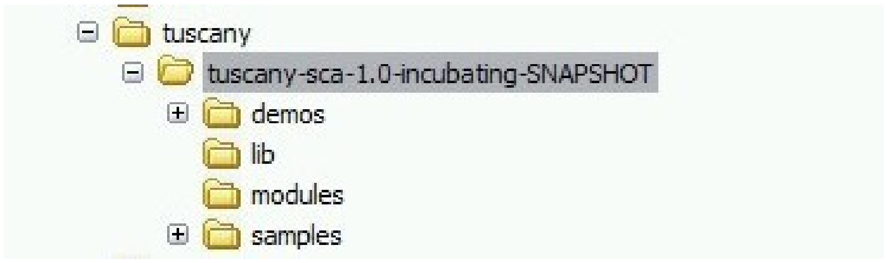
Ready, Set, Go – *Getting started with Tuscany*

Install the Tuscany Distribution

Create a folder (**we call it Tuscany in this doc**) on your disk into which you will download the TUSCANY distribution. Download the latest distribution from this URL (We use nightly for this exercise since this example requires code that is in the trunk until post .91 release becomes available):

Nightly Build - http://cwiki.apache.org/TUSCANY/tuscany-downloads-documentations.html

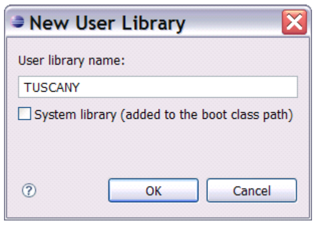
Download both the ***bin zip*** as well as the ***src zip*** (if available) to the folder that you created on your disk and unzip the ***bin zip*** in place, you should see the following folder file structure on your disk after unzip is complete.



Setup Eclipse for Tuscany

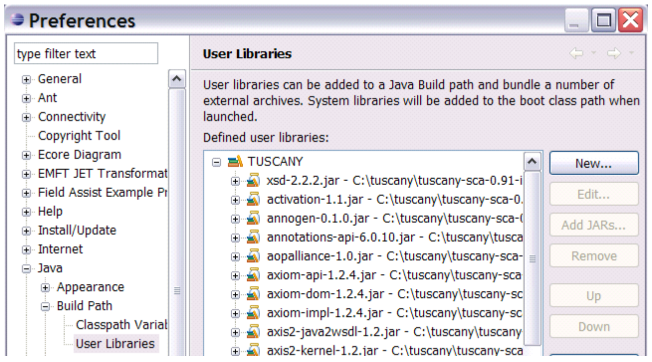
Start Eclipse and create a User Library to contain the TUSCANY runtime jar's as well as their depending jar's following the following steps:

From the menu bar select ***Window*** and then **Preferences**. The Preferences dialog will appear, in its left navigation tree select ***Java***, followed by ***Build Path***, and followed by ***User Libraries***. Select the ***New*** pushbutton on the right of the New Libraries dialog to create a new user library.



User library name:
TUSCANY
☐ System library (added to the boot class path)

The user library created is empty, select the ***Add JARs*** pushbutton on the right to add all the jar's from your Tuscany installation ***lib folder***. When completed all the jar's will appear under the TUSCANY user library.



User Libraries

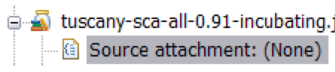
User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

```
TUSCANY
  xsd-2.2.2.jar - C:\tuscany\tuscany-sca-0.91-i
  activation-1.1.jar - C:\tuscany\tuscany-sca-0.
  annogen-0.1.0.jar - C:\tuscany\tuscany-sca-(
  annotations-api-6.0.10.jar - C:\tuscany\tusca
  aopalliance-1.0.jar - C:\tuscany\tuscany-sca-
  axiom-api-1.2.4.jar - C:\tuscany\tuscany-sca
  axiom-dom-1.2.4.jar - C:\tuscany\tuscany-sc
  axiom-impl-1.2.4.jar - C:\tuscany\tuscany-sc
  axis2-java2wsdl-1.2.jar - C:\tuscany\tuscany
  axis2-kernel-1.2.jar - C:\tuscany\tuscany-sca
```

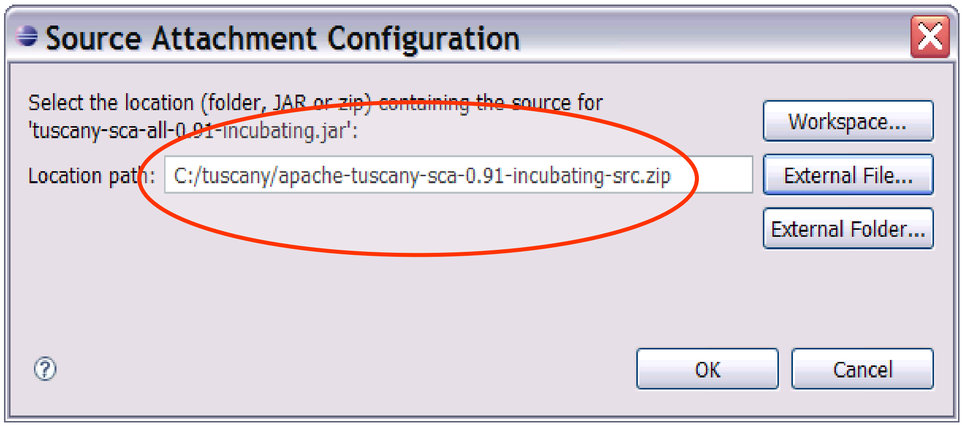
Setup for Debugging in Eclipse

Attach the Tuscany source to the Tuscany runtime jar in the following step. You do need to use the same source as where the binary is from. Please note that we are using this as an example to show how it is done since the nightly build does not have source zip file available yet.
In the User Libraies dialog scroll down until you see the ***Tuscany runtime jar*** and select its ***Source attachment***.



tuscany-sca-all-0.91-incubating.jar
Source attachment: (None)

Select the ***Edit*** pushbutton on the right and in the Edit dialog use the ***External File*** pushbutton to the select the Tuscany ***src zip*** that we downloaded earlier.



