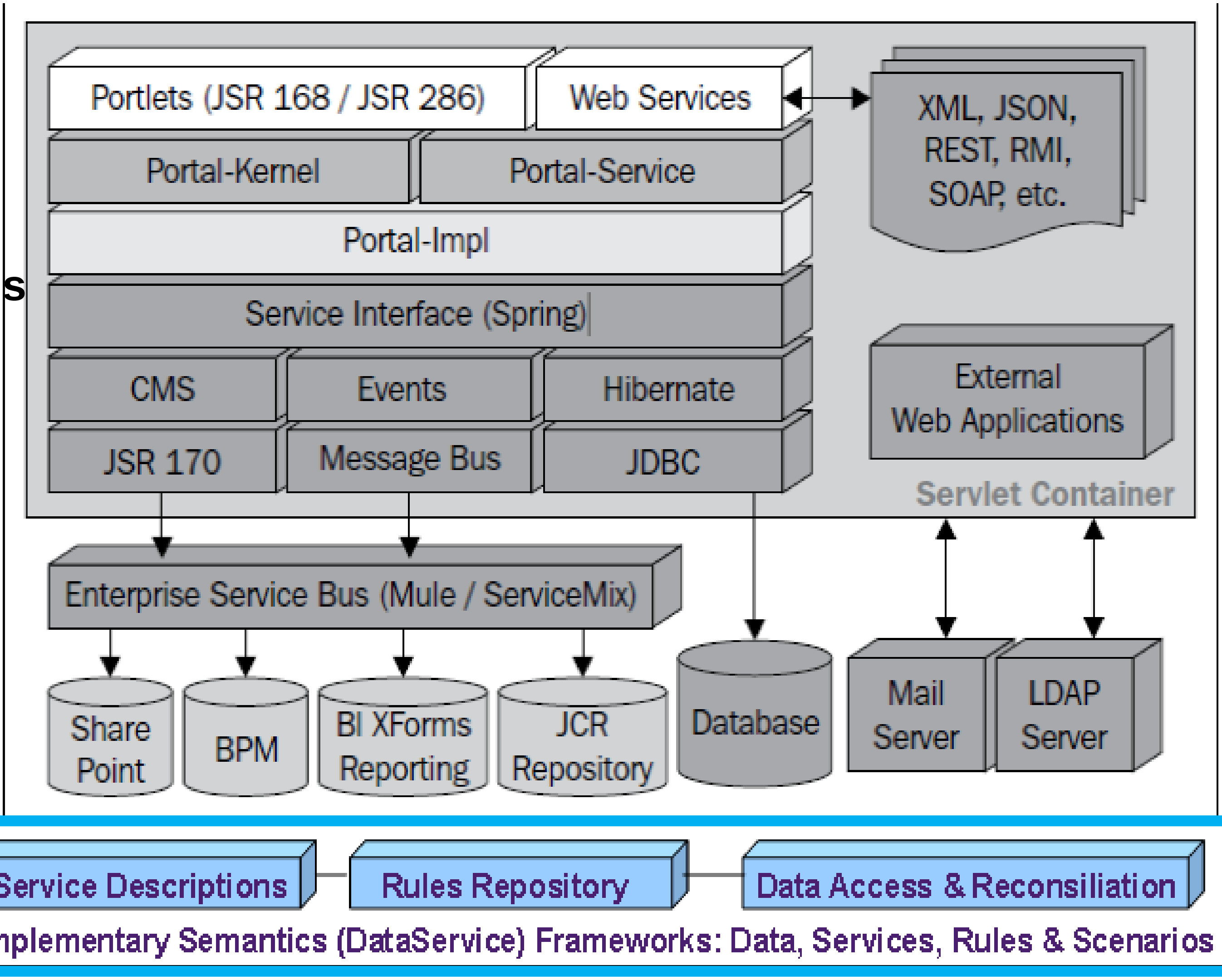
Liferay Customization Hints

- Hooks
- Themes
- Services
- Portlet Intercommunications


Portlets (JSR 168 / JSR 286)
Web Services
XML, JSON, REST, RMI, SOAP, etc.
Portal-Kernel
Portal-Service
Portal-Impl
Service Interface (Spring)
CMS
Events
Hibernate
JSR 170
Message Bus
JDBC
External Web Applications
Servlet Container
Enterprise Service Bus (Mule / ServiceMix)
Share Point
BPM
BI XForms Reporting
JCR Repository
Database
Mail Server
LDAP Server
Service Descriptions
Rules Repository
Data Access & Reconsiliation
Complementary Semantics (DataService) Frameworks: Data, Services, Rules & Scenarios

