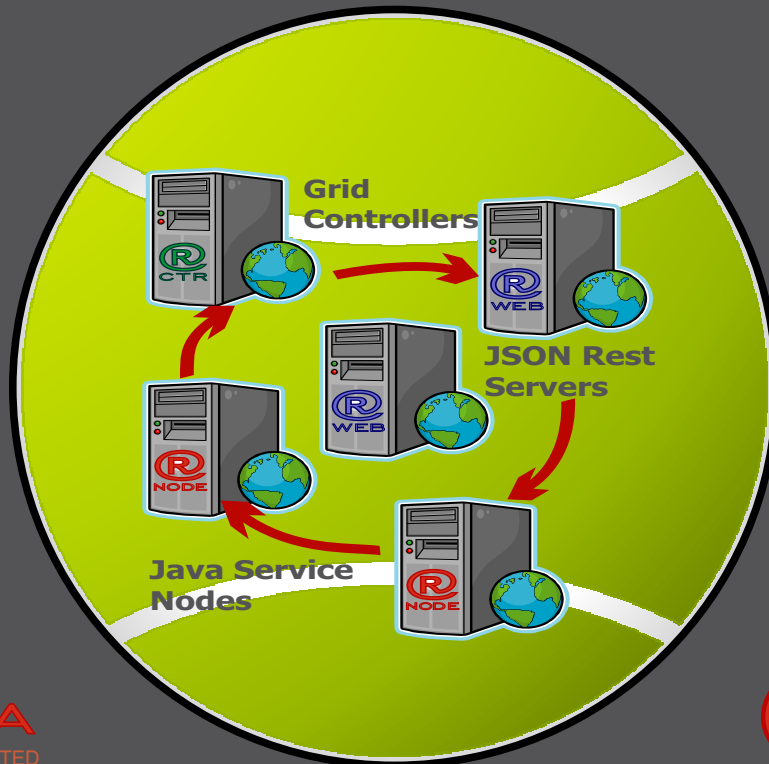


# Resoa Service & Web Development



**RESOA**  
REST SERVICE ORIENTED



## 1 Design



Open your XSD Design Tool

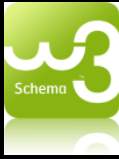
Create Elements & Types

Our example:  
Some Users might  
own some cars...

Create a User Element

User

# 1 Design

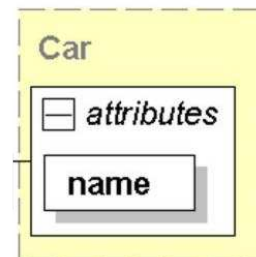


Open your XSD Design Tool

Create Elements & Types

Create a User Element

Create a  
ComplexType Car



# 1 Design



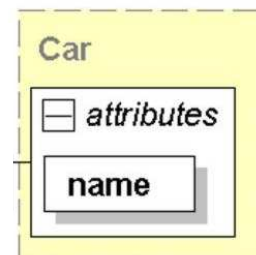
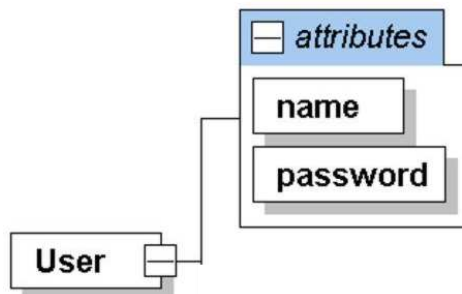
Open your XSD Design Tool