Source Attachment Configuration

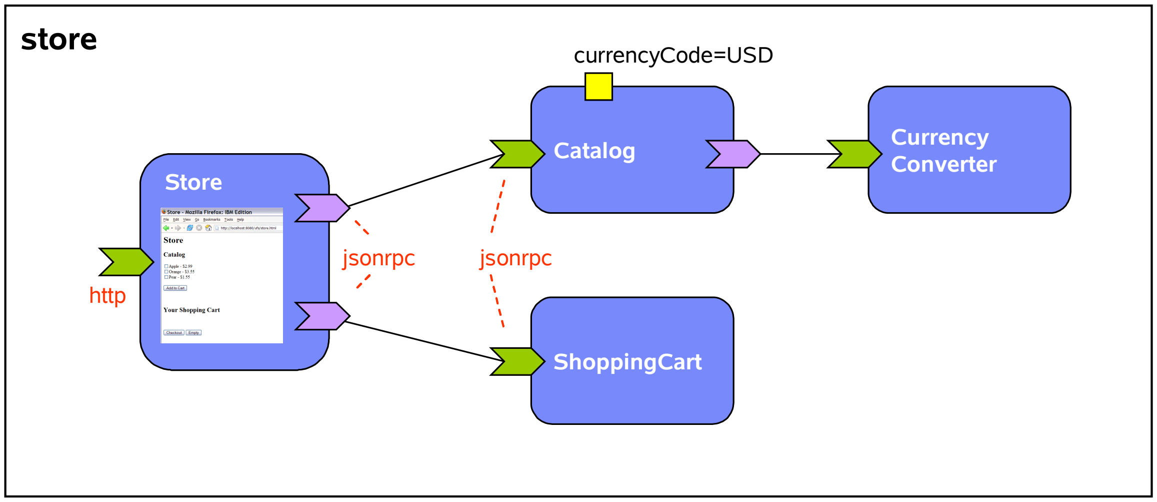
Select the location (folder, JAR or zip) containing the source for 'tuscany-sca-all-0.91-incubating.jar':

Location path: C:/tuscany/apache-tuscany-sca-0.91-incubating-src.zip

Select ***OK*** to complete this and the Preferences dialog. You are done with the Tuscany setup for Eclipse.

Create your 1st Composite Service Application

The following shows the composition diagram for the composite service application you are about to create.



The composite service application you will create is a composition of four services. The composed service provided is that of an on-line store.

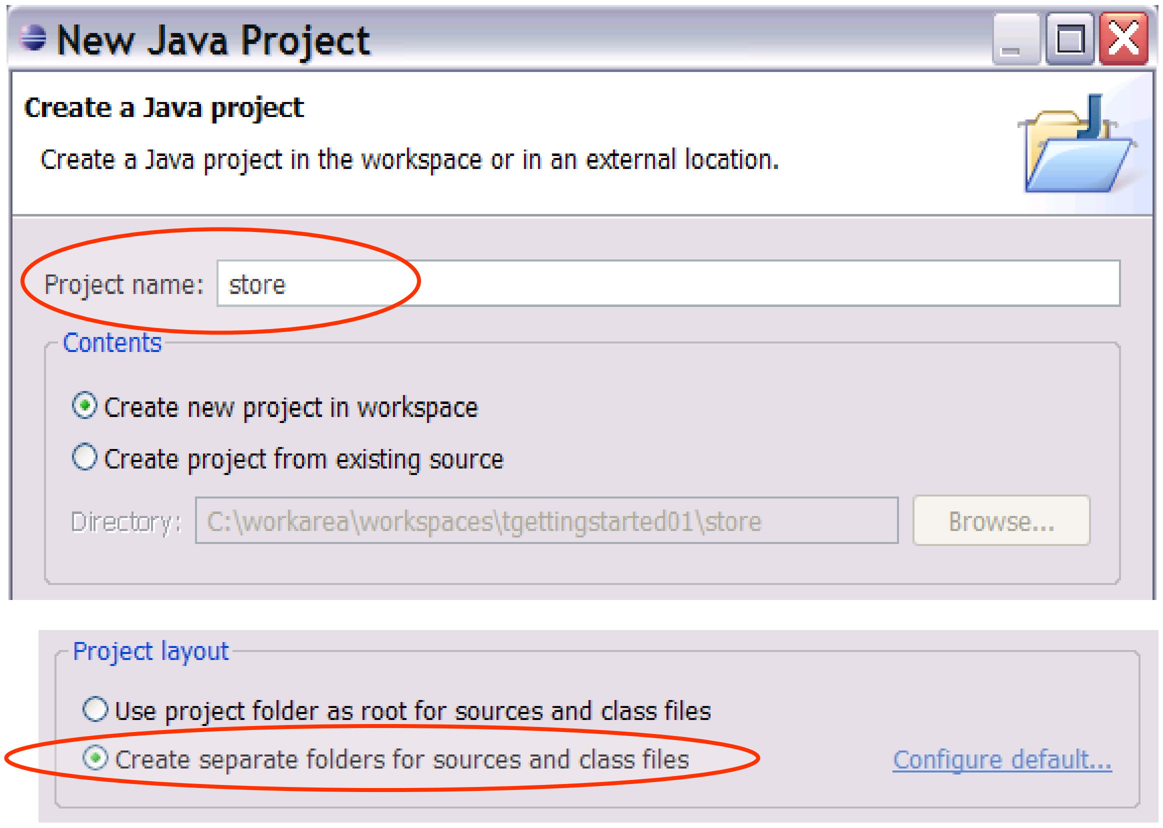
There is a Catalog service which you can ask for catalog items, and depending on its currency code property configuration it will provide the item prices in USD or EUR. The Catalog service is not doing the currency conversion itself it references a CurrencyConverter service to do that task. Then there is the ShoppingCart service into which items chosen from the catalog can be added. Both the Catalog and the ShoppingCart service are bound with a JSONRPC binding. Finally there is the Store user facing service that provides the browser based user interface of the store. The Store service makes use of the Catalog and ShoppingCart service using the JSONRPC binding.

