrderConfirmation createOrder(deliveryservice.Order order) throws java.rmi.RemoteException {

        OrderConfirmation confirmation = new OrderConfirmation();
        confirmation.setCompletedOrder(order);

        confirmation.setReceived(Calendar.getInstance());

        Calendar deliveryTime = Calendar.getInstance();
        deliveryTime.add(Calendar.MINUTE,30);
        confirmation.setDeliveryEstimate(deliveryTime);

        confirmation.setReference("WD-445");
        return confirmation;
    }

    public deliveryservice.ItemInfo lookupByItemCode(String itemCode) throws java.rmi.RemoteException {

        if(itemCode == null || itemCode.length() == 0) {
            return new ItemInfo(itemCode,"ERROR: Undefined code: use T1 - T10",0);
        }

        ItemInfo[] itemInfos = new ItemInfo[] {
            new ItemInfo("T1", "Soup Tom Yam",4.80F),
            new ItemInfo("T2", "Soup Tom Ka",4.80F),
            new ItemInfo("T3", "Thai Spring Rolls",4.50F),
            new ItemInfo("T4", "Ko Tao Butterfly Prawns",5.90F),
            new ItemInfo("T5", "Phi-Phi Haddock",5.30F),
            new ItemInfo("T6", "Tod Mun Pla",4.50F),
            new ItemInfo("T7", "Chicken Satay",4.90F),
            new ItemInfo("T8", "Mango and Papaya Salad",5.50F),
            new ItemInfo("T9", "Kao Pat Vegetable",6.40F),
            new ItemInfo("T10", "Phad Thai Vegetable",6.90F)
        };

        for(int i = 0; i < itemInfos.length; i++) {
            if(itemInfos[i].getItemCode().compareTo(itemCode) == 0) {
                return itemInfos[i];
            }
        }

        return new ItemInfo(itemCode,"ERROR: Undefined code: use T1 - T10",0);
    }
}

```

5.4 build.xml

```

<!-- Generic build file for creating Axis Web Services from WSDL
Created by Magoo Software 2006 (www.magoosoft.com) -->

<project name="DeliveryService" default="all" basedir=".">
  <property name="generated.dir" value="${basedir}/generated"/>
  <property name="build.dir" value="${basedir}/build"/>
  <property name="impl.dir" value="${basedir}/impl"/>
  <property name="classes.dir" value="${basedir}/classes"/>

  <!-- INSTALLATION SPECIFIC PROPERTIES - CHANGE AS REQUIRED -->
  <property name="axis.home" value="c:/axis-1.3"/>
  <property name="tomcat.home" value="c:/Program Files/Apache Software Foundation/Tomcat 5.0"/>
  <property name="axis.deploy" value="{tomcat.home}/webapps/axis/WEB-INF/classes"/>

```

```

<!-- SERVICE/PROJECT SPECIFIC PROPERTIES - CHANGE AS REQUIRED -->
<property name="wsdl.url" value="DeliveryService.wsdl"/>
<property name="package.name" value="deliveryservice"/>
<property name="package.path" value="deliveryservice"/>
<property name="xsd.namespace"
  value="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/xsd"/>
<property name="types.namespace"
  value="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/types"/>
<property name="wsdl.namespace"
  value="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/wsdl"/>

<!-- Set up classpath to pick up the required Axis jars -->
<path id="classpath">
  <fileset dir="${axis.home}/lib">
    <include name="**/*.jar" />
  </fileset>
  <pathelement path="{java.class.path}"/>
</path>

<taskdef name="wsdl2java" classname="org.apache.axis.tools.ant.wsdl.Wsdl2javaAntTask">
  <classpath refid="classpath"/>
</taskdef>

<taskdef name="admin" classname="org.apache.axis.tools.ant.axis.AdminClientTask">
  <classpath refid="classpath"/>
</taskdef>

<target name="all" depends="wsdl2java,merge,compile,upload,deploy"/>

<target name="wsdl2java">
  <echo message="Generating Java from WSDL..." />
  <mkdir dir="${generated.dir}"/>
  <delete quiet="true">
    <fileset dir="${generated.dir}/${package.path}" includes="**/*.java"/>
  </delete>

  <wsdl2java url="${wsdl.url}"
    output="${generated.dir}"
    deployscope="session"
    serverSide="yes"
    skeletonDeploy="no"
    noimports="no"
    verbose="no"
    typeMappingVersion="1.1"
    testcase="no">
    <mapping namespace="${xsd.namespace}" package="${package.name}"/>
    <mapping namespace="${types.namespace}" package="${package.name}"/>
    <mapping namespace="${wsdl.namespace}" package="${package.name}"/>
  </wsdl2java>
</target>

