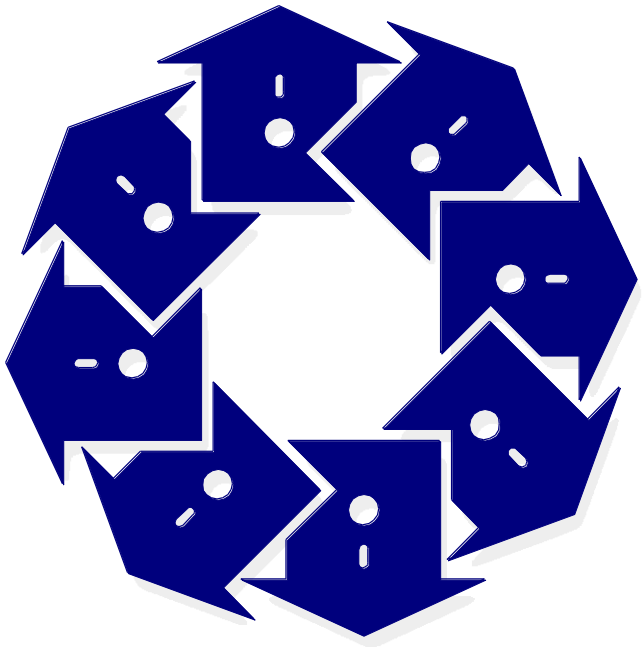


FairCom®

Tutorial

c-treeACE Web Service Using Java

Tutorial

c-treeACE Web Service Using Java

Trademarks

FairCom welcomes your comments on this document and the software it describes. Send comments to:

Documentation Comments
FairCom Corporation
6300 W. Sugar Creek Drive
Columbia, MO 65203

9/28/2012

CONTENTS

Creating a c-treeACE Web Service for JEE

This tutorial will explore the process of creating a Web Service that makes a c-treeACE database available to JEE (Java Enterprise Edition) applications. JEE allows developers to combine modules built with EJB (Enterprise Java Beans) to create powerful enterprise applications.

We will use NetBeans to create the Web Service and GlassFish to deploy it. Developers working in a Windows environment could instead use ADO.NET to create a Web Service. Your choice of .NET or Java largely depends on the environment you prefer; c-treeACE provides support for both.

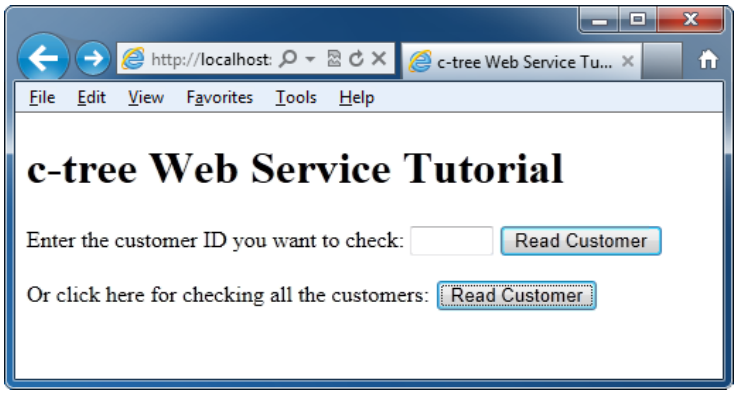
The Web Service will allow the database to be accessed by other applications; in this case, a web page. You can use these principles to make your own database available as a Web Service to JEE applications of your choosing.

The Web Service in this tutorial will provide two operations:

- **SearchByCustNum**: return a single customer record from a customer file (*custmast*) based on the customer ID
- **SelectAll**: return all the customer records in the customer file

Typical operations you may want a Web Service to perform might include the ability to add, edit, or delete records. You could perform complex queries specific to your application. Or you might invoke stored procedures containing your business logic.

In this tutorial, we will interact with the Web Service using a web page that contains two Submit buttons labeled Read Customer (one button for each of the two operations):



Prerequisites

For this tutorial, it will help to have a basic knowledge of Web Services and the c-treeACE JDBC driver (you will need c-treeACE V9.*x* for this tutorial). And, of course, you will need to have NetBeans and GlassFish set up and ready to go. You will also need a Java Developers Kit (JDK) and Java Runtime Environment (JRE).

This example will use the c-treeACE sample custmast database. If you have run any of the standard c-treeACE tutorials, you will have a custmast database and it will contain a few records. If you do not have a custmast database, you will need to create one. Either run one of the c-treeACE tutorials or use the c-treeACE SQL Explorer tool to create one. The image in the **Try it out...** (page 19) section at the end of the tutorial shows typical sample data; feel free to make up your own.

In this tutorial you will:

- **Configure GlassFish** (page 3) - Before you begin, you will need to configure GlassFish.
- **Create a Project in NetBeans** (page 4) - You will begin by creating a project in NetBeans.
- **Set Up the DataSource** (page 6) - Next, you will set up a DataSource in GlassFish using c-treeACE SQL JDBC.
- **Create the Web Service** (page 9) - Then you will create the Web Service:
  - First you will create an "empty" Web Service, which will be a file called *custmast.java*.
  - Next you will create the two operations: **SearchByCustNum** and **SelectAll**.
  - Then you will build and deploy the Web Service.
- **Create the Servlet** (page 13) - You will create a Servlet to connect the web page to the Web Service. The Servlet will call the appropriate operation in the Web Service (either **SearchByCustNum** or **SelectAll**).
- **Edit the Web Page** (page 18) - Finally, you will edit the initial web page, *index.jsp*, to create two Submit buttons labeled **Read Customer**: one will call **SearchByCustNum**; the other will call **SelectAll**.
- **Try it out...** (page 19)

You will find **Additional Resources** (page 20) at the end of this document.