Semantic Enhancements

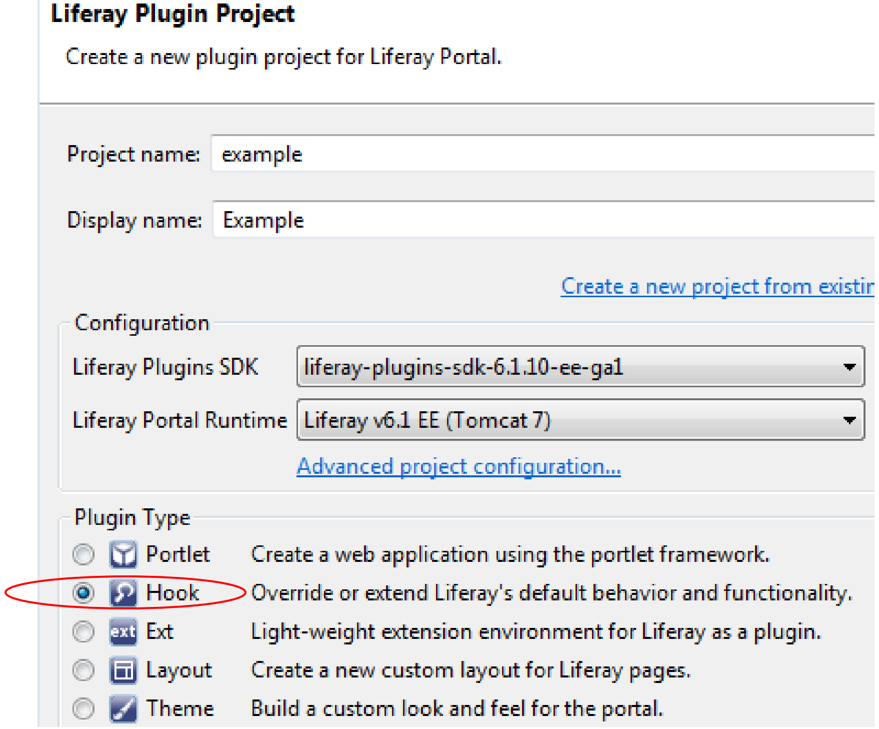
Business Architecture Sandbox for Enterprise (BASE)

Hooks

There are several ways

to customize a Liferay web site

- **It's possible to use Ext plugins** to override original sources
  - I do not recommend this way:
  - this might not work with the next Liferay release where source code might be different



Liferay Plugin Project

Create a new plugin project for Liferay Portal.

Project name:	example
Display name:	Example

Create a new project from existir

Configuration

Liferay Plugins SDK	liferay-plugins-sdk-6.1.10-ee-ga1
Liferay Portal Runtime	Liferay v6.1 EE (Tomcat 7)

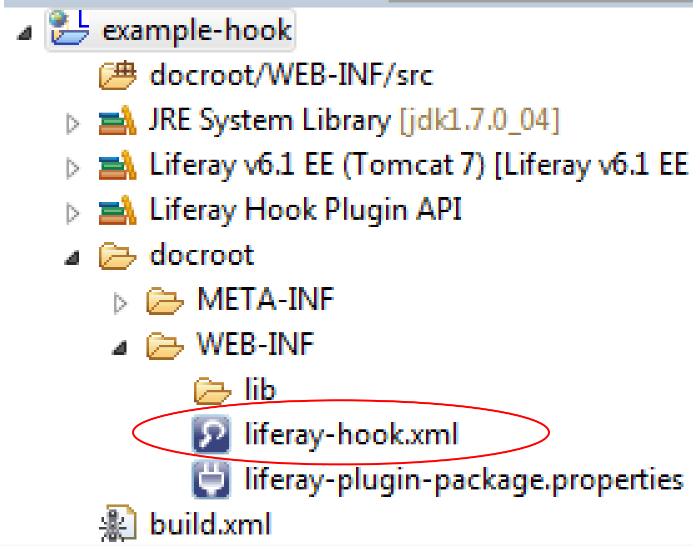
Advanced project configuration...