Create Elements & Types

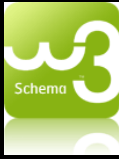
Create a User Element

Create a  
ComplexType Car

Assign the attributes



# 1 Design



Open your XSD Design Tool

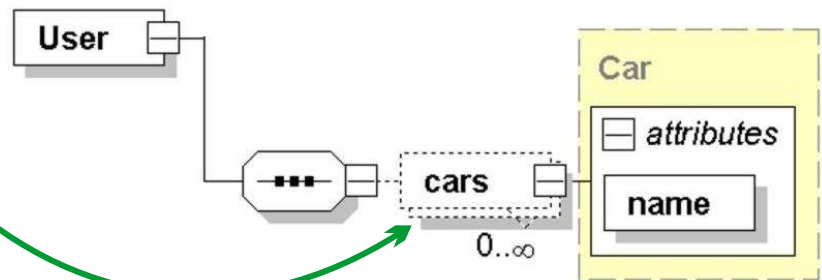
Create Elements & Types

Create a User Element

Create a  
ComplexType Car

Assign the attributes

Add a 'Car typed'  
element to User



# 1 Design



Open your XSD Design Tool

Create Elements & Types

Assign a TargetNamespace

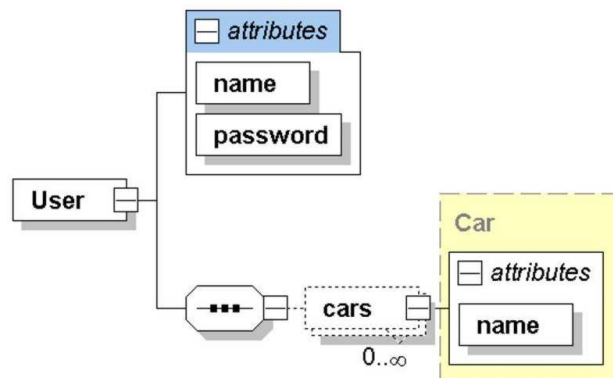
Create a User Element

Create a  
ComplexType Car

Assign the attributes

Add a 'Car typed'  
element to User

Assign the namespaces



```
<xs:schema xmlns="http://www.foo.org/user"
targetNamespace="http://www.foo.org/user"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
```

## 2 Develop



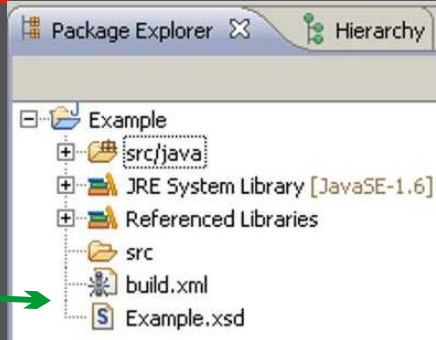
This example uses Eclipse IDE and ANT as build tool.

Prepare a project

Create a Project

Add a build.xml

Copy the XSD inside



## 2 Develop



This example uses Eclipse IDE and ANT as build tool.

Prepare a project

Create a Project  
Add a build.xml  
Copy the XSD inside

Add Taskdef for XJC compile

```
build.xml
<!-- XJC task definition -->
<target name="xjcdef">
  <taskdef name="xjc"
    classname="com.sun.tools.xjc.XJCTask">
    <classpath refid="classpath.jaxb2.xjc" />
  </taskdef>
</target>
```

## 2 Develop



This example uses Eclipse IDE and ANT as build tool.

Prepare a project

Create a Project  
Add a build.xml  
Copy the XSD inside

Add Taskdef for XJC compile

Add target for XJC compile

```
<!-- XJC task definition -->  
<target name="xjcdef">  
<taskdef name="xjc" classname="com.sun.tools.xjc.XJCTask">  
  <classpath refid="classpath.jaxb2.xjc" />  
</taskdef>  
</target>
```

build.xml

```
<!-- Compile by XJC -->  
<target name="xsd-compile" depends="xjcdef">  
  <!-- delete source dir first -->  
  <delete dir="${dir.src.xsd}" />  
  <!-- create src dir -->  
  <mkdir dir="${dir.src.xsd}" />  
  <!-- we compile xsd's -->  
  <xjc destdir="${dir.src.xsd}">  
    <arg line="-extension" />  
    <schema dir="."/>  
    <include name="Example.xsd"/>  
  </schema>  
</xjc>  
</target>
```