Create a Java Project For Composite Service

In this step you create a Java Project in Eclipse to hold the composite service application.

1. Click on the ***New Java Project*** button in the toolbar to launch the project creation dialog.

2. Enter "store" as the ***Project name***, and for ***Project Layout*** select ***Create separate folders for sources and class files***.


New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Project name: store

Contents

[✓ Create new project in workspace]

Create project from existing source

Directory: C:\workarea\workspaces\tgettingstarted01\store

Browse...

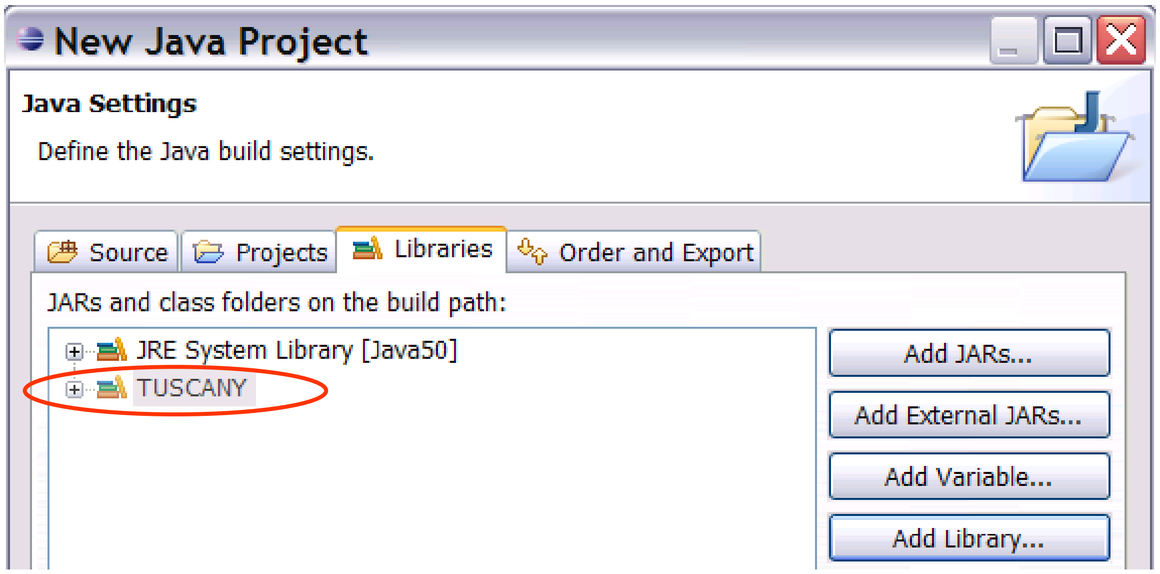
Project layout

Use project folder as root for sources and class files

[✓ Create separate folders for sources and class files]

Configure default...

Hit the ***Next*** button, and on the following page go to the ***Libraries*** tab. Use the ***Add Library*** button on the right to add the TUSCANY user library to the project.


New Java Project

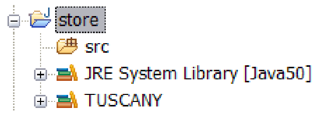
Java Settings

Define the Java build settings.

JARs and class folders on the build path:

- JRE System Library [Java50]
- TUSCANY

Hit the ***Finish*** button to complete the ***New Java Project*** dialog and complete creation of the "store" java project.

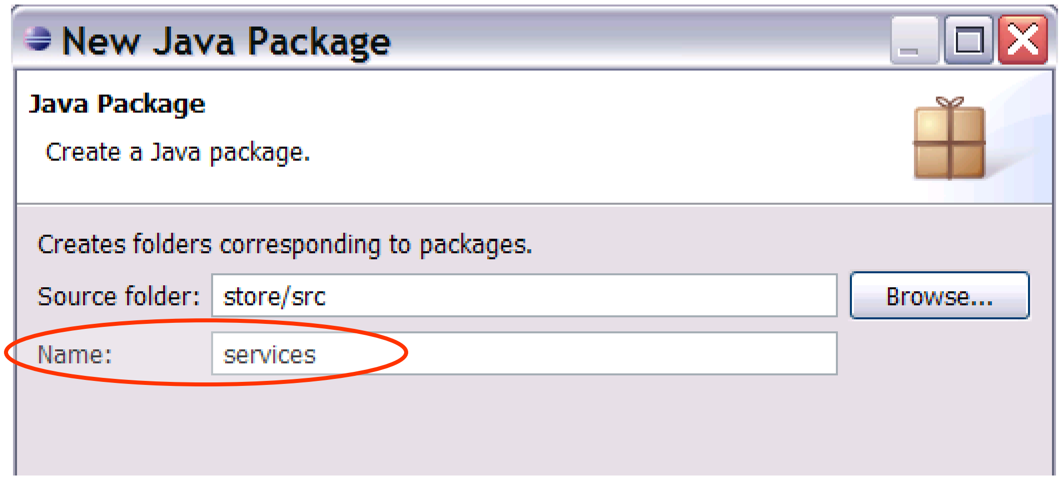

```
store
  src
  JRE System Library [Java50]
  TUSCANY
```

Construct Services

First you create two package folders into which later in this step you place service implementations.

Select the "store" project and click on the ***New Java Package*** button in the toolbar to launch the package creation dialog.

Next you enter "services" as the package ***Name***, and press the ***Finish*** button to complete the dialog.