1.1 Configure GlassFish

Before we begin, we need to configure GlassFish for c treeACE.

You will need to edit *domain.xml*, located in:
<GLASS_FISH_INSTALL_DIR>\glassfish\domains\domain1\config

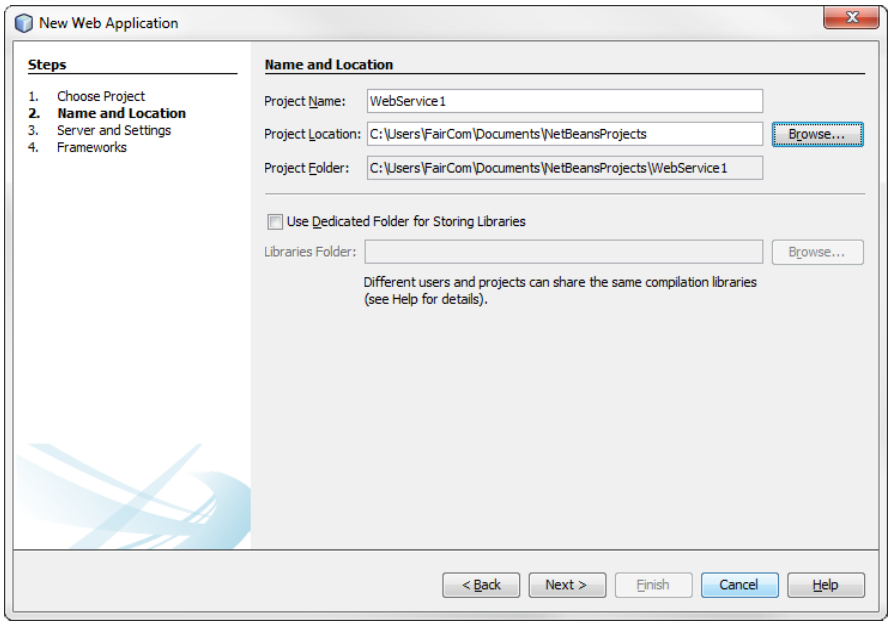
Change these settings:

- Change **PermSize** to: **-XX:PermSize=128m**
- Change **MaxPermSize** to: **-XX:MaxPermSize=512m**
- Change **Memory Usage** to: **-Xmx1024m**

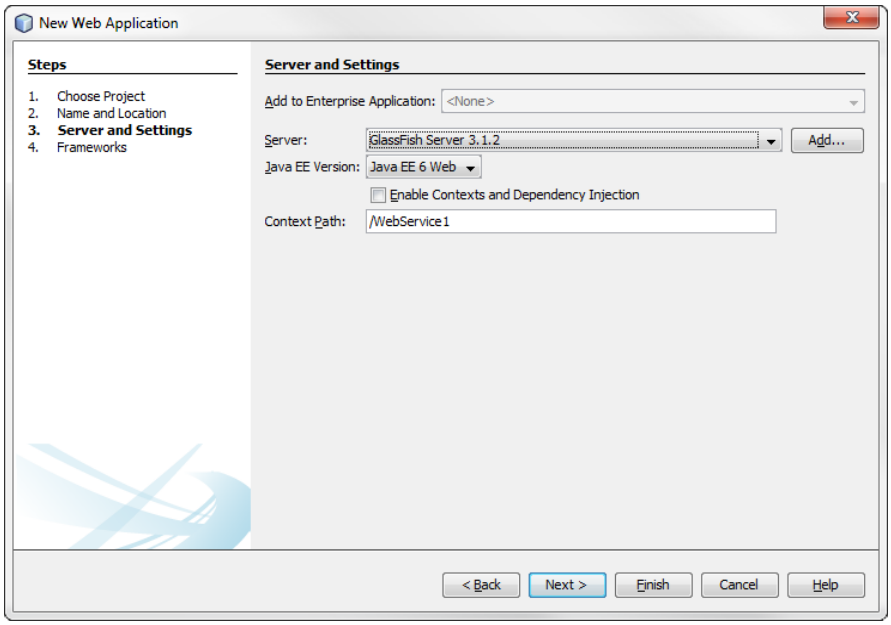
1.2 Create a Project in NetBeans

We will begin by creating the project so we can configure the JDBC DataSource:

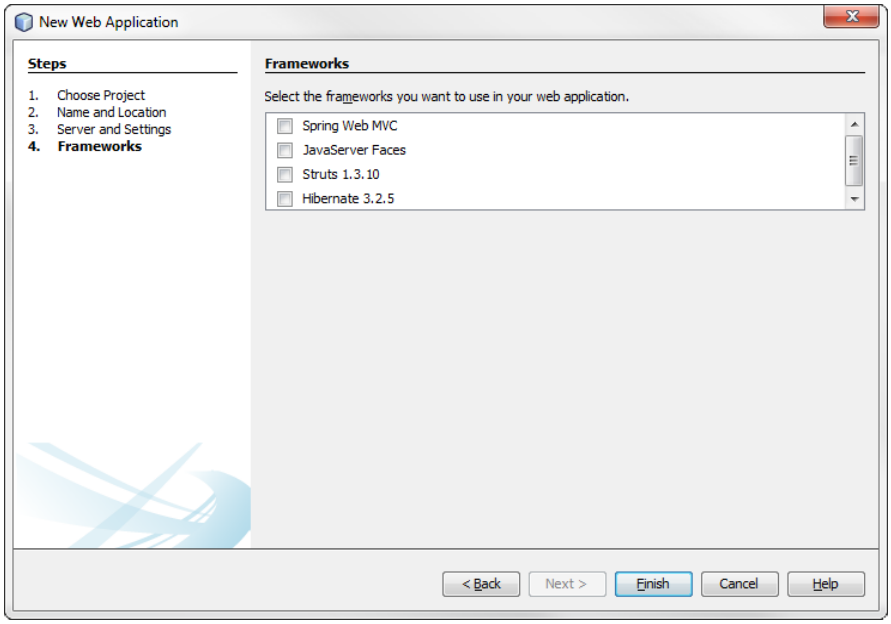
1. Create a new **Web Application** project (click **File > New Project** then **Java Web** and **Web Application**). We will call the project **WebService1**.



