Wiseman

Software Version: 1.0

Tutorial

Document Release Date: June 2007

Software Release Date: June 2007

Copyright Notices

Documentation Updates

Table 1 Document Changes

Contributor	Date	Contact	Change Log
Hewlett-Packard	06/2007	dev@wiseman.dev.java.net	Initial creation of the documentation. Primary Author: Joseph Ruzzi Contributing Authors: Simeon Pinder, Nancy Beers, Denis Rachal, and William Reichardt.

Contents

1 Introduction

Wiseman is an open source implementation of the WS-Management specification for the Java SE platform. Wiseman implements the WS-Management specification and its dependent specifications. The Wiseman tutorial provides a step-by-step approach for learning how to use Wiseman. Many of the concepts that are included in the tutorial are explained in detail in the *Server Developer's Guide*.

The tutorial recreates the traffic light server sample that is included with the Wiseman distribution. For demonstration purposes, the sample (client and server) is already built and can be run using the instructions that are provided in the "Getting Started" chapter of the *Server Developer's Guide*.

The traffic light sample uses a simple resource that represents a traffic light. To save time, the actual implementation of the traffic light model is already created and is reused in this tutorial. The traffic light model includes the implementation of the name, color, and location (x and y coordinates) for a traffic light instance.

Prerequisites

Before completing the tutorials, install Wiseman using the instructions located in the *Server Developer's Guide*.

Goals

The goals for the Wiseman tutorial are to:

- Create a working directory
- Represent the traffic light resource using XML schema
- Use the Wiseman tools to generate source files for the traffic light resource
- Use the generated files to implement the WS-Transfer operations
- Use the generated files to implement the WS-Enumeration operations
- Package and deploy the management Web service for the traffic light resource

2 Create a Working Directory

This tutorial creates a working directory that is used throughout the remaining tutorials. A Wiseman project typically begins by creating a working directory and extracting the Wiseman project template to the working directory.

Introduction

A working directory simplifies the Wiseman implementation process. The directory is used to store all the working files for a project and provides a central location for the Wiseman project template. As part of the template, an Ant script is provided to generate classes for a resource and to package a Wiseman implementation for deployment. A common directory structure is also included as part of the template.

Tasks

To create a working directory:

1. Create a directory on your computer. This directory is referred to as `WORK_DIR` throughout these tutorials.
2. Copy the `WISEMAN_HOME\samples\templates\wsdl_war_template.jar` template archive to `WORK_DIR`.
3. From a command prompt, change directories to `WORK_DIR` and issue the following command to extract the template.

   ```
   jar -xf wsdl_war_template.jar
   ```
4. The project template files and directory structure are extracted. The `wsdl_war_template.jar` template archive can be deleted from `WORK_DIR`.

Do not reuse the `WISEMAN_HOME\samples\trafficlight_server` directory as your working directory for this tutorial. If you do, the files in the `\trafficlight_server` directory are overridden during the generation process.

Summary

In this tutorial, a working directory was created for use with the Wiseman tutorials. The steps included extracting the Wiseman project template. The project template includes an Ant script that is used later to generate source files for a resource.

3 Create a Resource

This tutorial copies a traffic light resource model that is provided as part of the Wiseman samples. In addition, this tutorial creates an XML schema file that represents the traffic light resource.

To save time, the XML schema can also be copied from the Wiseman samples.

Introduction

Wiseman uses XML schema to represent a resource that is to be managed. The schema defines the attributes of the resource and assigns a namespace for the resource. The Wiseman generation tools use the schema to generate a WS-Management-compliant WSDL and a JAXB object that is used to marshall/umarsahall XML for the resource. The actual resource model that a schema represents is implementation specific and is typically known before creating a schema.