New Java Package

Java Package

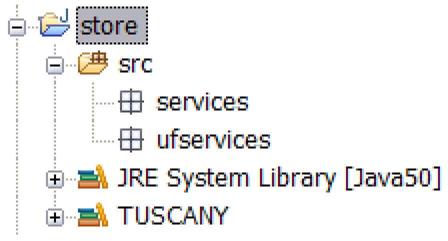
Create a Java package.

Creates folders corresponding to packages.

Source folder: store/src

Name: services

Repeat the previous step to create another package named “ufservices”. The store project now should look as follows.



```
store
  src
    services
    ufservices
  JRE System Library [Java50]
  TUSCANY
```

In this exercise you will place in the “services” package the regular services, and in the “ufservices” package the user facing services of the composite service application you create.

Catalog

In this step you create the Catalog service interface and implementation.

Select the “services” package. Next you click on the dropdown arrow next to the ***New Java Class*** button and select the ***New Java Interface*** option from the dropdown list. In the dialog enter “Catalog” as the ***Name*** of the interface and select the Finish button to complete the dialog.

The Java editor will open on the new created Java interface. Replace the content of the editor by ***copy-paste*** of the following Java interface code snippet.

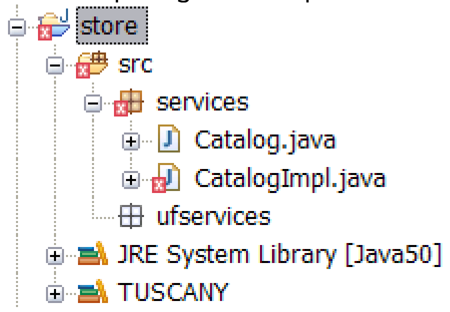
```java
package services;

import org.osoa.sca.annotations.Remotable;

@Remotable
public interface Catalog {

        String[] get();
}
```

Select the “services” package again. Select the ***New Java Class*** button . In the dialog enter “CatalogImpl” as the ***Name*** of the class, add “Catalog” as the interface this class implements, and then select ***Finish*** to complete the dialog.

The Java editor will open on the new created Java class. Replace the content of the editor by ***copy-paste*** of the following Java class code snippet.

```java
package services;

import java.util.ArrayList;
import java.util.List;

import org.osoa.sca.annotations.Init;
import org.osoa.sca.annotations.Property;
import org.osoa.sca.annotations.Reference;

public class CatalogImpl implements Catalog {

        @Property
        public String currencyCode = "USD";
        @Reference
        public CurrencyConverter currencyConverter;

        private List<String> catalog = new ArrayList<String>();
        @Init
        public void init() {
                String currencySymbol = currencyConverter.getCurrencySymbol(currencyCode);
                catalog.add("Apple  - " + currencySymbol +
                        currencyConverter.getConversion("USD", currencyCode, 2.99f));
                catalog.add("Orange - " + currencySymbol +
                        currencyConverter.getConversion("USD", currencyCode, 3.55f));
                catalog.add("Pear   - " + currencySymbol +
                        currencyConverter.getConversion("USD", currencyCode, 1.55f));
        }

        public String[] get() {
                String[] catalogArray = new String[catalog.size()];
                catalog.toArray(catalogArray);
                return catalogArray;
        }
}
```

After completing these steps the content of the “store” project will look as follows.



Note: CatalogImpl is red x’ed because it makes use of the CurrencyConverter interface that we have not implemented yet.

CurrencyConverter

In this step you create the CurrencyConverter service interface and implementation.

You follow the same steps that you learned previously to create the interface and implementation.

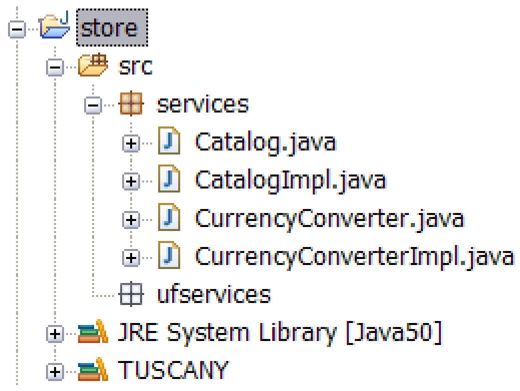
First create a Java interface in the “services” package named “CurrencyConverter” and ***copy-paste*** the following Java interface code snippet into it.

```
package services;

import org.osoa.sca.annotations.Remotable;

@Remotable
public interface CurrencyConverter {

        public float getConversion(String fromCurrenycCode,
                                      String toCurrencyCode, float amount);
        public String getCurrencySymbol(String currencyCode);
}
```

Next create a Java class in the “services” package named “CurrencyConverterImpl” and ***copy-paste*** the following Java class code snippet into it.

```
package services;

public class CurrencyConverterImpl implements CurrencyConverter {

        public float getConversion(String fromCurrencyCode,
                                      String toCurrencyCode, float amount) {
                if (toCurrencyCode.equals("USD"))
                    return amount;
                else
                if (toCurrencyCode.equals("EUR"))
                    return amount*0.7256f;
                return 0;
            }

        public String getCurrencySymbol(String currencyCode) {
                if (currencyCode.equals("USD"))
                    return "$";
                else
                if (currencyCode.equals("EUR"))
                    return "€";
                return "?";
        }
}
```

After completing these steps the content of the “store” project will look as follows.



ShoppingCart

In this step you create the ShoppingCart service interface and implementation.

You follow the same steps that you learned previously to create the interface and implementation.

First create a Java interface in the “services” package named “ShoppingCart” and ***copy-paste*** the following Java interface code snippet into it.