<!-- This target merges any existing implementation files with the generated
  stubs. Note that the ordering of the fileset elements in copy is critical
  to ensure that the implementation files overwrite their generated counterparts -->
<target name="merge" depends="wsdl2java">
  <mkdir dir="${build.dir}"/>
  <mkdir dir="${impl.dir}/${package.path}"/>
  <delete quiet="true">
    <fileset dir="${build.dir}/${package.path}" includes="**/*.java"/>
  </delete>
  <mkdir dir="${impl.dir}"/>
  <echo message="Merging generated and implementation..." />
  <copy todir="${build.dir}/${package.path}">
    <fileset dir="${generated.dir}/${package.path}" excludes="**Impl.java"/>
    <fileset dir="${impl.dir}/${package.path}"/>
  </copy>
</target>

<!-- Compile Service -->
<target name="compile">
  <mkdir dir="${classes.dir}"/>
  <delete quiet="true">
    <fileset dir="${classes.dir}/${package.path}" includes="**/*.class"/>
  </delete>
  <javac srcdir="${build.dir}" destdir="${classes.dir}"
    debug="true">
    <classpath refid="classpath"/>
  </javac>
</target>

<!-- Upload classes to Axis deployment -->
<target name="upload">
  <copy todir="${axis.deploy}/${package.path}">
    <fileset dir="${classes.dir}/${package.path}"/>
  </copy>
</target>

<!-- Deploy to Axis
  N.B: Note that Axis/Tomcat is not hot-deploy. Dont forget to re-start Tomcat to
  ensure the newly-deployed service implementation classes get loaded -->
<target name="deploy">

```

```

        <admin port="8080" hostname="localhost"
            servletpath="/axis/services/AdminService"
            xmlfile="${build.dir}/${package.path}/deploy.wsdd"/>
    </target>
</project>

```

5.5 itemLookup.js

```

//
// The variable 'createOrder' can be used to access the message contents.
// Drag nodes from the XML Viewer Tree onto this editor area to obtain the
// javascript required to reference specific nodes.
//
var createOrder = XML(msg);
var deliverytypes = new Namespace('http://www.magoosoft.com/blueprints/DeliveryService/v1_0/types');
var soapenv = new Namespace('http://schemas.xmlsoap.org/soap/envelope/');

// Invoke Web Service operation lookupByItemCode
fn_lookupByItemCode();

//
// Function to invoke lookupByItemCode
//
function fn_lookupByItemCode() {
    var lookupByItemCode = <?xml version='1.0' encoding='UTF-8'?>
    <soapenv:Envelope
        xmlns:deliverytypes="http://www.magoosoft.com/blueprints/DeliveryService/v1_0/types"
        xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
        <soapenv:Body>
            <deliverytypes:lookupByItemCode>
                <itemCode></itemCode>
            </deliverytypes:lookupByItemCode>
        </soapenv:Body>
    </soapenv:Envelope>

    // Initialise Namespace variables
    var deliverytypes = new Namespace('http://www.magoosoft.com/blueprints/DeliveryService/v1_0/types');
    var soapenv = new Namespace('http://schemas.xmlsoap.org/soap/envelope/');

    // Initialize HTTP Request Object
    var http = new Packages.com.magoosoft.e4x.XMLHttpRequest();
    http.open("http://localhost:8080/axis/services/DeliveryServicePort");
    http.setRequestHeader("SOAPAction", "http://www.magoosoft.com/DeliveryService/lookupByItemCode");

    // Fill out request
    lookupByItemCode.soapenv::Body.deliverytypes::lookupByItemCode.itemCode = fn_determineSelectedItemCode();

    // Send Request
    http.send(lookupByItemCode.toString());

    // Process Response
    var lookupByItemCodeResponse = XML(http.responseText);

    // Determine index to be populated and copy values to form
    var selectedItemIndex = fn_determineSelectedItemIndex();

    createOrder.soapenv::Body.deliverytypes::createOrder.order.orderItem[selectedItemIndex].description =
    lookupByItemCodeResponse.soapenv::Body.deliverytypes::lookupByItemCodeResponse.itemDetails.description;

    createOrder.soapenv::Body.deliverytypes::createOrder.order.orderItem[selectedItemIndex].unitPrice =
    lookupByItemCodeResponse.soapenv::Body.deliverytypes::lookupByItemCodeResponse.itemDetails.unitPrice;