2. Click **Next** to configure the Server and Java EE information. We will use the GlassFish Server and Java EE 6 Web.



3. No frameworks will be used in this tutorial.

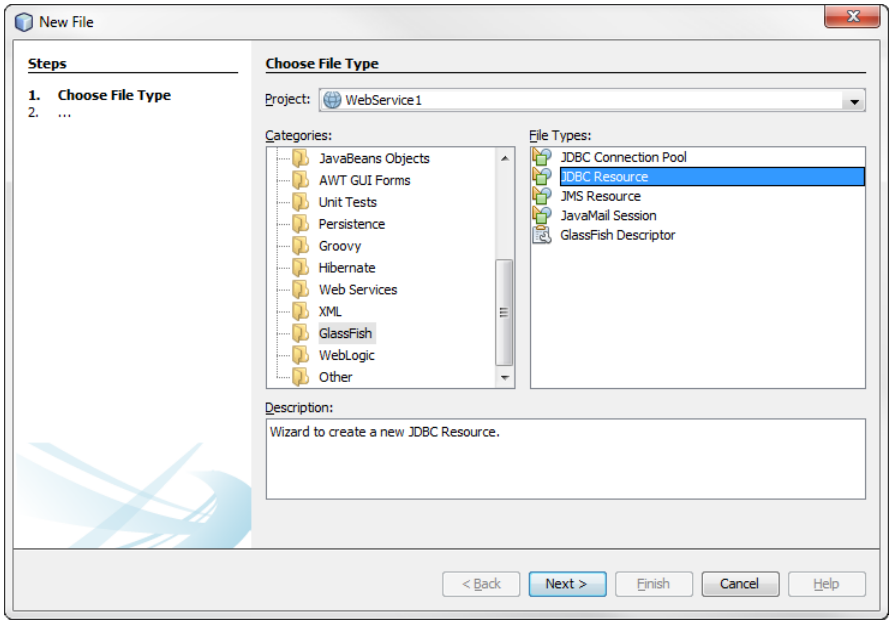


4. Click **Finish** to create the project.

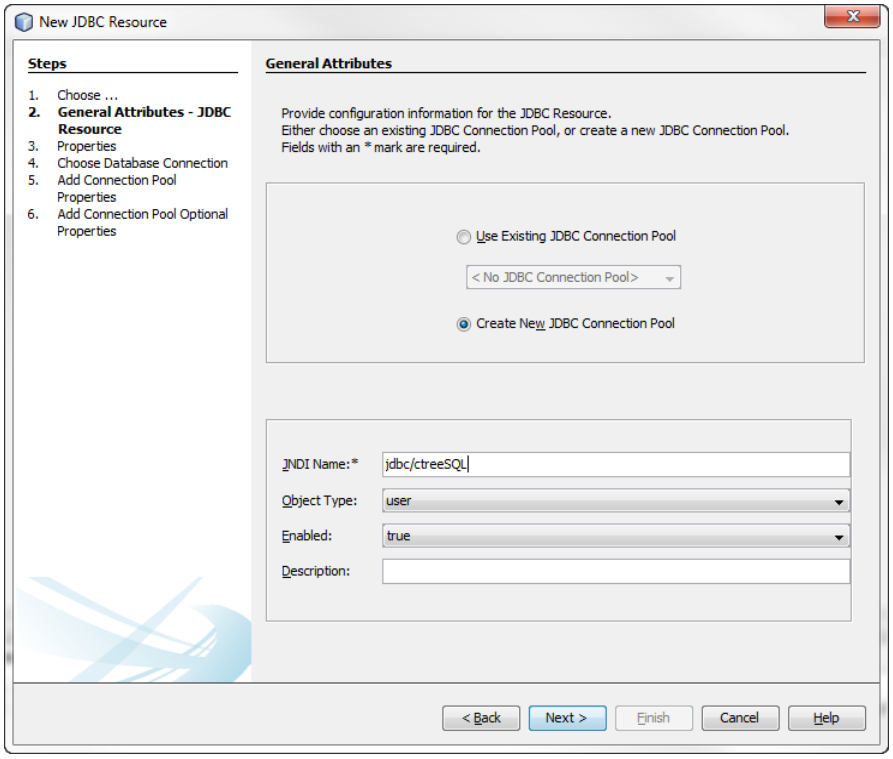
1.3 Set Up the DataSource

Next, we will set up a JDBC DataSource in GlassFish:

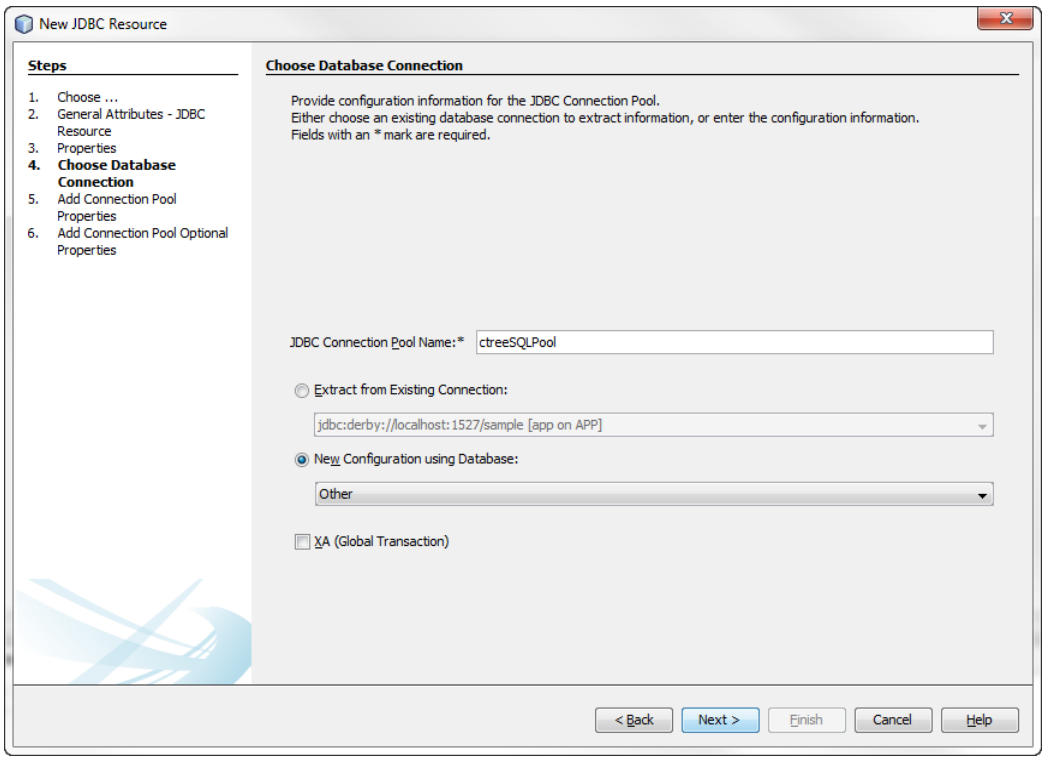
1. Using Windows Explorer or the command prompt, copy the *ctreeJDBC.jar* file (shipped with c-treeACE) from the *<CTREE_INSTALL_DIR>\lib\sql.jdbc* directory to *<GLASS_FISH_INSTALL_DIR>\lib*
2. Start the GlassFish server.
3. In NetBeans, click **File > New File** and select **GlassFish > JDBC Resource**.



4. Choose **Create New JDBC Connection Pool** and enter the **JNDI Name**: **jdbc/ctreeSQL**

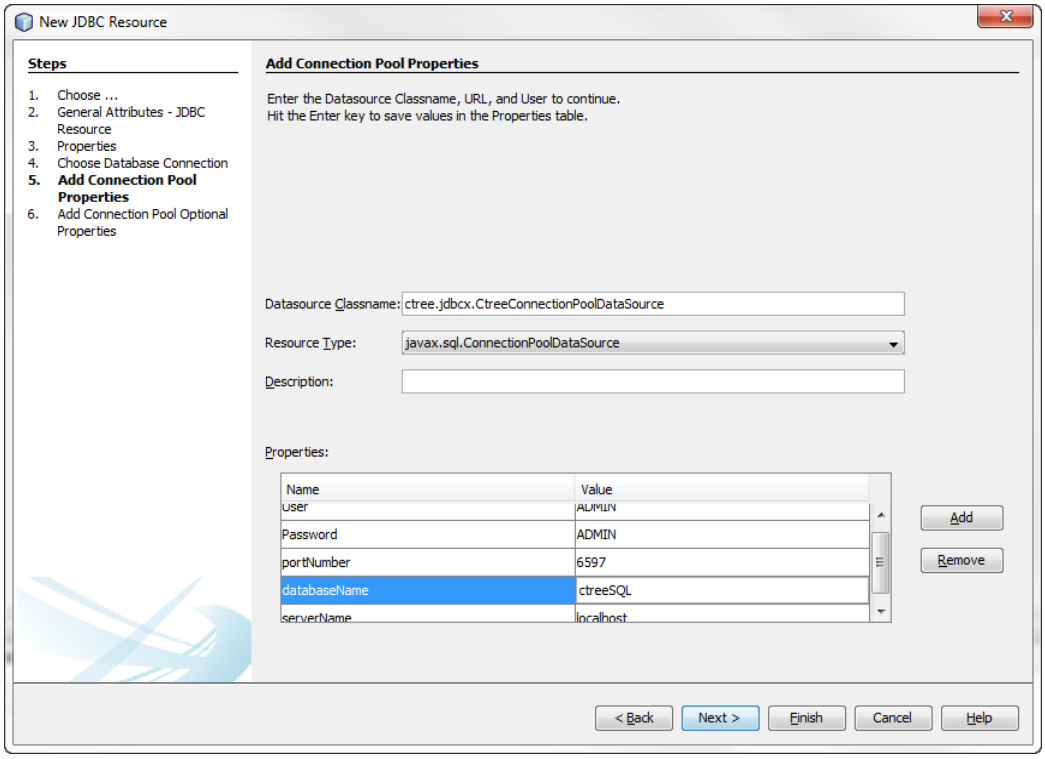


5. You will not enter anything in **Additional Properties**, so click **Next** on the subsequent screen.
6. Enter the **JDBC Connection Pool Name**, **ctreeSQLPool**, and select **Other** for the **New Configuration using Database** option.