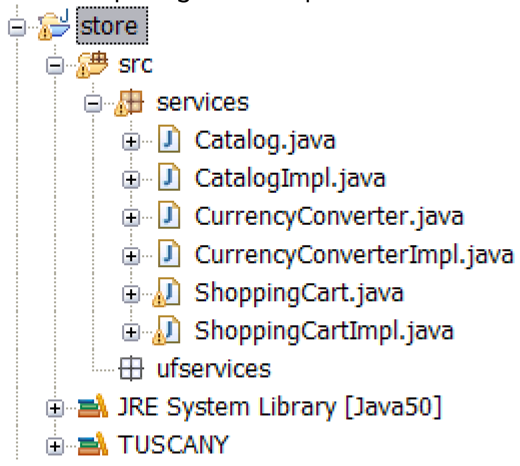
```
package services;

import org.osoa.sca.annotations.Conversational;
import org.osoa.sca.annotations.Remotable;

@Remotable
//@Conversational
public interface ShoppingCart {

        void add(String[] items);
        void empty();
        String[] get();
        String getTotal();
}
```

Next create a Java class in the “services” package named “ShoppingCartImpl” and ***copy-paste*** the following Java class code snippet into it.

```
package services;

import java.util.ArrayList;
import java.util.List;

import org.osoa.sca.annotations.Scope;

//@Scope("CONVERSATION")
public class ShoppingCartImpl implements ShoppingCart {

        // the following needs to change to instance var ones conversation scope works
        private static List<String> cart = new ArrayList<String>();

        public void add(String[] items) {
                for (String item : items) cart.add(item);
        }

        public void empty() {
                cart.clear();
        }

        public String[] get() {
                String[] cartArray = new String[cart.size()];
                cart.toArray(cartArray);
                return cartArray;
        }

        public String getTotal() {
                float total = 0;
                String currencySymbol = "";
                if (!cart.isEmpty()) currencySymbol =
cart.get(0).substring(cart.get(0).indexOf("-")+2, cart.get(0).indexOf("-")+3);
                for (String item : cart) total +=
Float.valueOf(item.substring(item.indexOf("-")+3));
                return currencySymbol + String.valueOf(total);
        }
}
```

Note: Since the Tuscany conversational support is not ready yet the cart is realized through a hack. The cart field is defined as static.

After completing these steps the content of the "store" project will look as follows.



Store

In this step you create the user facing Store service that will run in a Web browser and provide the user interface to the other services you created.

Select the “ufservices” package. ***Right click*** to get the context menu, select ***New***, and then ***File***. In the ***New File*** dialog enter “store.html” for the ***File name***, and then select ***Finish*** to complete the dialog.

The Text editor will open on the new created html file. Replace the content of the editor by ***copy-paste*** of the following html snippet.

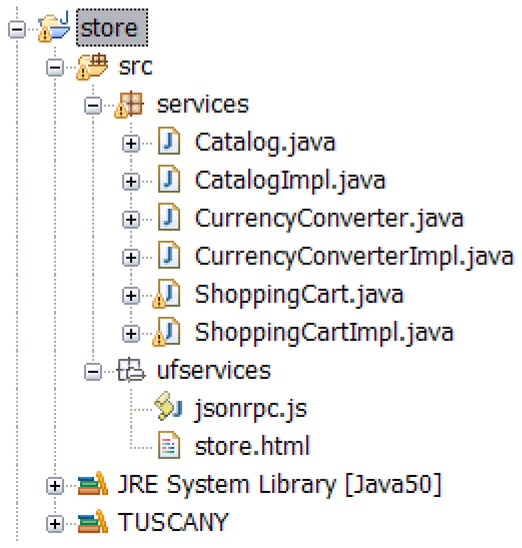
```
<html>
<head>
        <title>Store</TITLE>

        <script type="text/javascript" src="jsonrpc.js"></script>
        <script language="JavaScript">
                catalog = (new JSONRpcClient("../SCADomain/Catalog")).Catalog;
                shoppingCart = (new
JSONRpcClient("../SCADomain/ShoppingCart")).ShoppingCart;

                function catalog_getResponse(items) {
                        var catalog = "";
                        for (var i=0; i<items.length; i++)
                                catalog += '<input name="items" type="checkbox" value="' +
items[i] + '">' + items[i]+ ' <br>';
                        document.getElementById('catalog').innerHTML=catalog;
                }
                function shoppingCart_addResponse(result) {
                        shoppingCart.get(shoppingCart_getResponse);
                        shoppingCart.getTotal(shoppingCart_getTotalResponse);
                }
                function shoppingCart_getResponse(items) {
                        var shoppingCart = "";
                        for (var i=0; i<items.length; i++)
                                shoppingCart += items[i] + ' <br>';
                        document.getElementById('shoppingCart').innerHTML=shoppingCart;
                }
                function shoppingCart_getTotalResponse(total) {
                        document.getElementById('total').innerHTML="Total : " + total;
                }

                function addToCart() {
                        var items  = document.catalogForm.items;
                        var addItems = new Array();
                        var j = 0;
                        for (var i=0; i<items.length; i++)
                                if (items[i].checked) {
                                        addItems[j++] = items[i].value;
                                        items[i].checked = false;
                                }
                        shoppingCart.add(addItems, shoppingCart_addResponse);
                }
                function checkout() {
                        document.getElementById('store').innerHTML='<h2>Thanks for Shopping
With Us!</h2>'+
                                       '<h2>Your Order</h2>'+
                                       '<form name="orderForm" action="/ufs/store.html">'+

        document.getElementById('shoppingCart').innerHTML+
                                               '<br>'+
                                               document.getElementById('total').innerHTML+
                                               '<br>'+
                                               '<br>'+
                                               '<input type="submit" value="Continue
Shopping">'+
                                       '</form>';
                        shoppingCart.empty();
                }
                function empty() {
                        shoppingCart.empty();
                        document.getElementById('shoppingCart').innerHTML="";
                        document.getElementById('total').innerHTML="";
                }

                catalog.get(catalog_getResponse);
    </script>

        <link rel="stylesheet" type="text/css" href="style.css" />
</head>

<body>
<h1>Store</h1>
  <div id="store">
        <h2>Catalog</h2>
        <form name="catalogForm">
                <div id="catalog" ></div>
                <br>
                <input type="button" onClick="addToCart()"  value="Add to Cart">
        </form>

        <br>

        <h2>Your Shopping Cart</h2>
        <form name="shoppingCartForm">
                <div id="shoppingCart"></div>
                <br>
                <div id="total"></div>
                <br>
                <input type="button" onClick="checkout()" value="Checkout">
                <input type="button" onClick="empty()" value="Empty">
        </form>
  </div>
</body>
</html>
```

Next select the “ufservices” package again. ***Right click*** to get the context menu, select ***New***, and then ***File***. In the ***New File*** dialog enter “jsonrpc.js” for the ***File name***, and then select ***Finish*** to complete the dialog.