Tasks

To create the traffic light resource:

1 Copy `WISEMAN_HOME\samples\trafficlight_server\model` to `WORK_DIR`.

2 Using a text editor, create a new file and copy the following schema representing the traffic light resource:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema targetNamespace="http://schemas.wiseman.dev.java.net/
                            traffic/1/light.xsd"
           elementFormDefault="qualified"
           blockDefault="#all"
           xmlns:tl="http://schemas.wiseman.dev.java.net/
                     traffic/1/light.xsd"
           xmlns:xs="http://www.w3.org/2001/XMLSchema">
   <xs:complexType name="TrafficLightType">
      <xs:sequence>
         <xs:element name="name" type="xs:string"/>
         <xs:element name="color" type="xs:string"/>
         <xs:element name="x" type="xs:int"/>
         <xs:element name="y" type="xs:int"/>
      </xs:sequence>
   </xs:complexType>
   <xs:element name="trafficlight" type="tl:TrafficLightType"/>
</xs:schema>
```

3 Save the file as `light.xsd` to the `WORK_DIR\xsd\schemas.wiseman.dev.java.net\traffic\1` directory. This directory needs to be created.

4 Copy the new `light.xsd` to the `WORK_DIR\schemas` directory.

Summary

In this tutorial, the traffic light resource's model was copied to the working directory and the traffic light XML schema was created and saved to the working directory.

4 Generate the Source

In this tutorial, the Wiseman generation tools are used to generate source files based on the traffic light schema. The source files are used to expose the traffic light resource as a WS-Management-compliant Web service.

Introduction

The process of exposing a resource as a WS-Management-compliant Web service can be very time consuming. Wiseman's Xsd2Wsdl and Wsdl2Wsman tools are provided to alleviate some of the more low-level development work and can save developers a considerable amount of time. The template Ant script contains targets that execute the tools based on a given XSD file.

The generated source files include:

- A WS-Management-compliant WSDL
- Class files to implement the WS-Management operations specific to the resource
- JAXB classes for marshalling\unmarshalling XML specific to the resource

Tasks

To generate source files for the traffic light resource:

1. Open the `WORK_DIR\project.properties` file using a text editor.
2. Modify the following properties with the values given:
   - `wiseman.root`: the location of `WISEMAN_HOME`.
   - `war.context.path`: **`/traffic`**
   - `user.wsdl.file`: **`light.wsdl`**
   - `user.xsd.file`: **`light.xsd`**
   - `war.filename`: **`traffic.war`**
   - `resource.uri`: **`urn:resources.wiseman.dev.java.net/traffic/1/light`**
3. Modify the proxy properties if you use a proxy to connect to the internet.
4. Save and close `project.properties`.
5. Open `WORK_DIR/etc/binding.properties` and modify the `com.sun.ws.management.xml.custom.packagenames` property to include the following custom package name:

   `net.java.dev.wiseman.schemas.traffic._1.light`
6. Save and close `binding.properties`.

7 From a command prompt, change directories to `WORK_DIR` and issue the following command:

```
ant generate
```

The source files are generated and placed in `WORK_DIR\gen-src`. In addition, a WS-Management-compliant WSDL for the traffic light resource is created and placed in the `WORK_DIR\wsdls` directory.

Summary

In this tutorial, the Wiseman generation tools were used to generate source files for the traffic light resource. The source files are written to `WORK_DIR\gen-src`.

5 Implement Access Resource

In this tutorial, the WS-Transfer operations (referred to as access resource in the WS-Management specification) are implemented for the traffic light resource. The operations are implemented in the generated traffic light handler class.

Introduction

The WS-Transfer specification defines four operations (`create`, `get`, `put`, and `delete`) that are used to interact with a resource instance. A Wiseman generated WSDL automatically defines these operations and a method for each of these operations is automatically generated in a resource's handler class. Developers must implement these methods using the facilities that a resource provides. If no facilities presently exist to satisfy these methods, they must be created. The traffic light resource that is used in this tutorial provides the necessary facilities to implement these methods.

Tasks

To implement the WS-Transfer operations for the traffic light resource:

1 Using a text or Java editor, open `WORK_DIR\gen-src\net\java\dev\wiseman\resources\traffic\_1\light\LightHandler.java`.

2 Import the following packages and classes:

```
import java.util.HashMap;
import java.util.Set;
import java.util.logging.Level;
import javax.xml.bind.JAXBElement;
import javax.xml.bind.JAXBException;
import javax.xml.namespace.QName;
import javax.xml.soap.SOAPException;
import org.dmtf.schemas.wbem.wsman._1.wsman.SelectorType;
import org.publicworks.light.model.TrafficLightModel;
import org.publicworks.light.model.ui.TrafficLight;
import org.xmlsoap.schemas.ws._2004._08.addressing.EndpointReferenceType;
import net.java.dev.wiseman.schemas.traffic._1.light.ObjectFactory;
import net.java.dev.wiseman.schemas.traffic._1.light.TrafficLightType;
import com.sun.ws.management.InvalidSelectorsFault;
import com.sun.ws.management.UnsupportedFeatureFault;
import com.sun.ws.management.framework.Utilities;
import com.sun.ws.management.transfer.InvalidRepresentationFault;
import com.sun.ws.management.transfer.TransferExtensions;
```

3 Add the following variables:

```
private static final ObjectFactory trafficLightFactory = new ObjectFactory();
private static final QName QNAME = new QName(
   "http://schemas.wiseman.dev.java.net/traffic/1/light.xsd", "trafficlight");
```

4 Implement the `Get()` method as follows:

```
public static void Get(HandlerContext context, Management request,
                       Management response, Logger log) {

   try {

   // Use name selector to find the right light in the model
   String name = getNameSelector(request);
   TrafficLight light = TrafficLightModel.getModel().find(name);

   if (light == null) {
      log.info("An attempt was made to get a resource that did not exist called " +
         name);
      throw new InvalidSelectorsFault(
         InvalidSelectorsFault.Detail.INSUFFICIENT_SELECTORS);
   }

   TransferExtensions xferRequest = new TransferExtensions(request);
   TransferExtensions xferResponse = new TransferExtensions(response);
   xferResponse.setFragmentGetResponse(
      xferRequest.getFragmentHeader(), createLight(light));

   } catch JAXBException e) {
      throw new InternalErrorFault(e);
   } catch (SOAPException e) {
      throw new InternalErrorFault(e);
   }
}
```

5 Implement the `Put()` method as follows:

```
public static void Put(HandlerContext context, Management request,
                       Management response, Logger log) {

   // Use name selector to find the right light
   String name = getNameSelector(request);
   TrafficLight light = TrafficLightModel.getModel().find(name);

   if (light == null) {
      throw new InvalidSelectorsFault(
         InvalidSelectorsFault.Detail.INSUFFICIENT_SELECTORS);
   }

   // Get the resource passed in the body
   Object obj = getResource(request);

   if ((obj instanceof JAXBElement) == false) {
      throw new InternalErrorFault("Wrong resource type \n");
   }

   JAXBElement elem = (JAXBElement) obj;
   JAXBElement<TrafficLightType> tlElement = (JAXBElement<TrafficLightType>) obj;
   TrafficLightType tlType = tlElement.getValue();

   // Transfer values
   light.setName(tlType.getName());
   light.setColor(tlType.getColor());
   light.setX(tlType.getX());
   light.setY(tlType.getY());

   try {
      TransferExtensions xferResponse = new TransferExtensions(response);
      xferResponse.setPutResponse();
   } catch (SOAPException e) {
      throw new InternalErrorFault(e);
   }
}
```

6 Implement the `Delete()` method as follows:

```
public static void Delete(HandlerContext context, Management request,
                          Management response, Logger log) {

   // Use name selector to find the right light
   String name = getNameSelector(request);
   TrafficLight light = TrafficLightModel.getModel().find(name);

   if (light == null) {
      log.log(Level.WARNING, "An attempt was made to delete a resource that
         did not exist called " + name);
      throw new InvalidSelectorsFault(
         InvalidSelectorsFault.Detail.INSUFFICIENT_SELECTORS);
   }

   // Remove it from the list and then remove the actual GUI instance.
   TrafficLightModel.getModel().destroy(name);
   try {
      TransferExtensions xferResponse = new TransferExtensions(response);
      xferResponse.setDeleteResponse();
   } catch (SOAPException e) {
      throw new InternalErrorFault(e);
   }
}
```

7 Implement the `Create()` method as follows:

```
public static void Create(HandlerContext context, Management request,
                          Management response, Logger log) {
   try {
         // Get the resource passed in the body
         TransferExtensions xferRequest = new TransferExtensions(request);
         TransferExtensions xferResponse = new TransferExtensions(response);
         Object element = xferRequest.getResource(QNAME);

         TrafficLight light;

         if (element != null) {
            JAXBElement<TrafficLightType> tlElement = getResource(request);
            TrafficLightType tlType = tlElement.getValue();

            // Create and store a reference to this object in our mini-model
            light = TrafficLightModel.getModel().create(tlType.getName());

