Eclipse Web Tools Platform – Uncovered:
Web Application Concepts and Development Quick Start

Naci Dai, WTP PMC
eteration a.s.

Web Applications & JEE Development – Naci Dai
JavaServer Faces (JSF) - Raghu Srinivasan
Java Persistence with Dali – Neil Hauge & Tom Ware
Web Services – Kathy Chan
Agenda

PART I

- WTP Quick Tour
  - Setting up Your Workspace
  - The Presentation Tier

PART II

- Building The Service Layer
  - Business and Persistence Tier
  - Web Services
Eclipse Web Tools Platform

Developing Java Web Applications

By:
- Naci Dai
- Lawrence Mandel
- Arthur Ryman

http://www.eclipsewtp.org
WTP Quick Tour Overview

An overview of the many WTP tools

- **Iteration 1**
  - Work with an application server,
  - Create a Web application
  - Develop a simple JavaServer™ Pages (JSP) document

- **Iteration 2**
  - Web MVC with Java Server Faces (JSF)
  - Debugging

- **Iteration 3**
  - Data Tools & Java Persistence API (JPA)

- **Iteration 4**
  - Web Services
  - SOAP Monitor
Iteration 1: Web Applications

• Configure a development environment
• Tasks:
  1. Configure an application server
  2. Create a Web application project
  3. Develop a simple JavaServer Pages (JSP) document that prints a greeting
  4. Run the JSP on the server
Configure an Application Server

An application server is needed to run our Java Web application

- Add a runtime
  - Window->Preferences
    - Server
      - Installed Runtimes

- Click Add.
  - Locate Tomcat
  - Specify JDK so your JSPs will compile.
Create a Dynamic Web Project

The project will contain all of our Web artifacts such as JSPs, servlets

• Create Project
  ▪ File->New->Project
  ▪ Web
    – Dynamic Web Project wizard.
    – Name the project (LeaguePlanetWeb)
    – Facets (Servlet 2.4, Java 5)

• Target Runtime
  ▪ Choose Tomcat

• Click Finish.
Dynamic Web Projects

- Dynamic Web projects are:
  - JEE Web modules
  - Combine Java and Web Development
  - Support Servers
Web Project Facets

- Project facet describes runtime aspects of the Web module
  - Choose defaults
    - add JSF 1.1
  - Click Next
JSF Libraries

- Configuring the Web Application for JSF
  - WTP allows you to choose/define JSF libraries
  - Configures the application
Develop a Simple JSF Application

The JSP will display “Hello, World” in a client’s Web browser

• In WebContent folder
  ▪ New->JSP
  ▪ Name the JSP hello-world.jsp.
  ▪ Select JSP with html markup.
  ▪ Click Finish.

• Open with Web Page Editor
  ▪ Change the title and add body contents for “Hello, world”
Run the JSP on the Server

To make use of the JSP it must be run on a server, in our case Tomcat

- **hello-world.jsp**
  - select Run As->Run on Server
  - Tomcat starts up and displays the JSP.

```
Hello, world.
```
Iteration 1 Summary

We have:

- Configured Tomcat to act as our application server
- Created a Web application project
- Developed a simple JSP that prints “Hello, world” in a browser
- Ran the JSP on the Tomcat server
Iteration 2: Web MVC with JSF

- Create a dynamic Web application with Java Server Faces
- Tasks:
  1. Create a new Managed Bean with an action
  2. Add a page with a login form
  3. Debug the application
Simple JSF Application

- Login

```
\begin{itemize}
  \item Login Bean
  \item User
\end{itemize}
```
Steps

1. Setup JSF application in faces-config
   - A) Register LoginBean as a managed bean
   - B) Define page flow as navigation rules
2. Create a input form (login.jsp) using JSF tags
3. Link button action to LoginBean method
4. Store user in session
5. Forward the application to the result page
6. Display page shows user info
Step 2: Faces Configuration