## 2 Develop

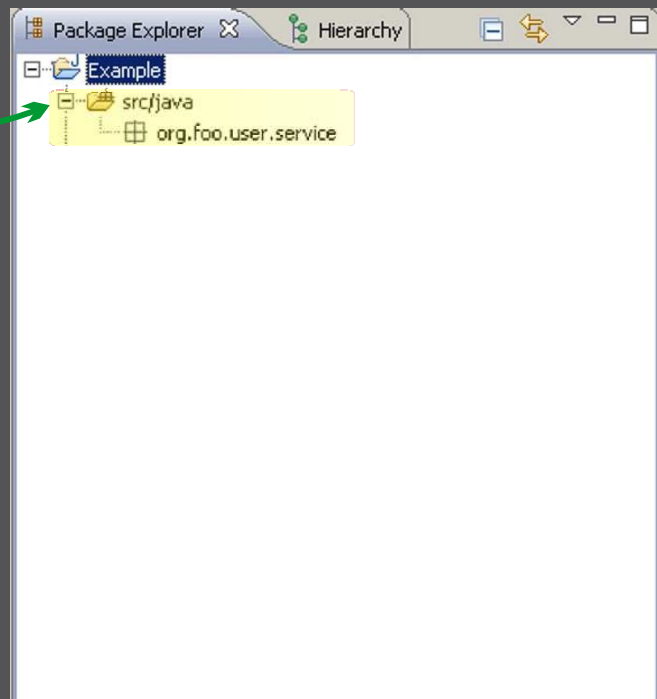


Prepare a project

Prepare to develop

Create the Service package

It extends the XSD Namespace by "Service".



## 2 Develop



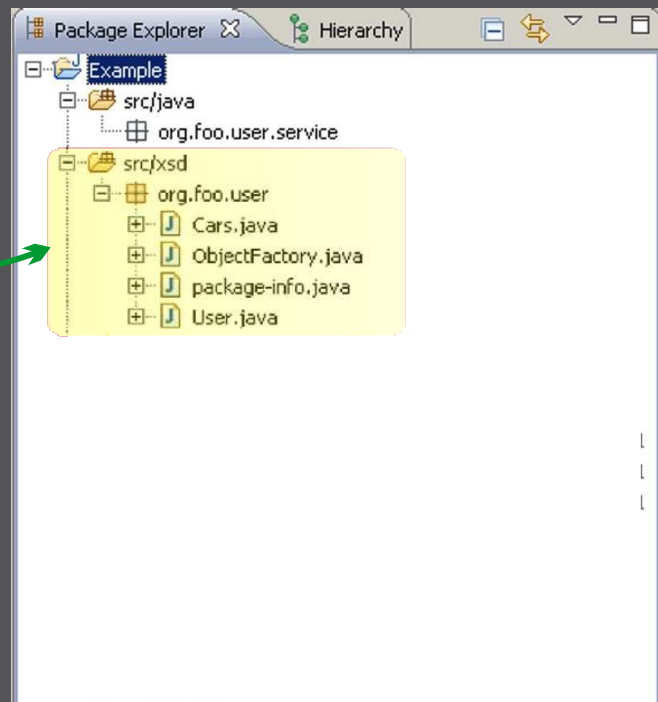
Prepare a project

Prepare to develop

Create the Service package

Compile the XSD by running ANT

This will create the Java representation of your XSD defined business model



## 2 Develop



Prepare a project

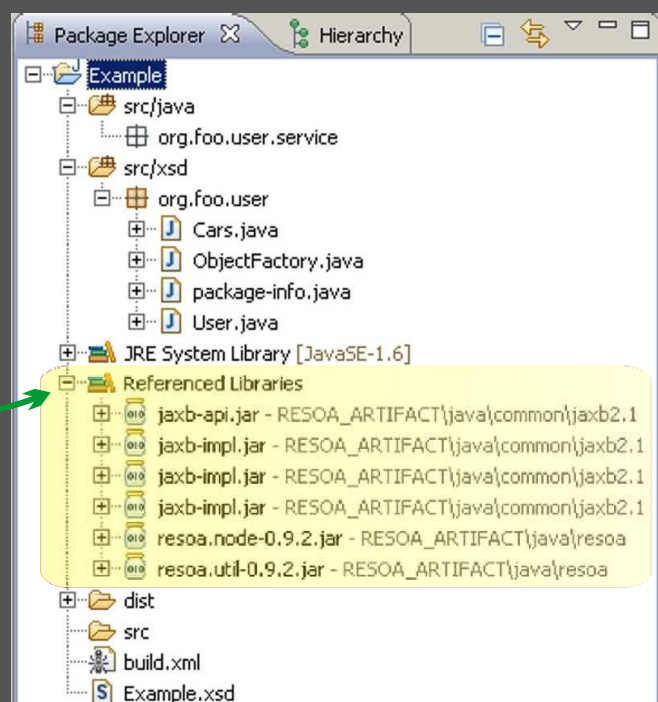
Prepare to develop

Create the Service package

Compile the XSD by running ANT

Define your dependencies

Ensure that you have defined **RESOA\_ARTIFACT** as System environment variable.