The Text editor will open on the new created javascript file. Replace the content of the editor by ***copy-paste*** of the javascript snippet you find here:
https://svn.apache.org/repos/asf/incubator/tuscany/java/sca/modules/binding-jsonrpc/src/main/resources/org/apache/tuscany/sca/binding/jsonrpc/jsonrpc.js

Note: That we have to have the jsonrpc.js local in our project is only temporary, so this step will be removed in the future.

After completing these steps the content of the “store” project will look as follows.



Compose Services

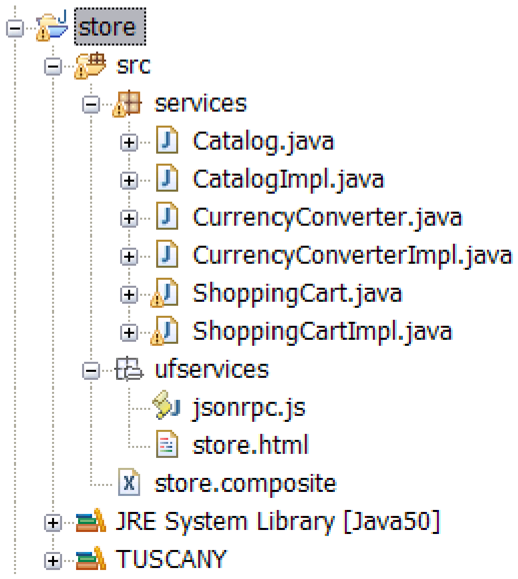
Now that you have all the required service implementations you compose them together to provide the store composite service. The composition is stored in a .composite file.

Select the “src” folder of the “store” project. ***Right click*** to get the context menu, select ***New***, and then ***File***. In the ***New File*** dialog enter “store.composite” for the ***File name***, and then select ***Finish*** to complete the dialog.

The Text editor will open on the new created composite file. Replace the content of the editor by ***copy-paste*** of the following composite snippet.

```
<?xml version="1.0" encoding="UTF-8"?>
<composite      xmlns="http://www.osoa.org/xmlns/sca/1.0" xmlns:s="http://store"
                name="store">

    <service name="ufs" promote="ufs">
        <binding.resource/>
    </service>
    <component name="ufs">
        <implementation.resource location="ufservices"/>
    </component>

    <service name="Catalog" promote="Catalog">
        <binding.jsonrpc/>
    </service>
    <component name="Catalog">
        <implementation.java class="services.CatalogImpl"/>
        <property name="currencyCode">USD</property>
        <reference name="currencyConverter" target="CurrencyConverter"/>
    </component>

    <service name="ShoppingCart" promote="ShoppingCart">
        <binding.jsonrpc/>
    </service>
    <component name="ShoppingCart">
        <implementation.java class="services.ShoppingCartImpl"/>
    </component>

    <component name="CurrencyConverter">
        <implementation.java class="services.CurrencyConverterImpl"/>
    </component>

</composite>
```

After completing these steps the content of the “store” project will look as follows.



Launch Services

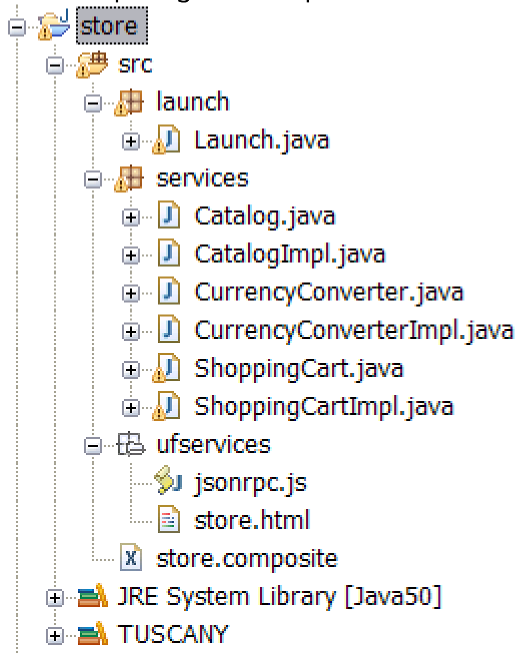
In this step you create the code to launch the Tuscany runtime with the new store composite service you created.
Select the “store” project and click on the ***New Java Package*** button in the toolbar to start the package creation dialog. Use the dialog to create a new package named “launch”.

Select the “launch” package. Select the ***New Java Class*** button . In the dialog enter “Launch” as the ***Name*** of the class, ***check the checkbox for creating a main method stub***, and then select ***Finish*** to complete the dialog.

The Java editor will open on the new created Java class. Replace the content of the editor by ***copy-paste*** of the following Java class code snippet.

```java
package launch;

import org.apache.tuscany.sca.host.embedded.SCADomain;

public class Launch {
    public static void main(String[] args) throws Exception {

        System.out.println("Starting ...");
        SCADomain scaDomain = SCADomain.newInstance("store.composite");
        System.out.println("store.composite ready for big business !!!");
        System.out.println();
    }
}
```

After completing these steps the content of the “store” project will look as follows.