- Use the faces-config editor
  - double click on faces-config.xml
  - Managed bean tab
The faces-config.xml

```
<faces-config>

    <!-- Managed Beans -->
    <managed-bean>
        <managed-bean-name>loginBean</managed-bean-name>
        <managed-bean-class>
            org.leagueplanet.beans.LoginBean
        </managed-bean-class>
        <managed-bean-scope>request</managed-bean-scope>
    </managed-bean>
    ...

</faces-config>
```
Managed Beans (POJO)

- Plain Old Java Objects

```java
public class LoginBean {
    private User user;

    public String login() {
        // Some Code here
        ....
        return "success";
    }
}
```

```java
public class User {
    private String userId;
    private String fullname;
    ...
```
Input Form Using JSF Tags

- Login Input Form (login.jsp)

```
<f:view>
    <h:form>
        <h:outputText value="User:" />
        <h:inputText value="#{loginBean.user.userId}" />
        <h:commandButton value="Login" action="#{loginBean.login}"/>
    </h:form>
</f:view>
```

The login JSP will submit information to a servlet using a form

note: missing some html code here
Final Page (login_success.jsp)

Result page

```html
<%@ taglib uri="http://java.sun.com/jsf/core" prefix="f"%>
<%@ taglib uri="http://java.sun.com/jsf/html" prefix="h"%>
<f:view>
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
    "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head><title>Successful Login</title></head>
<body>
<h:form>
<p>User <h:outputText value="#{loginBean.user.userId}"/> has been successfully logged in.</p>
</h:form>
</body>
</html>
</f:view>
```
Add Page Navigation

- Use faces-config editor
  - Add navigation from login.jsp to login_success.jsp for outcome “success”

```xml
<faces-config>
...

<!-- Navigation Rules -->
...

<navigation-rule>
  <from-view-id>/login.jsp</from-view-id>
  <navigation-case>
    <from-outcome>success</from-outcome>
    <to-view-id>/login_success.jsp</to-view-id>
  </navigation-case>
</navigation-rule>
...

</faces-config>
```
Debug a Web Application

- Set a breakpoint in LoginBean on the line `public String login()`
- Login.jsp
  - Debug As->Debug on Server.
  - Select to resume execution.
  - Try changing the name and watch the execution path.

Debugging a servlet is very similar to debugging a Java class
Iteration 2 Summary

• We have:
  ▪ Setup JSF
  ▪ Understand the basic elements of faces configuration
  ▪ Build and run a simple JSF application
  ▪ Added Java to a JSP
  ▪ Created a login JSP and a Bean to handle login requests
  ▪ Debugged a JSP/Bean on the server
Iteration 3: Database Access

• Add a data layer to our Web application

• Tasks:
  1. Connect to a Database
  2. Execute SQL Statements
  3. Add Database Access to our Web Application
Show the Database Views

DTP provides a perspective that assist in working with databases. We need to show it before proceeding.

- Open Perspective
  - Database Development
Connect to a Database

Before working with a database we must first connect to it.

• Data Source Explorer
  ▪ Database New…
    – Select a driver (Derby 10.1.)
    – Enter the location of derby.jar. It’s located in the apache derby
    – Enter a convenient location for the database such as C:\DerbyDatabases\LeagueDB
    – Click Test Connection.
    – If the connection test was successful, click Finish.
Populate our database by executing SQL statements.

Open the SQL Scrapbook

- File New > New SQL File - selecting the DB, and enter a name of leaguesql.sqlpage.

```
CREATE TABLE WEB1.LOGIN
    (USERID CHAR(8) NOT NULL, FULLNAME CHAR(20), PRIMARY KEY(USERID))
INSERT INTO WEB1.LOGIN (USERID, FULLNAME) VALUES ('dai','Naci Dai')
INSERT INTO WEB1.LOGIN (USERID, FULLNAME) VALUES ('mandel','Lawrence Mandel')
INSERT INTO WEB1.LOGIN (USERID, FULLNAME) VALUES ('ryman','Arthur Ryman')
SELECT * FROM WEB1.LOGIN ORDER BY FULLNAME
SELECT FULLNAME FROM WEB1.LOGIN WHERE USERID = 'ryman'
```

Select Run SQL.