7. Enter the **DataSource Classname**: **ctree.jdbcx.CtreeConnectionPoolDataSource**. For the **Resource Type**, select **javax.sql.ConnectionPoolDataSource**. Add the following properties:

   User = **ADMIN**
   Password = **ADMIN**
   portNumber = **6597**
   databaseName = **ctreeSQL**
   serverName = **localhost**



8. Click **Finish** to create the connection resources.

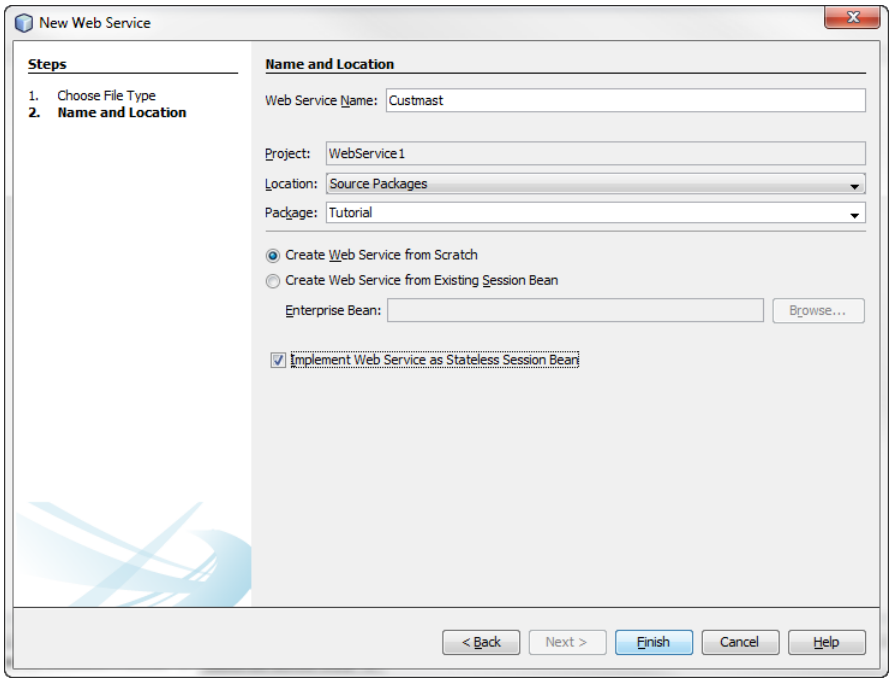
1.4 Create the Web Service

Now it is time to create the Web Service.

Begin Creating the Web Service

First we will create an "empty" Web Service (this will be a file called *custmast.java*):

1. Right-click the project and choose **New > Web Service**.
2. Enter the **Web Service Name** (we will call it **Custmast**). In the **Package** box type **Tutorial**, and check **Implement Web Service as Stateless Session Bean**.

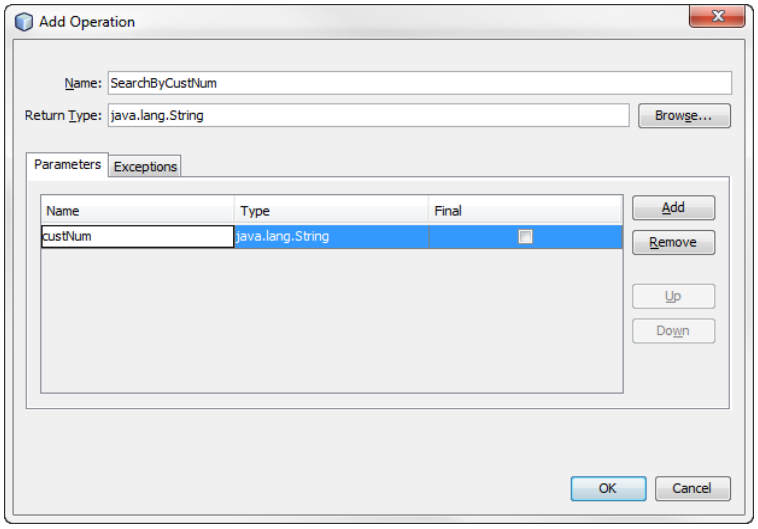


3. Click **Finish** to create the Web Service.

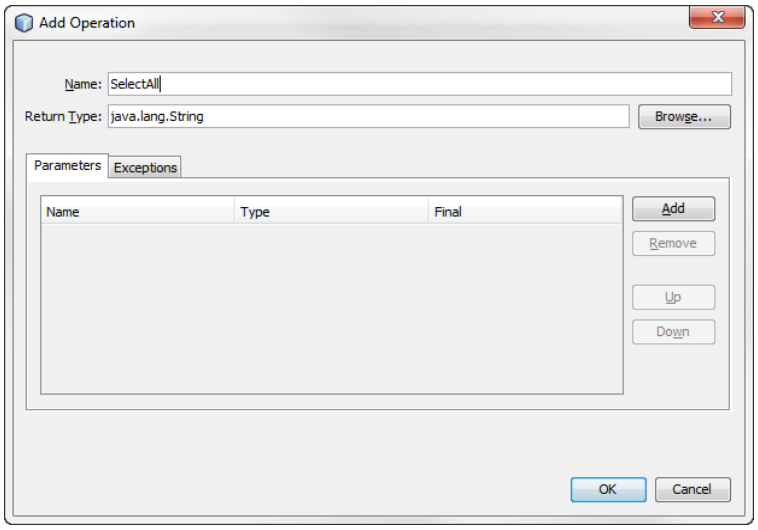
Add Operations

Having created our Web Service, we must add the operations it will perform. For this tutorial, we will create two operations: **SearchByCustNum** and **SelectAll**:

1. Expand the **Web Services** group. Right-click on the new **Custmast** Web Service and select **Add Operation**.
2. Enter an operation **Name** (call it **SearchByCustNum**). **SearchByCustNum** will need a single parameter (call it **custNum**). Click **Add** to add the parameter.



3. Repeat the steps above to create a second operation in your web service called **SelectAll**. **SelectAll** does not need a parameter.



The new operations are implemented by adding Java code in the method's body. You will need to edit the Java in the *Custmast.java* file to make it appear as shown below:

Custmast.java

```
package Tutorial;

import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.logging.Level;
import java.util.logging.Logger;
import javax.jws.WebService;
import javax.jws.WebMethod;
import javax.jws.WebParam;
import javax.ejb.Stateless;
import javax.naming.Context;
import javax.naming.InitialContext;
import javax.naming.NamingException;
```

```
import javax.sql.DataSource;

/**
 *
 * The Custmast provides a basic web service for retrieving data from the Custmast
 * table.
 *
 * @author FairCom Corporation
 */
@WebService(serviceName = "Custmast")
@Stateless()
public class Custmast {

    /**
     * Searches for a Custmast record based on the "customer number" information
     * (Primary Key)
     *
     * @param custNum Customer number to searched.
     *
     * @see SelectAll
     **/
    @WebMethod(operationName = "SearchByCustNum")
    public String SearchByCustNum(@WebParam(name = "custNum") String custNum) {

        String resultValue = "";

        // Retrieve the list of custmast records from the database connection
        try {
            Statement statement = getStatement();
            ResultSet resultSet = statement.executeQuery("select * from custmast where
cm_custnumb = " + custNum);

            // Build the output string with the list of records read
            while (resultSet.next())
                resultValue = resultValue + getRecordText(resultSet) + "<br />";

        } catch (SQLException ex) {
            resultValue = "Couldn't retrieve the custmast - ";
            resultValue.concat(ex.toString());
            Logger.getLogger(Custmast.class.getName()).log(Level.SEVERE, null, ex);
            return resultValue;
        }

        return resultValue;
    }

    /**
     * Retrieve all the Custmast records and build an string with this data
     *
     * @see SearchByCustNum
     **/
    @WebMethod(operationName = "SelectAll")
    public String SelectAll() {

        String resultValue = new String();

        // Retrieve the list of custmast records from the database connection
        try {
            Statement statement = getStatement();
            ResultSet resultSet = statement.executeQuery("select * from custmast");

            // Build the output string with the list of records read
            while (resultSet.next()) {
                resultValue = resultValue + getRecordText(resultSet) + "<br /> ";
            }

        } catch (SQLException ex) {
            resultValue = "Couldn't retrieve the list of custmasts - ";
            resultValue.concat(ex.toString());
            Logger.getLogger(Custmast.class.getName()).log(Level.SEVERE, null, ex);
            return resultValue;
```

```
        }

        return resultValue;
    }

    /**
     * Creates an Statement object for the configured JDBC Resource in the
     * application server
     **/
    private Statement getStatement() throws SQLException {

        // Retrieve the JDBC Resource from the application server
       Context ctx;
        DataSource ds;
        try {
            ctx = new InitialContext();
            ds = (DataSource) ctx.lookup("jdbc/ctreeSQL");
        } catch (NamingException ex) {
            Logger.getLogger(Custmast.class.getName()).log(Level.SEVERE, null, ex);
            return null;
        }

        // Retrieve the SQL Connection from the resource and create a Statement
        Connection con = ds.getConnection();
        return con.createStatement();
    }

    /**
     * Builds a string with a Custmast record information
     **/
    private String getRecordText(ResultSet resultSet) throws SQLException {

        String resultValue = resultSet.getString(1).trim() + ", ";
        resultValue = resultValue + resultSet.getString(2).trim() + ", ";
        resultValue = resultValue + resultSet.getString(3).trim() + ", ";
        resultValue = resultValue + resultSet.getString(4).trim() + ", ";
        resultValue = resultValue + resultSet.getString(5).trim() + ", ";
        resultValue = resultValue + resultSet.getString(6).trim() + ", ";
        resultValue = resultValue + resultSet.getString(7).trim();
        return resultValue;
    }
}
```