            // Transfer values
            light.setColor(tlType.getColor());
            light.setX(tlType.getX());
            light.setY(tlType.getY());

         } else {

            // Create and store a reference to this object in our mini-model
            light = TrafficLightModel.getModel().create(null);
            log.log(
               Level.INFO, "The body of your request is empty but it is optional.");
         }

         // Define a selector (in this case name)
```

```
            HashMap<String, String> selectors = new HashMap<String, String>();
            selectors.put("name", light.getName());

            EndpointReferenceType epr = xferResponse.createEndpointReference(
               request.getTo(), request.getResourceURI(), selectors);
            xferResponse.setCreateResponse(epr);

      } catch (SOAPException e) {
         throw new InternalErrorFault(e);
      } catch (JAXBException e) {
         throw new InternalErrorFault(e);
      }
   }
```

8 Implement the following JAXB utility methods that are used by the transfer operation methods as follows:

```
private static String getNameSelector(Management request) throws InternalErrorFault {
   Set<SelectorType> selectors;

   try {
      selectors = request.getSelectors();
   } catch (JAXBException e) {
   throw new InternalErrorFault(e);
   } catch (SOAPException e) {
   throw new InternalErrorFault(e);
   }

   if (Utilities.getSelectorByName("name", selectors) == null) {
      throw new InvalidSelectorsFault(
         InvalidSelectorsFault.Detail.INSUFFICIENT_SELECTORS);
   }
   return (String) Utilities.getSelectorByName("name", selectors).getContent().get(0);
}


private static JAXBElement<TrafficLightType> getResource(Management request) {
   JAXBElement<TrafficLightType> tlElement;

   try {

      // Get JAXB Representation of Soap Body property document
      TransferExtensions transfer = new TransferExtensions(request);
      Object element = transfer.getResource(QNAME);

      if (element == null) {
         throw new InvalidRepresentationFault(
            InvalidRepresentationFault.Detail.MISSING_VALUES);
      }

      if (element instanceof JAXBElement) {
         if (((JAXBElement) element).getDeclaredType()
            .equals(TrafficLightType.class)){
            tlElement = (JAXBElement<TrafficLightType>) element;
         } else {

            // XmlFragment only supported on Get
            throw new UnsupportedFeatureFault(
               UnsupportedFeatureFault.Detail.FRAGMENT_LEVEL_ACCESS);
         }
      } else {
         throw new InvalidRepresentationFault(
                InvalidRepresentationFault.Detail.INVALID_VALUES);
            }
        } catch (SOAPException e) {
            throw new InternalErrorFault(e);
        } catch (JAXBException e) {
            throw new InternalErrorFault(e);
        }
        return tlElement;
    }


    private static TrafficLightType createLightType(TrafficLight light) {

        // Create a new, empty JAXB TrafficLight Type
        TrafficLightType tlType = trafficLightFactory.createTrafficLightType();

        // Transfer State from model to JAXB Type
        tlType.setName(light.getName());
        tlType.setColor(light.getColor());
        tlType.setX(light.getX());
        tlType.setY(light.getY());
        return tlType;
    }


    public static JAXBElement<TrafficLightType> createLight(TrafficLight light)
        throws JAXBException {
        TrafficLightType lightType = createLightType(light);
        return trafficLightFactory.createTrafficlight(lightType);
    }
```

9 Save and close the `LightHandler.java` file.

Summary

In this tutorial, the WS-Transfer operations (`create`, `get`, `put`, `delete`) were implemented for the traffic light resource. These operations were implemented in the generated `LightHandler.java` file which contains methods for each of the operations. The methods used facilities included with the traffic light resource model.

6 Implement Enumeration

In this tutorial, WS-Enumeration is implemented for the traffic light resource. In particular, the generated traffic light iterator factory class is used to create new iterator instances specific to the traffic light resource.

Introduction

The WS-Enumeration specification defines three operations (`Enumerate`, `Pull`, and `Release`) that are used to enumerate over many instances of a resource. A Wiseman generated WSDL automatically defines these operations and a method for each of these operations is automatically generated in a resource's handler class.

Unlike the WS-Transfer methods, the enumeration methods that are generated in the resource's handler class do not need to be implemented. Instead, theses methods delegate enumeration requests to a Wiseman support class, `EnumerationSupport`, through the resource handler's extended class (either `EnumerationHandler` or `ResourceHandler`).

The Wiseman support class, among other things, uses a resource's generated iterator factory class to create a new iterator instance. Developers must implement four methods that are automatically added to the generated iterator factory class (`estimateTotalItems`, `hasNext`, `next`, and `release`). A resource must provide the facilities to implement these methods. If no facilities presently exist to satisfy these methods, they must be created. The traffic light resource that is used in this tutorial provides the necessary facilities to implement these methods.

Tasks

To implement WS-Enumeration for the traffic light resource:

1. Using a text or Java editor, open `WORK_DIR\gen-src\net\java\dev\wiseman\resources\traffic\_1\light\LightIteratorFactory.java`.
2. Import the following packages and classes:

```
import java.util.ArrayList;
import java.util.Collection;
import javax.xml.bind.JAXBException;
import org.publicworks.light.model.TrafficLightModel;
import org.publicworks.light.model.ui.TrafficLight;
import com.sun.ws.management.framework.transfer.TransferSupport;
```

3. Add the following variables:

```
private static final String WSMAN_TRAFFIC_RESOURCE =
   urn:resources.wiseman.dev.java.net/traffic/1/light";
private static final Logger log = Logger.getLogger(LightIteratorImpl.class.getName());
private int length;
private final String address;
private final boolean includeEPR;
private Iterator<TrafficLight> lights;
```

4 Implement the `next()` method as follows:

```
@SuppressWarnings("unchecked")
public EnumerationItem next() {

   // Get the next light
   TrafficLight light = lights.next();
   Map<String, String> selectors = new HashMap<String, String>();
   selectors.put("name", light.getName());
   try {
      final EndpointReferenceType epr;

      if (includeEPR == true) {
         epr = TransferSupport.createEpr(address, WSMAN_TRAFFIC_RESOURCE, selectors);

      } else {
         epr = null;
      }

      // Create the EnumerationItem and return it
      JAXBElement<TrafficLightType> lightElement =
         LightHandlerImpl.createLight(light);
      return new EnumerationItem(lightElement, epr);
   } catch (JAXBException e) {
      throw new InternalErrorFault(e.getMessage());
   }
}
```

5 Implement the `estimateTotalItems()` method as follows:

```
public int estimateTotalItems() {
   return length;
}
```

6 Implement the `hasNext()` method as follows:

```
public boolean hasNext() {
   return lights.hasNext();
}
```

7 Implement the `release()` method as follows:

```
public void release() {
   length = 0;
   lights = new ArrayList<TrafficLight>().iterator();
}
```

8 Save and close the `LightIteratorFactory.java` file.

Summary

In this tutorial, WS-Enumeration was implemented for the traffic light resource. The generated `LightIteratorFactory.java` file was used to implement iteration methods (`estimateTotalItems`, `hasNext`, `next`, and `release`) that are used to iterate over many traffic light instances.

7 Deploy the Management Web Service

In this tutorial, the Wiseman project template Ant script is used to compile and build the traffic light sample Web application. The Web application is then deployed to a Java EE Web container.

Introduction

The Wiseman project template Ant script contains targets for compiling service implementation classes that are in a project's `WORK_DIR` and package the files as a Web archive. The Web archive contains the Wiseman Servlet, all dependent libraries, all required schemas, and all required WSDLs.

Tasks

To deploy the traffic light Web service:

1. From a command prompt, change directories to `WORK_DIR`.
2. Issue the following command:

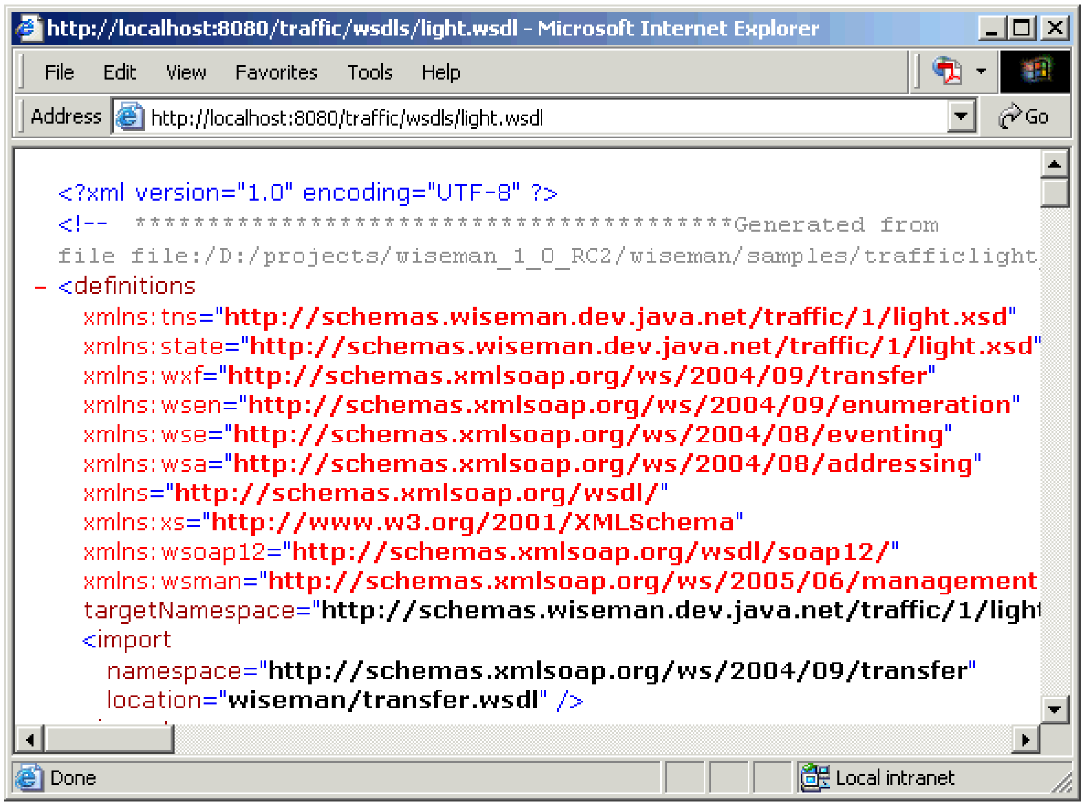
   ```
   ant
   ```

   The service implementation classes are compiled and the `traffic.war` Web application is created and written to the `WORK_DIR/dist` directory.
3. Deploy the `traffic.war` to a Java EE Web container.
4. Using a Web browser, verify that the deployment succeeded by accessing the WS-Management WSDL for the traffic light resource.

   ```
   http://<host>:<port>/traffic/wsdls/light.wsdl
   ```

   Replace `<host>`:`<port>` with the host name and port number for the Web container where the WAR was deployed. The WSDL for the sample is returned as shown below:

The `web.xml` file defines security constraints for the `wsman` role. Either modify your application server's roles to include this role, or modify the `web.xml` with a valid role already defined on the application server.


http://localhost:8080/traffic/wsdls/light.wsdl - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:8080/traffic/wsdls/light.wsdl
Go
Done
Local intranet

```
<?xml version="1.0" encoding="UTF-8" ?>
<!-- ******************************************Generated from
file file:/D:/projects/wiseman_1_0_RC2/wiseman/samples/trafficlight
- <definitions
    xmlns:tns="http://schemas.wiseman.dev.java.net/traffic/1/light.xsd"
    xmlns:state="http://schemas.wiseman.dev.java.net/traffic/1/light.xsd"
    xmlns:wxf="http://schemas.xmlsoap.org/ws/2004/09/transfer"
    xmlns:wsen="http://schemas.xmlsoap.org/ws/2004/09/enumeration"
    xmlns:wse="http://schemas.xmlsoap.org/ws/2004/08/eventing"
    xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing"
    xmlns="http://schemas.xmlsoap.org/wsdl/"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:wsoap12="http://schemas.xmlsoap.org/wsdl/soap12/"
    xmlns:wsman="http://schemas.xmlsoap.org/ws/2005/06/management
    targetNamespace="http://schemas.wiseman.dev.java.net/traffic/1/ligh
    <import
      namespace="http://schemas.xmlsoap.org/ws/2004/09/transfer"
      location="wiseman/transfer.wsdl" />
```

Summary

In this tutorial, the traffic light sample was compiled and packaged as a Web application. The application was then deployed to a Java EE Web container. Lastly, the deployment was verified by retrieving the sample `light.wsdl` from the Web application.