- Results are shown in the Data Output view.
Disconnect from the Database

- Database Explorer
  - Right click on the DB and select Disconnect.

Derby only supports a connection to a given database from one process
Database Access with JPA

- Modify LeaguePlanetWeb Project Properties
  - Add JPA 1.0 Facet
  - Configure JPA Library
JPA Development Perspective

- Open the JPA Development Perspective to access JPA Tools
  - Data Source Explorer
  - JPA Details
Map User Class to User Table

- Map User Class
  - Entity
  - Choose schema
    - WEB1
  - Choose Table
    - LOGIN
  - Choose primary key (ID)
    - userId

```java
@Entity
@Table(schema="WEB1", name = "LOGIN")
public class User {

@Id
private String userId;

private String fullname;
```
DB Access from a Web App

• Update our Web application
  – to retrieve user names from the database

• Tasks:
  1. Add the Derby library to our Web application
  2. Add connection parameters to persistence.xml
  3. Update LoginBean to use JPA to retrieve the user from DB
Our Web App needs access to the Derby libraries in order to access a Derby database

- Copy JAR files to a Web Application
- WebContent
  - WEB-INF/lib
    - derby.jar

*Derby Note: While this method works fine for a single application, if multiple applications need access to a Derby database a shared copy of Derby must be used.*
Manage the persistence.xml file

- To synchronize the file with mapped classes
  - Right-click the persistence.xml
  - JPA > Synchronize Classes

- To add database-specific information
  - Add the following after after the final <class>

```xml
<persistence-unit name="default">
  <mapping-file>META-INF/orm.xml</mapping-file>
  <class>org.leagueplanet.beans.User</class>
  <properties>
  <property name="eclipselink.logging.level" value="FINEST" />
  <property name="eclipselink.jdbc.driver" value="org.apache.derby.jdbc.EmbeddedDriver" />
  <property name="eclipselink.jdbc.url" value="jdbc:derby:C:\DerbyDatabases\LeagueDB;create=true" />
  <property name="eclipselink.jdbc.user" value="test" />
  <property name="eclipselink.jdbc.password" value="test" />
  </properties>
</persistence-unit>
```
Update LoginBean

The class will contain all the logic to access the database

```java
public class LoginBean {
    protected EntityManagerFactory emf;
    public String login() {
        EntityManager em = getEMF().createEntityManager();
        try{
            Query query = em.createQuery("SELECT u FROM User u WHERE u.userId = :uid ");
            query.setParameter("uid", user.getUserId());
            user = (User)query.getSingleResult();
        }finally{
            em.close();
        }
        return "success";
    }

    public EntityManagerFactory getEMF() {
        if (emf == null) {
            emf = Persistence.createEntityManagerFactory("default",
                    new java.util.HashMap());
        }
        return emf;
    }
}
```
Iteration 3 Summary

- We have:
  - Connected to a Derby database
  - Executed SQL statements to populate the database and view the values we added
  - Mapped a class using JPA
  - Used JPA to retrieve Users
Iteration 4: Web Services

- Expose a Java Component through a Web service
- Tasks:
  1. Deploy a Web service
  2. Test a Web service with a test client
  3. Monitor SOAP messages
Deploy a Web Service

Create Web service using the bottom-up approach

- LoginBean
  - Add a new method lookupFullName(id)
- Web Services
  - Create Web service.
- Wizard:
  1. Start Web service in Web project
  2. Generate a Proxy
  3. Test the Web service
  4. Monitor the Web service
- Click Finish.
Web Service Test Client

- The test client allows you to easily test a Web service.
- The test client was created by selecting the option in the WS wizard

- lookupFullname method.
  - Enter the name
  - Click Invoke.
Monitor SOAP Messages

- Web services send SOAP messages between the client and server.
- We can monitor these messages to see the traffic and find problems.