Plugin Type

Portlet	Create a web application using the portlet framework.
[✓ Hook]	Override or extend Liferay's default behavior and functionality.
Ext	Light-weight extension environment for Liferay as a plugin.
Layout	Create a new custom layout for Liferay pages.
Theme	Build a custom look and feel for the portal.

- **The better way is to use Liferay Hooks**
  - Liferay Hooks do not touch original sources
  - Hooked to the properties, they allow developers to provide alternative behavior (methods) related to these properties
  - Can be easily created with the Liferay IDE as Hooks Project
  - Stored in the Plugins SDK's hooks directory.

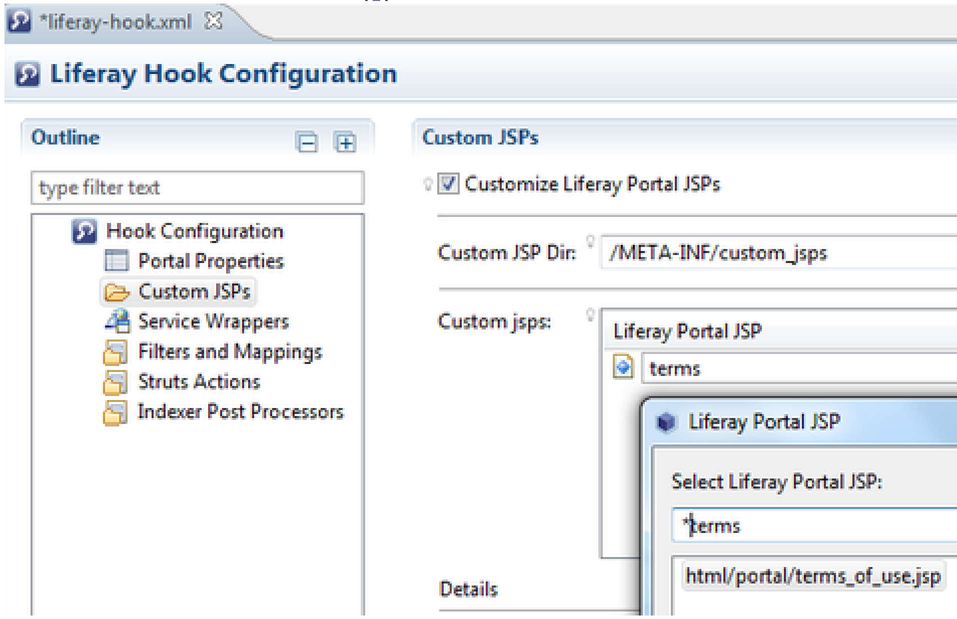
Hook Project
*Example: modifying portal's *terms_of_use.jsp*


example-hook
docroot/WEB-INF/src
JRE System Library [jdk1.7.0_04]
Liferay v6.1 EE (Tomcat 7) [Liferay v6.1 EE
Liferay Hook Plugin API
docroot
META-INF
WEB-INF
lib
liferay-hook.xml
liferay-plugin-package.properties
build.xml

- Open the hook's project file:

docroot/META-INF/custom_jsps/html/portal/terms_of_use.jsp

and modify as necessary.