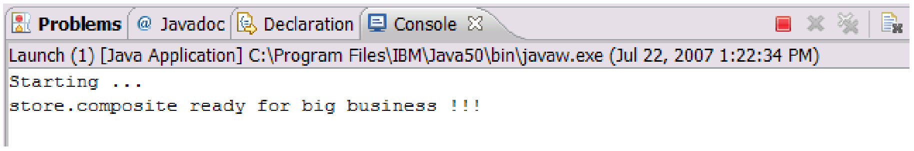
Congratulations you completed your 1st composite service applications, now its time to run it!

Use Services

In this step you launch and use the store composite service application you created.

First select the “Launch” class in the “launch” package of your “store” project. ***Right click*** to get the context menu, select ***Run As***, and then ***Java application***. The Tuscany runtime will start up adding the store composition to its domain.

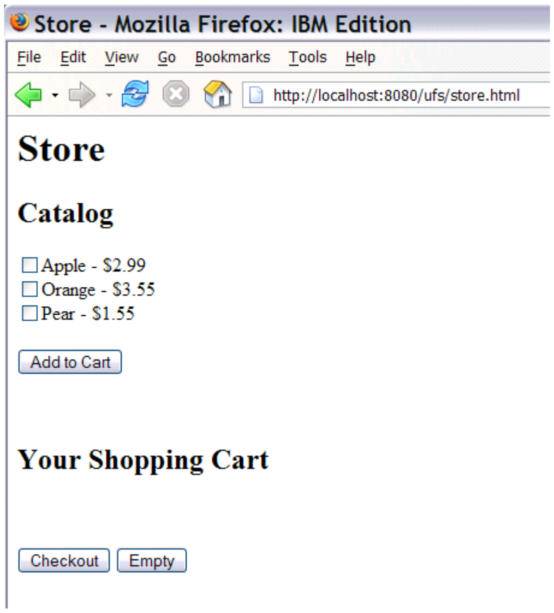
The Eclipse console will show the following messages.



```
Launch (1) [Java Application] C:\Program Files\IBM\Java50\bin\javaw.exe (Jul 22, 2007 1:22:34 PM)
Starting ...
store.composite ready for big business !!!
```

Next Launch your Web browser and enter the following address:
http://localhost:8080/ufs/store.html

You get to the Store user facing service of the composite service application.


Store - Mozilla Firefox: IBM Edition
File Edit View Go Bookmarks Tools Help
http://localhost:8080/ufs/store.html

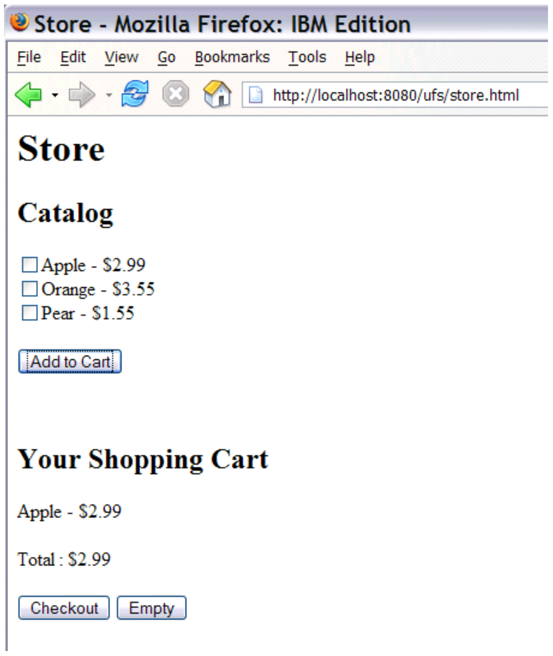
Store

Catalog

☐ Apple - $2.99
☐ Orange - $3.55
☐ Pear - $1.55

Your Shopping Cart

You can select items from the Catalog and add them to your Shopping Cart.


Store - Mozilla Firefox: IBM Edition
File Edit View Go Bookmarks Tools Help
http://localhost:8080/ufs/store.html

Store

Catalog

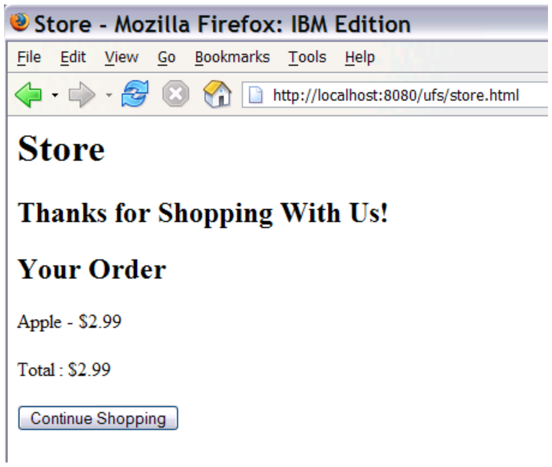
☐ Apple - $2.99
☐ Orange - $3.55
☐ Pear - $1.55

Your Shopping Cart

Apple - $2.99

Total : $2.99

And you can Checkout to complete your order.


Store - Mozilla Firefox: IBM Edition
File Edit View Go Bookmarks Tools Help
http://localhost:8080/ufs/store.html

Store

Thanks for Shopping With Us!