## 2 Develop



Prepare Environment

Develop Services

### Create Service Classes

Create a **class** for every XSD type, you like to **add business intelligence** to.

The **classname** must **extend** the XSD type name by "Service".

### Resoa Service Classes



name begins with XSD Type name

## 2 Develop



Prepare Environment

Develop Services

### Create Service Classes

### Implement Services by adding methods

Create an **init** method for generic initialization.

Services are **class methods**, identified by parameter sequence.

**UserService**

```
public class UserService {  
  
    /** This is invoked one during Node start */  
    public void init(ResoaGateway gateway) {  
        // add your initialization code here  
    }  
  
    /** The User create Service */  
    public void create (User usr, ResoaResponse rsp, ResoaGateway gw) {  
        // add your service implementation here  
    }  
  
    /** The User delete Service */  
    public void delete (User usr, ResoaResponse rsp, ResoaGateway gw) {  
        // add your service implementation here  
    }  
}
```

1st param is XSD type

3rd param is gateway

2nd param is Response handle

## 2 Develop



Prepare Environment

Develop Services

Create Service Classes

Implement Services  
by adding methods

Set response object  
and desired action

Response object must  
be a **XSD defined**  
type as well.

indicate success or  
error to the underlying  
request **transaction**.

```
public class UserService {  
  
    /** The User create Service */  
    public void create (User usr, ResoaResponse rsp, ResoaGateway gw) {  
        // add your service implementation here  
  
        rsp.setValue(usr, ResponseAction.COMMIT);  
    }  
  
    /** The User delete Service */  
    public void delete (User usr, ResoaResponse rsp, ResoaGateway gw) {  
        // add your service implementation here  
  
        // set the response object and action type  
  
        rsp.setValue(usr, ResponseAction.COMMIT);  
    }  
}
```

UserService

Create Service Classes

Implement Services  
by adding methods

Set response object  
and desired action

Use Resoa Persistence

Use the Resoa Persistor  
for accessing the  
**BTREE** based and  
**GRID** managed  
database storage.

Prepare Environment

Develop Services

```
/** The User create Service */  
public void create (User usr, ResoaResponse rsp, ResoaGateway gw) {  
    try {  
        gw.getPersistor().createNew(usr);  
    } catch (PersistenceException e) {  
        // TODO design and instance error Object  
        rsp.setValue(error, ResponseAction.ERROR);  
    }  
    rsp.setValue(usr, ResponseAction.COMMIT);  
}  
  
/** The User delete Service */  
public void delete (User usr, ResoaResponse rsp, ResoaGateway gw) {  
    try {  
        gw.getPersistor().delete(usr.getName());  
        rsp.setValue(usr, ResponseAction.COMMIT);  
    } catch (PersistenceException e) {}  
}
```

UserService

## 2 Develop



Prepare Environment

**Develop Services**

Create Service Classes

Implement Services by adding methods

Set response object and desired action

**Use Resoa Persistence**

### Resoa Persistor methods

**Read data from storage by key or regex expression.**

`read(class, key)`

`readRegex(class, regex)`

`readKeySet(class, regex)`

**Check for exists.**

`exists(class, key)`

**Store or update objects in storage.**

`store(instance)`

**Create new, not yet existing objects**

`createNew(instance)`

**Delete objects**

`delete(instance/key)`

## 2 Develop



Prepare

Develop

**Package**

Create resoa.xml

**Package to JAR**

The **name of JAR** file is the **Service Domain** used within Grid Deployment.

resoa.xml

**META-INF**

resoa.xml must reside in META-INF folder of JAR.

```
<Resoa xmlns="http://www.resoa.org/deploy">
```

```
</Resoa>
```

## 2 Develop



Prepare

Develop

Package

Create resoa.xml

Package to JAR

Define permissions for service invocation

Restrict access to services where required.

```
<Resoa xmlns="http://www.resoa.org/deploy">
```

resoa.xml

```
<!-- Service Permission declaration -->
<service className="org.foo.Car">
  <!-- Set a generic role restriction -->
  <roles>member</roles>
</service>
<service className="org.foo.User">
  <!-- Set role restriction on for dedicated methods -->
  <executions method="delete">
    <roles>admin</roles>
    <roles>member</roles>
  </executions>
</service>
</Resoa>
```

## 2 Develop



Prepare

Develop

Package

Create resoa.xml

Package to JAR

Define permissions

Declare types as persistent

Activate Resoa persistence for dedicated types.