- TCP/IP monitor view.
  - Window->Show View->Other…
  - Debug->TCP/IP Monitor

![TCP/IP Monitor View Image]
TCP/IP Monitor

- WTP contains a very useful tool called the TCP/IP Monitor that lets you peek into the HTTP traffic and see what's going on.
Monitoring Ports

- Servers view, select the Tomcat server, right click
  - Monitoring->Properties
Add a TCP/IP monitor to Tomcat.

- Click Add
- Monitor HTTP Port 8080
  - Add monitor to port 8081
  - Click start
Iteration 4 Summary

- We have:
  - Deployed a Bottom-up Web service
  - Tested the Web service
  - Monitored SOAP messages
Quick Tour Summary

- WTP has many tools:
  - Development of Web applications
  - Databases
  - Web services

- Other WTP tools include:
  - CSS, DTD, EAR, EJB, HTML, XHTML, J2EE, JavaBeans™, JavaScript™, JSP, Servlet, SQL, WSDL, XML, XML Schema, WAR, Web services
  - JSF, JPA, AJAX
Agenda

PART I
- WTP Quick Tour
- Setting up Your Workspace
- The Presentation Tier

PART II
- Building The Service Layer
  - Business and Persistence Tier
  - Web Services
Workspace Setup Overview

- Installation of Server Runtimes
- Workspace Settings
- Understanding Web Projects
Server Runtimes

- WTP does not include server runtimes
  - Runtimes must be downloaded separately
  - Adapters are included for some runtimes
    - Apache Tomcat, IBM® WebSphere®, JBoss, ObjectWeb JOnAS
    - Others can be installed on demand
      - BEA WebLogic, Oracle, Apache Geronimo
Installation of a Runtime

- Example: Apache Tomcat
  - Servlet container that can only run Java Web applications

- To add:
  - Window->Preferences → Server->Installed Runtimes
  - Click Add
  - Select the correct version of Tomcat
  - Select the installation dir and a JDK.
  - Click Finish.
JRE vs. JDK

- Application servers require a JDK
  - JRE does not have tools for compiling JSP, Servlet, etc.

- To Install a JDK
  - Window->Preferences
  - Java->Installed JREs
Workspace Settings

- WTP contains settings for various Web development tools and artifacts
  - Data
  - Internet
  - Java
  - Run/Debug
  - Server
  - Validation
  - Web/XML
  - Web Services
  - XDoclet
Sharing Settings

- Project settings can be shared by a team
  - Share .settings in CVS

- Global workspace settings can be exported
  - Export->Preferences
WTP Project Types

- Projects
  - Static Web
  - Dynamic Web (JEE Web)
  - EJB
  - EAR
  - J2CA
  - Utility Modules
  - Other …
Flexible Web Projects

- Directory layout can be changed

J2EE Standard

Workspace
Project Facets

- A project facet describes a runtime aspect of the module
  - The JEE version,
  - The Java version
  - Runtime support
  - Builders
Dependent Modules

- Other projects, or external libraries can be linked via module dependencies

- Project Properties
  - modify
Workspace Setup Summary

• We have seen how to:
  – Install a server runtime
  – Change and share workspace settings
  – Create and link WTP project
Agenda

PART I
- WTP Quick Tour
- Setting up Your Workspace
- The Presentation Tier

PART II
- The Business Tier
- The Persistence Tier
- Web Services
Presentation Tier

- Presentation Tier separates the user interface of a system from its core.