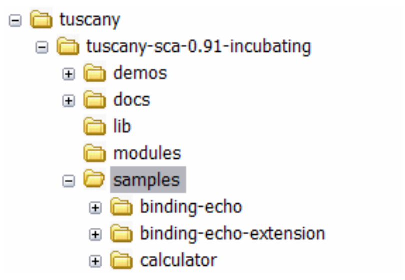
Your Order

Apple - $2.99

Total : $2.99

Explore the Samples from the Tuscany Distribution

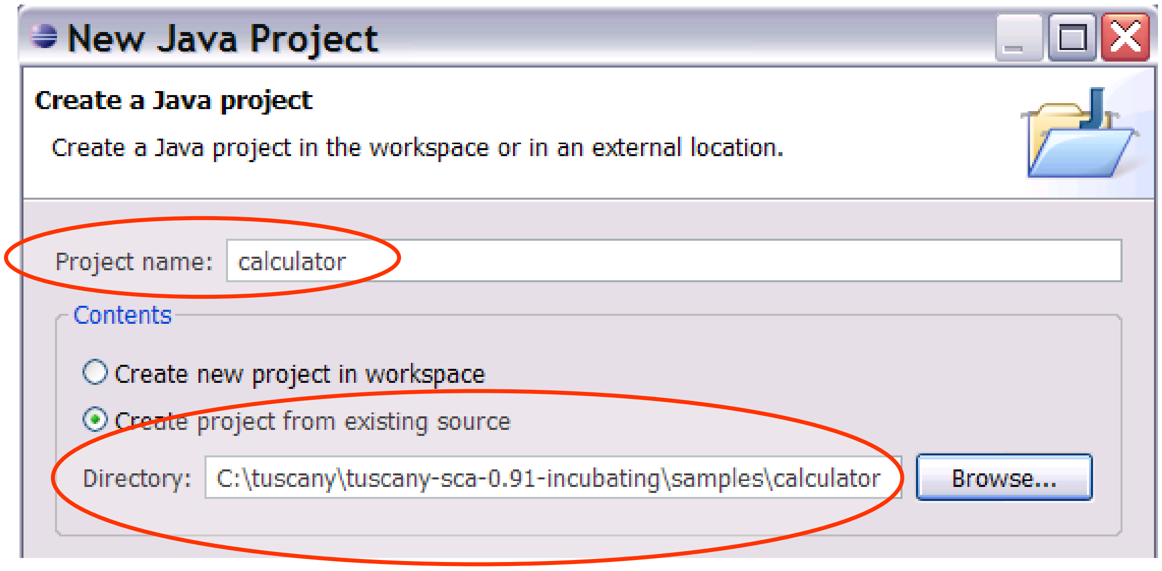
The sample folder of the Tuscany distribution provides a rich set of samples ready for you to explore.



tuscany
tuscany-sca-0.91-incubating
demos
docs
lib
modules
samples
binding-echo
binding-echo-extension
calculator

...

In Eclipse create a ***New Java Project***, specify the ***project name***, select ***Create project from existing source***, and ***specify the folder that contains the sample source***.



New Java Project

Create a Java project

Create a Java project in the workspace or in an external location.

Project name: calculator

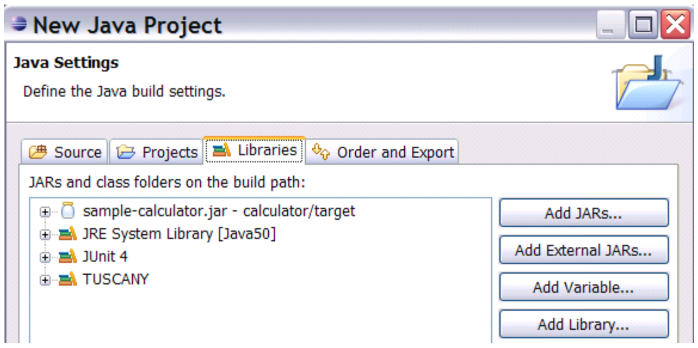
Contents

Create new project in workspace

Create project from existing source (selected)

Directory: C:\tuscany\tuscany-sca-0.91-incubating\samples\calculator

Use ***Next*** to get to the next page in the New Java Project dialog. There go to the ***Libraries*** tab, use the "***Add Library…*** pushbutton to add the ***JUnit*** library and the user library ***TUSCANY***.



New Java Project

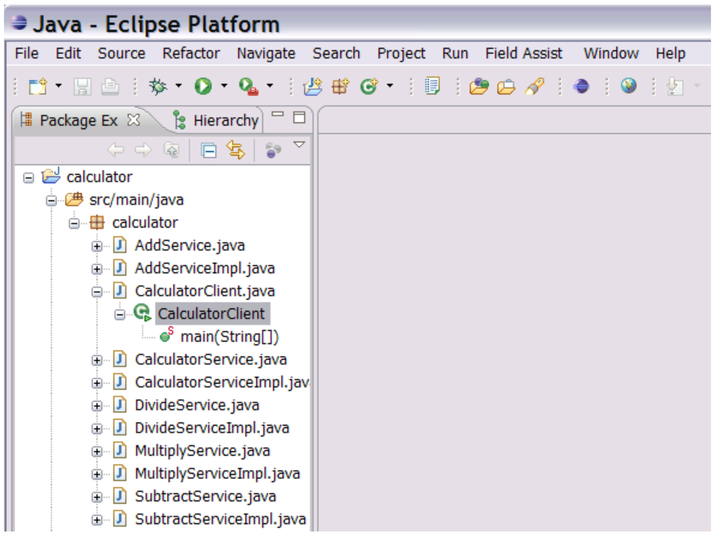
Java Settings

Define the Java build settings.

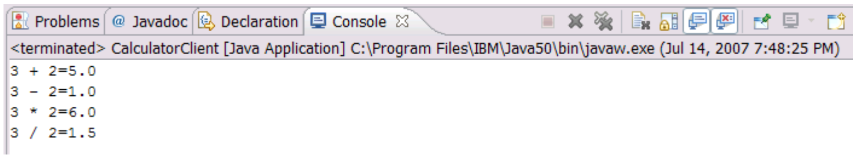
JARs and class folders on the build path:

sample-calculator.jar - calculator/target
JRE System Library [Java50]
JUnit 4
TUSCANY

Finish the New Java Project dialog. You now have the sample project available in the Eclipse workbench.



For the calculator sample that we've chosen go to its ***CalculatorClient*** class and select ***Run As –> Java Application***. You will see the following output in the console.



```
<terminated> CalculatorClient [Java Application] C:\Program Files\IBM\Java50\bin\javaw.exe (Jul 14, 2007 7:48:25 PM)
3 + 2=5.0
3 - 2=1.0
3 * 2=6.0
3 / 2=1.5
```