    // Add a new blank item and assign focus
    var orderItemCount = createOrder.soapenv::Body.deliverytypes::createOrder.order.orderItem.length();
    if(selectedItemIndex == orderItemCount - 1) {
        createOrder.soapenv::Body.deliverytypes::createOrder.order.appendChild("<orderItem>" +
            "<quantity>1</quantity>" +
            "<itemCode></itemCode>" +
            "<description/>" +
            "<unitPrice>0.0</unitPrice>" +
            "<itemPrice>0.0</itemPrice>" +
            "</orderItem>");
        selectedItemIndex++;
        dialog.focus("createOrder.soapenv::Body.deliverytypes::createOrder.order.orderItem[" + selectedItemIndex +
            "].quantity");
    }

    fn_calculate();
}

function fn_calculate() {
    var total = 0.0;
    for each(orderItem in createOrder.soapenv::Body.deliverytypes::createOrder.order.orderItem) {

```

```

        orderItem.itemPrice = orderItem.quantity * orderItem.unitPrice;
        total += 1 * orderItem.itemPrice;
    }

createOrder.soapenv::Body.deliverytypes::createOrder.order.totalPrice = total.toFixed(2);
}

function fn_determineSelectedItemCode() {
    var selectedItemIndex = fn_determineSelectedItemIndex();
    if(selectedItemIndex == -1) {
        return "UNDEFINED";
    } else {
        return createOrder.soapenv::Body.deliverytypes::createOrder.order.orderItem[selectedItemIndex].itemCode;
    }
}

function fn_determineSelectedItemIndex() {
    var path = "" + dialog.getSelectionPath();
    var parts = new Array();
    parts = path.split("\\.");
    for (partsIndex = 0; partsIndex < parts.length; partsIndex++) {
        var part = parts[partsIndex];
        if(part.indexOf("orderItem") >= 0) {
            var openBracketIndex = part.indexOf("[");
            if(openBracketIndex < 0) {
                return 0;
            } else {
                return part.substring(openBracketIndex + 1,part.length - 1);
            }
        }
    }
    return -1;
}

```

5.6 reverseLookup.js

```

//
// The variable 'createOrder' can be used to access the message contents.
// Drag nodes from the XML Viewer Tree onto this editor area to obtain the
// javascript required to reference specific nodes.
//
var createOrder = XML(msg);
var deliverytypes = new Namespace('http://www.magoosoft.com/blueprints/DeliveryService/v1_0/types');
var soapenv = new Namespace('http://schemas.xmlsoap.org/soap/envelope/');

// Invoke Web Service operation Lookup
fn_Lookup();

//
// Function to invoke Lookup
//
function fn_Lookup() {
    var Lookup = <?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope
    xmlns:s0="http://www.strikeiron.com" xmlns:tns1="http://ws.strikeiron.com"
    xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <soapenv:Header>
    <tns1:LicenseInfo>
        <tns1:UnregisteredUser>
            <tns1:EmailAddress></tns1:EmailAddress>
        </tns1:UnregisteredUser>
        <tns1:RegisteredUser>
            <tns1:UserID>[Enter your UserID here]</tns1:UserID>
            <tns1:Password></tns1:Password>
        </tns1:RegisteredUser>
    </tns1:LicenseInfo>
    </soapenv:Header>
    <soapenv:Body>
        <s0:Lookup>
            <s0:Number></s0:Number>
        </s0:Lookup>
    </soapenv:Body>
</soapenv:Envelope>

// Initialise Namespace variables
var s0 = new Namespace('http://www.strikeiron.com');
var tns1 = new Namespace('http://ws.strikeiron.com');
var soapenv = new Namespace('http://schemas.xmlsoap.org/soap/envelope/');

Lookup.soapenv::Body.s0::Lookup.s0::Number =
createOrder.soapenv::Body.deliverytypes::createOrder.order.address.contactNumber.toString();

```

```
// Initialize HTTP Request Object
var http = new Packages.com.magoosoft.e4x.XMLHttpRequest();
http.open("http://ws.strikeiron.com/StrikeIron/ReversePhoneLookup");
http.setRequestHeader("SOAPAction", "http://www.strikeiron.com/Lookup");

// Send Request
http.send(Lookup.toString());
var LookupResponse = XML(http.responseText);

createOrder.soapenv::Body.deliverytypes::createOrder.order.address.street =
LookupResponse.soapenv::Body.s0::LookupResponse.s0::LookupResult.s0::Address.toString();

createOrder.soapenv::Body.deliverytypes::createOrder.order.address.city =
LookupResponse.soapenv::Body.s0::LookupResponse.s0::LookupResult.s0::City.toString();

createOrder.soapenv::Body.deliverytypes::createOrder.order.address.postalCode =
LookupResponse.soapenv::Body.s0::LookupResponse.s0::LookupResult.s0::PostalCode.toString();
}
```