Define type relations, which should be automatically resolved during database processing.

```
<Resoa xmlns="http://www.resoa.org/deploy">
```

resoa.xml

```
<!-- Persistence declaration -->
<persistent className="org.foo.User"
  encrypted="true"
  keyMethod="getName">
  <!-- Define relations to other types -->
  <relation fieldName="cars"
    type="CREATE_UPDATE"/>
</persistent>
<!-- Car must be declared as persistent as well -->
<persistent className="org.foo.Car"
  encrypted="false"
  keyMethod="getName">
</persistent>
</Resoa>
```

### 3 Develop



Web / Javascript

Create Javascript API

Create a Javascript library out of your XSD business model

build.xml

```
<!-- Javascript source code generator task -->

<target name="create.javascript">
  <java
    classname="org.resoa.util.codedom.CodeGenerator"
    classpathref="classpath.compile" fork="true">
    <arg value="dist" />
    <arg value="dist" />
    <arg value=
      "%RESOA_ARTIFACT%../java/resoa/util/prototype.function.xml" />
  </java>
</target>
```

1st arg is JAR source directory

2nd arg is target directory for .js

3rd arg is file with prototype functions

### 3 Develop



Web / Javascript

Use Javascript API

Create a Javascript library out of your XSD business model

#### prototype functions

Create instances

```
var i = new MyType(json)
```

Check type of JSON

```
var i = wrap(object)
```

Marshall to JSON / JSONP

```
prototype.instanceOf(json)
```

```
instance.toJSON(method, ..)
```

Auto-Fill business object from HTML form

HTML Form naming convention:  
name= "[Typename]\_[Attribute]"

```
instance.fromForm(f)
```

Auto-Fill HTML form with business object

```
instance.fillForm(f)
```

### 3 Develop



Web / Javascript

#### AJAX POST with JQuery

Prepare a generic function for an AJAX option object

// create a option object for our ajax call

init.js

```
org.foo.getAJAXOptions = function(onSuccess, onError) {
  var options = {
    contentType : "application/json; charset=utf-8",
    dataType : "json",
    url : "/service",
    cache : false,
    global : false,
    processData : false,
    type : "POST",
    complete : onSuccess,
    error : onError
  };
  return options;
};
```

The URL of your REST API

### 3 Develop



Web / Javascript

#### AJAX POST with JQuery

Prepare a generic function for an AJAX option object

Define a success and error function

// the success function

user.js

```
org.foo.onUserGet = function(req, textStatus){
  // check type of response JSON
  if (org.foo.User.prototype instanceof (req.responseText){
    // create instance out of JSON payload
    var usr = new org.foo.User(req.responseText);
    usr.fillForm($("#formedit"));
    //access properties by function or directly
    alert(usr.getName());
    alert(usr.name);
  });

  // the error function
  org.onError = function(req, textStatus) {
    // add your error handling here
  }
}
```

### 3 Develop



Web / Javascript

#### AJAX POST with JQuery

Prepare a generic function for an AJAX option object

Define a success and error function

Create your AJAX request object

Perform the request, invoke Resoa service

submit.js

```
// get AJAX option object
var ao = org.foo.getAJAXOptions
    (org.foo.onUserGet, onError);

// create request object
var usr = new org.foo.User();

// fill object from HTML form
usr.fromForm($("#formNewUser"));

// set JSON to option object for service "create"
ao.data = usr.toJson("create");

// perform ajax
$.ajax(ao);
```

### 3 Develop



Web / Javascript

#### AJAX with JSONP

Perform a JSONP service request

\*.html

```
<script type="text/javascript"
  src="/jsonp_api?
  {'User':{'name':'admin'},
  'method':'get',
  'callback':'org.foo.onUserGet'} ">
```

The URL of your JSONP API

The request object in JSON

The service to call

The function, the response should be wrapped into