- Constructing the presentation-tier
  - HTTP, HTML, CSS, JavaScript™, XML, XSLT, Servlets, JSP, and JSF
JEE and the Presentation Tier

- **JEE Multi Tier Distributed Application**
  - **client tier** - the end-user device
  - **middle tier** - Web and EJB containers
  - **Enterprise Information System (EIS) tier** – databases and other business systems
Business Tier Design

- Business Tier has two distinct sets of models
  - Domain Logic
  - Application Logic

- Interfaces to other tiers
  - Service Layer
  - Data Layer

WTP tools support development for all tiers
Agenda

- **PART I**
  - WTP Quick Tour
  - Setting up Your Workspace
  - The Presentation Tier

- **PART II**
  - Building The Service Layer
    - Business and Persistence Tier
    - Web Services
Overview

- Business and Persistence-Tier
  - Use a Utility Project
  - Domain Logic with POJOs
  - Testability without a server runtime
  - Persistence with JPA
  - Enterprise JavaBeans with WTP
- Service Layer
Utility Project

- A Java project with:
  - Awareness of WTP modules
  - Module dependencies to link
    - Utility projects in other modules
    - e.g. Use the model in an EJB module

- File > New >
  - J2EE > Utility Project
Domain Model

**POJO**
- Plain-old-java-objects
Testing the model

- Utility project are Java projects
  - No need to “Run on server” to test

- Can create a JUnit Test
  - Run independently
Service layer

- Service Interfaces
- Capture Application Logic
  - Flow of events
  - Business processes and workflows
Service Interface

```java
package org.leagueplanet.services;

import org.leagueplanet.model.*;

public interface LeagueFacade {
    public List<Schedule> getSchedulesForLeague(String league);
}
```
Using EJBs to Implement Services

- **Component architecture**
  - Coarse-grain business-level functions with clear interfaces,
  - Easily Distributed over a network.

- **Enterprise Java Beans**
  - EJB3 is a complete revision of the EJB specification
    - Implementation and maintenance of EJBs are simpler
  - Support for Web Services with JAX-WS
Data Layer

- Ties the business tier with the persistence tier
  - Business tier must be independent of the persistence tier
  - Support multiple datastores and mapping tools
package org.leagueplanet.dao;

import java.util.Set;

import org.leagueplanet.model.*;

public interface LeagueRepository {
    public List<Schedule> getSchedulesForLeague(String league);
}
Enterprise Application with EJBs

Enterprise applications are used to assemble components that are needed to make the whole work

- Create an enterprise application
  - Utility Project (Model)
  - EJB Project (Components)
Creating an Enterprise Application

- This ear application will contain the model and an EJB that will provide the Remote services
- Tasks
  1. Define a new Server Runtime (EJB 3 Support)
  2. Create an EAR application project
  3. Add a new EJB3 module to EAR (LeaguePlanetEJB)
  4. Add model Utility project (LeaguePlanetModel)
  5. Link EJB project with the Model
  6. Implement the EJB
Adding Server Runtime

- WTP uses Server Runtimes to run EJBs
  - Server runtime must support EJB3
    - OSS
      - ObjectWeb JOnAS, JBoss 4.2, Apache Geronimo, ...
    - Commercial
      - IBM® WebSphere®, BEA WebLogic, Oracle AS, ...
EAR Project

File>New>Project
• J2EE > Enterprise Application Project
Linking EJB with the Model

- EJB Project will use the classes from the Model project

- Open the LeaguePlanetEJB project properties
  - Goto J2EE Module Dependencies
    - Select LeaguePlanetModel
EJB3 Web Service Interface

- We will need an interface that defines our service endpoint
  - Contains the methods that are published by the Web service
  - Inherits the methods from the model
  - Uses JEE5 JAX-WS Annotations

- Add the following interface to EJB Project:

```java
@WebService
@SOAPBinding(style = Style.DOCUMENT, use= Use.LITERAL)

public interface LeagueFacadeRemote
  extends LeagueFacade, Remote {

}
```
Web Service Implementation Class

• Add an EJB3 Stateless Session Bean
  – Create a new Java Class that implement our service layer interfaces
  – Uses JEE5 EJB3 Annotations

• Add the following Java class to EJB Project:

```java
@Stateless
@WebService(endpointInterface = "org.leagueplanet.ejb.LeagueFacadeRemote")
@Remote(LeagueFacadeRemote.class)

public class LeagueFacadeBean implements LeagueFacadeRemote {

    LeagueFacade leagueFacade = IceHockeyFactory.getLeagueFacade();

    public List<Schedule> getSchedulesForLeague(String league) {
        return leagueFacade.getSchedulesForLeague(league);
    }
}
```
Running the EJB

- Our EJB is contained in an EAR
  - Select EAR Application
  - Run As -> Run On Server...
League Planet App Running...