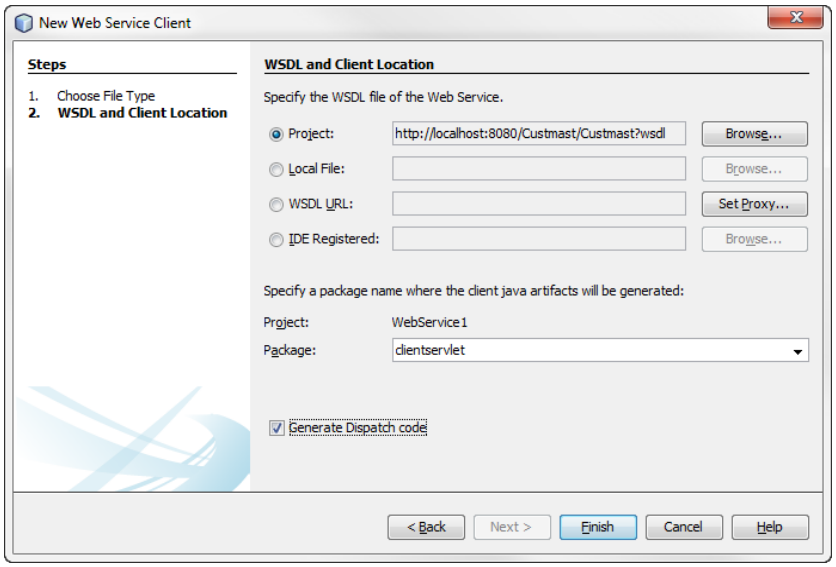
Finish the Web Service

The final step in creating the Web Service is to build and deploy it. We do that by right-clicking the Web Service project (**WebService1**) and selecting **Deploy**.

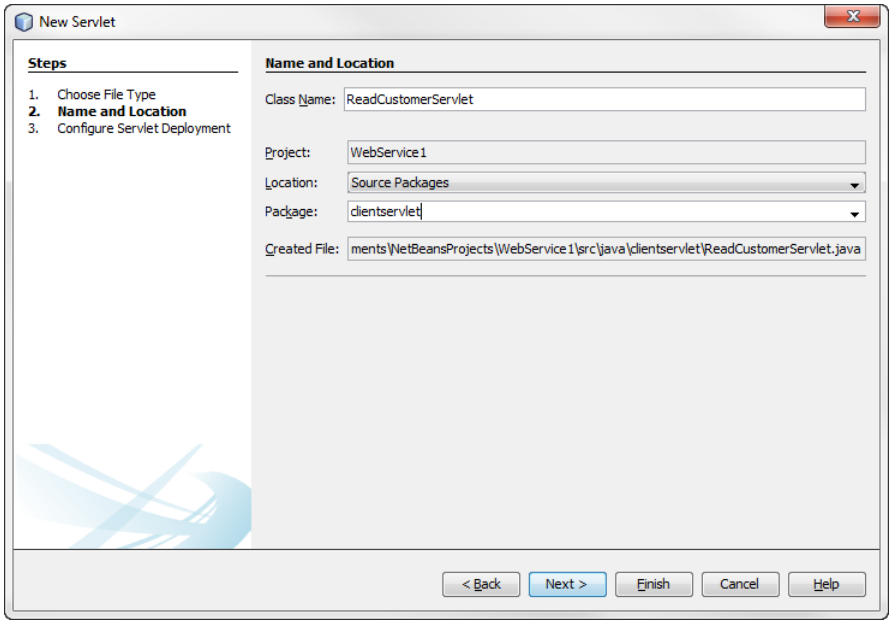
1.5 Create the Servlet

In the JEE environment, we use a Servlet to connect the .JSP page to the Web Service. Think of a Servlet as a little like a CGI script or an .ASP page that is executed when you click Submit. The Servlet calls the appropriate operation in the Web Service, either **SearchByCustNum** or **SelectAll**.

1. Right-click the Web Service project and select **New > Web Service Client**.
2. For **Project**, browse to the **Custmast** Web Service project. In the **Package** field type the name of the package (call it **clientservlet**) and check **Generate Dispatch code**.



3. Right-click the **WebService1** project and select **New > Servlet** module.
4. Enter the **Class Name** (e.g., **ReadCustomerServlet**) and **Package** (e.g., **clientservlet**).



5. Accept the default values and click **Finish**.

6. Edit the Servlet module **ReadCustomerServlet.java** to implement the **searchByCustNum** Web Service call. The resultant source file will appear as shown below:

ReadCustomerServlet.java

```
package clientservlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;
import javax.xml.transform.Source;
import javax.xml.ws.Dispatch;

/**
 *
 * @author FairCom Corporation
 */
@WebServlet(name = "ReadCustomerServlet", urlPatterns = {"/ReadCustomerServlet"})
public class ReadCustomerServlet extends HttpServlet {
    @WebServiceRef(wsdlLocation = "WEB-INF/wsdl/localhost_8080/Custmast/Custmast.wsdl")
    private Custmast_Service service;

    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String CustNumb = request.getParameter("CustNumb");
        String result = null;

        try { // Call Web Service Operation
            Dispatch<Source> sourceDispatch = null;
            result = service.getCustmastPort().searchByCustNum(CustNumb);
        } catch (Exception ex) {
            // TODO handle custom exceptions here
        }

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet ReadCustomerServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>c-tree Web Service Response:</h1>");
            out.println("<h3>Customer:</h3>" + result);
            out.println("<a href=\"javascript:window.history.go(-1)\">Back</a>");
            out.println("</body>");
            out.println("</html>");
        } finally {
            out.close();
        }
    }

    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on
the left to edit the code.">
    /**
```