- WTP Packages and deploys the EAR
- Starts the server
Testing the Web Service

- WTP provides a Web Service Explorer
  - UDDI Explorer
  - WSDL Explorer
  - Run -> Launch Web Services Explorer
  - Provide the URL for the WSDL to our service

http://127.0.0.1:8080/LeagueFacadeBeanService/LeagueFacadeBean?WSDL
Test a service
Service Layer Summary

We have:

- Reviewed business tier design
- Developed domain logic
- Created the service layer
- Implemented Service Components with EJBs
- Created and Tested JAX-WS Web Services
Agenda

PART I
- WTP Quick Tour
- Setting up Your Workspace
- The Presentation Tier

PART II
- Building The Service Layer
  - Business and Persistence Tier
  - Web Services
Web Services Introduction

- Web services have emerged as the standard technology for integrating heterogeneous systems across the Internet and intranet.

- The key technologies are:
  - Extensible Markup Language: XML – for messages
  - XML Schema Description: XSD – for message description
  - Web Service Description Language: WSDL – for service description
  - Universal Description, Discovery, and Integration: UDDI – for service registries
  - Web Service Interoperability: WS-I – for interoperability
Top-Down Web Services

- Top-Down development means:
  - Designing the Web service interface first and then developing the implementation code
  - This approach yields the best interoperability because the underlying implementation details do not “bleed through” into the interface

- Best when:
  - Messages use existing industry or corporate XML document formats

- You need:
  - XSD and WSDL design skills

- WTP has two great editors that make this task easier
Top-Down Overview

• Tasks
  1. Use the XSD editor to describe the League Planet schedule format
  2. Use the WSDL editor to describe a Web service for querying schedules
  3. Use the Web service wizard to generate a Java skeleton for the service and deploy it to the Axis SOAP engine running on Tomcat
  4. Fill in the implementation of the skeleton by accessing the League Planet Service Layer
  5. Use the Web service explorer to test the Web service
LeaguePlanetService Web Project

- Create a new Dynamic Web Project
  - LeaguePlanetService to contain the Web service.
Describing schedule.xml

- League Planet has an XML format for schedules
  - Example
    - schedule.xml

- Task
  1. Your goal is to describe this format using XSD

```xml
<?xml version="1.0" encoding="UTF-8"?>
<schedule id="1" xmlns="http://leagueplanet.com/resource/schedule"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://leagueplanet.com/resource/schedule schedule.xsd">
  <name>2005-2006 Regular Season</name>
  <league id="1">
    <name>Roswell Girls Hockey League</name>
  </league>
  <games>
    <game id="1">
      <dateTime>2006-01-07T19:00:00</dateTime>
      <arena id="1">
        <name>Hillyview High School</name>
      </arena>
      <visitor id="1">
        <name>Ladybugs</name>
      </visitor>
      <home id="2">
        <name>Vixens</name>
      </home>
      <score>
        <visitor>0</visitor>
        <home>7</home>
      </score>
    </game>
    <game id="2">
      <dateTime>2006-01-07T21:00:00</dateTime>
      <arena id="2">
        <name>Hillaryview High School</name>
      </arena>
      <visitor id="2">
        <name>Ladybugs</name>
      </visitor>
      <home id="3">
        <name>Vixens</name>
      </home>
      <score>
        <visitor>7</visitor>
        <home>0</home>
      </score>
    </game>
  </games>
</schedule>
```
Designing a Meta-Object Model

- Services will use XML based meta-models
  - Create a new XML Schema file named schedule.xsd in LeaguePlanetService.
- Describing a model with XSD
  - Web services:
    - Use formats that work well with XML data binding toolkits
      - i.e. JAX-RPC
  - Define complexTypes for the content model of each element
- The XSD editor lets you edit in the source tab, the graphical tab, the outline view, and the property view
XSD editor.

View schedule.xsd in the Graph tab of the XSD Editor.
Defining a WSDL: query.wsdl

- Describe a Web service
  - will have a single operation
    - getSchedule
      - Input: Schedule id,
      - Output: Schedule document (XML)
WSDL editor

*WSDL describes Web service using*
  - Message
  - PortType,
  - Binding
  - Service

*The editor has a wizard that generates binding content for you*
  - Graph tab, Source tab,
  - Outline view, and the property view.
WSDL editor.

View query.wsdl in the Graph tab of the WSDL Editor.
Generating a Top-Down Service

• Web service is described with the WSDL, data types are defined in XSD
  – Create a Web service and deploy it

• Tasks
  1. Use Tomcat (You must have defined a server already)
  2. Select query.wsdl and execute the command
     – Web Services > Generate Java bean skeleton
  3. Use the Web service wizard to develop the Java service
Web Service Wizard

- Choose the Server
  - Tomcat

- Web Service Runtime
  - Apache Axis

- Customize the steps
  - Develop
  - Deploy
  - Test
  - Etc..
What the wizard does.

1. Installs the Axis SOAP engine to the project
2. Generates the Java bean skeleton for the service
   - Java XML data binding classes in the src folder
3. Copies query.wsdl to
   WebContent/wsdl/querySOAP.wsdl
   - Sets endpoint to your Web application
     • also copies dependent files such as schedule.xsd
4. Created the Axis deployment descriptor
   - WebContent/WEB-INF/server-config.wsdd
5. Creates Axis files to deploy and undeploy
   - Web service in a subfolder of WebContent/WEB-INF
6. Starts Tomcat to make the Web service available
Testing the Web Service

- Select AxisServlet
  - execute the command
    - Run As > Run on Server.

And now... Some Services
- querySOAP (read)
  - getSchedule
- AdminService (read)
  - AdminService
- Version (read)
  - getVersion

View the list of deployed Web services.
Implementing the Web Service

- Implementation of the Java bean skeleton generated by the Wizard
  - Use the classes in the LeaguePlanetModel Business Tier
  - The Web service is running but it just returns null at this point
- Link the Service Project with the LeaguePlanetModel

![Properties for LeaguePlanetService](image)
Module structure

- Module dependency structure is visible in the Servers view.
Filling in the Java bean skeleton

**Skeleton Class**
- com.leagueplanet.ws.query.QuerySOAPImpl

```java
public class QuerySOAPImpl
    implements Query_PortType{
    public ScheduleContent getSchedule(GetScheduleRequest request)
        throws java.rmi.RemoteException {
        return null;
        // WILL REPLACE THIS WITH SMT LIKE
        // return new QueryHelper().getSchedule(request);
    }
}
```
Shortcut: Import the solution

- Import classes
  - QuerySOAPImpl.java

- Skeleton simply delegates
  - to a new class com.leagueplanet.Query
  - to avoid confusion with the generated code
  - Import QueryHelper.java before proceeding
Testing the Web service

- At this point the Web service is ready to test
  - Use the Web Service Explorer
  - Select WebContent/wsdl/querySOAP.wsdl
    - Web Services > Test with Web Service Explorer
Running a service

- View the `getSchedule` operation.
  - Enter 1 in the scheduled field
  - Click **Go**.
  - The schedule is returned in the Status pane.
**SOAP Message**

- View the SOAP message source.
  - Click the Form
Attributions

- IBM and WebSphere are registered trademarks of International Business Machines Corp. in the United States, other countries, or both.
- Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- Other company, product, or service names may be trademarks or service marks of others.