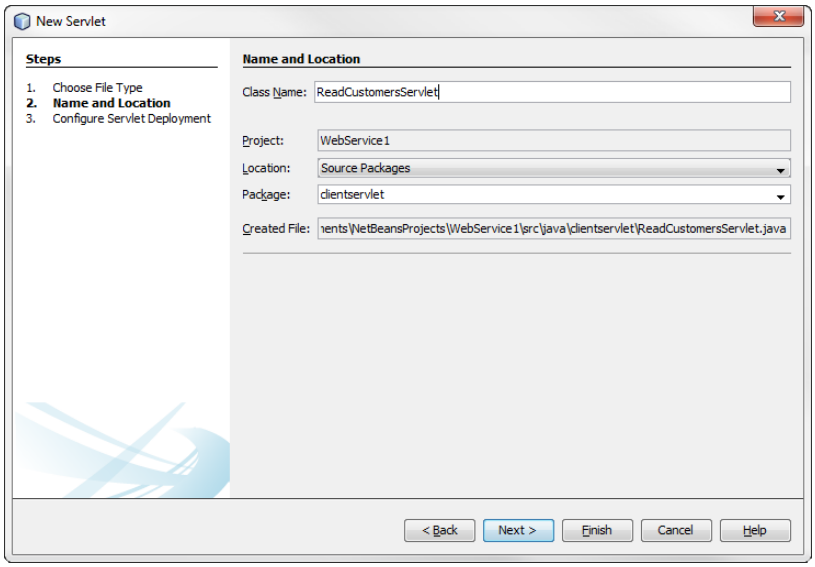
```
     * Handles the HTTP
     * <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP
     * <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

Create the second Servlet:

1. Repeat steps 3-5 to create a second Servlet called **ReadCustomersServlet**.



2. Edit the Servlet module **ReadCustomersServlet.java** to implement the **selectAll** web service call. The resultant source file will appear as shown below:

ReadCustomersServlet.java

```
package clientservlet;

import java.io.IOException;
import java.io.PrintWriter;
import javax.servlet.ServletException;
import javax.servlet.annotation.WebServlet;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.ws.WebServiceRef;
import javax.xml.transform.Source;
import javax.xml.ws.Dispatch;

/**
 *
 * @author FairCom Corporation
 */
@WebServlet(name = "ReadCustomersServlet", urlPatterns = {"/ReadCustomersServlet"})
public class ReadCustomersServlet extends HttpServlet {
    @WebServiceRef(wsdlLocation = "WEB-INF/wsdl/localhost_8080/Custmast/Custmast.wsdl")
    private Custmast_Service service;

    /**
     * Processes requests for both HTTP
     * <code>GET</code> and
     * <code>POST</code> methods.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    protected void processRequest(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {

        String result = null;

        try { // Call Web Service Operation
            Dispatch<Source> sourceDispatch = null;
            result = service.getCustmastPort().selectAll();
        } catch (Exception ex) {
            // TODO handle custom exceptions here
        }

        response.setContentType("text/html;charset=UTF-8");
        PrintWriter out = response.getWriter();
        try {
            out.println("<html>");
            out.println("<head>");
            out.println("<title>Servlet ReadCustomerServlet</title>");
            out.println("</head>");
            out.println("<body>");
            out.println("<h1>c-tree Web Service Response:</h1>");
            out.println("<h3>Customers:</h3>" + result);
            out.println("<a href=\"javascript:window.history.go(-1)\">Back</a>");
            out.println("</body>");
            out.println("</html>");


        } finally {
            out.close();
        }
    }
```

```
    // <editor-fold defaultstate="collapsed" desc="HttpServlet methods. Click on the + sign on
the left to edit the code.">
    /**
     * Handles the HTTP
     * <code>GET</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doGet(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Handles the HTTP
     * <code>POST</code> method.
     *
     * @param request servlet request
     * @param response servlet response
     * @throws ServletException if a servlet-specific error occurs
     * @throws IOException if an I/O error occurs
     */
    @Override
    protected void doPost(HttpServletRequest request, HttpServletResponse response)
            throws ServletException, IOException {
        processRequest(request, response);
    }

    /**
     * Returns a short description of the servlet.
     *
     * @return a String containing servlet description
     */
    @Override
    public String getServletInfo() {
        return "Short description";
    }// </editor-fold>
}
```

1.6 Edit the Web Page

Finally, we will need to edit the initial web page, *index.jsp*. We will create two Submit buttons labeled **Read Customer**: one will call **SearchByCustNum**; the other will call **SelectAll**.

1. Expand the **Web Pages** group.

2. Edit *index.jsp* so the <Body> of the HTML contains the following:

index.jsp

```
<body>
<h1>c-tree Web Service Tutorial</h1>
<p>
<form name="Test" method="post" action="ReadCustomerServlet">
    <p>Enter the customer ID you want to check:
       <input type="text" name="CustNumb" size="4" ID="CustNumb">
       <input type="submit" value="Read Customer" name="readcustomerbutton">
    </p>
</form>
<form name="Test2" method="post" action="ReadCustomersServlet">
    <p>Or click here for checking all the customers:
       <input type="submit" value="Read Customer" name="readcustomersbutton">
    </p>
</form>
</body>
```

1.7 Try it out...

We will test our Web Service using the web page we just made. The address will be:
http://localhost:8080/WebService1

- Enter a customer ID and click **Read Customer** to see information about a single customer.
  or
- Click the lower **Read Customer** button to show all customers.

We should see the customer record(s) from the custmast table. In this example, the page shows all of the records:

That's how easy it is to expand the power of your c-treeACE database by making it available to other JEE applications as a Web Service.

Stay tuned for more exciting tips and tricks from FairCom.

1.8 Additional Resources

We encourage you to explore the additional resources listed here:

- Additional documentation and the other tutorials may be found at www.FairCom.com (http://www.FairCom.com).
- NetBeans can be downloaded from www.netbeans.org (http://netbeans.org/).
- GlassFish can be downloaded from glassfish.java.net (http://glassfish.java.net/).