*liferay-hook.xml

Liferay Hook Configuration

Outline

type filter text

Hook Configuration
Portal Properties
Custom JSPs
Service Wrappers
Filters and Mappings
Struts Actions
Indexer Post Processors

Custom JSPs

☑ Customize Liferay Portal JSPs

Custom JSP Dir: /META-INF/custom_jsps

Custom jsps: Liferay Portal JSP

terms

Liferay Portal JSP

Select Liferay Portal JSP:

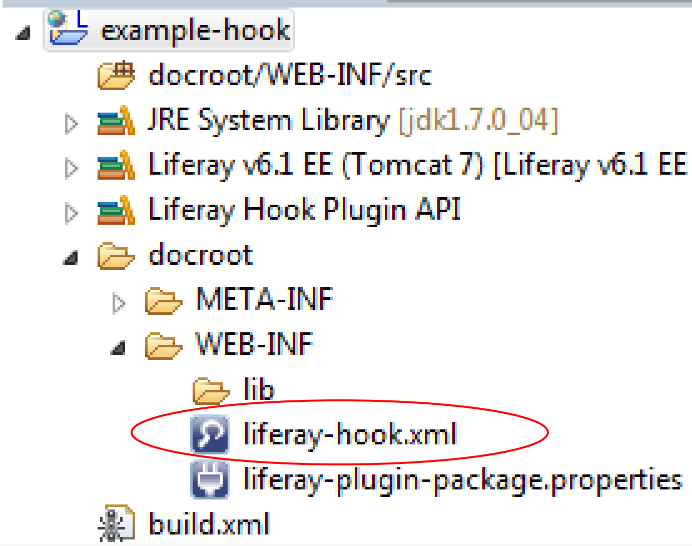
*terms

html/portal/terms_of_use.jsp

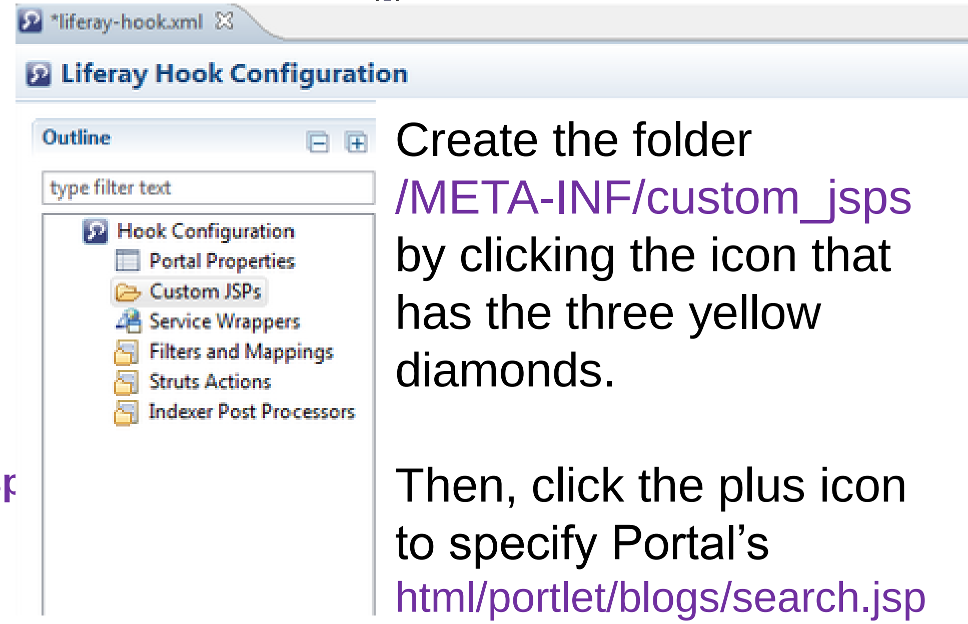
Details

* Another way to modify the *Terms of Use* is setting properties in portal-ext.properties.

Hook Project *Example: Add more to existing JSP html/portlet/blogs/search.jsp


example-hook
docroot/WEB-INF/src
JRE System Library [jdk1.7.0_04]
Liferay v6.1 EE (Tomcat 7) [Liferay v6.1 EE
Liferay Hook Plugin API
docroot
META-INF
WEB-INF
lib
liferay-hook.xml
liferay-plugin-package.properties
build.xml

Open the liferay-hook.xml and select the Custom JSPs.



Follow the hints on the right and Liferay IDE will copy the Liferay Portal JSP into your project.

Create the folder /META-INF/custom_jsps by clicking the icon that has the three yellow diamonds.

Then, click the plus icon to specify Portal's html/portlet/blogs/search.jsp

Then open this copy-file

docroot/META-INF/custom_jsps/html/blogs/search.jsp

Replace existing code with this:

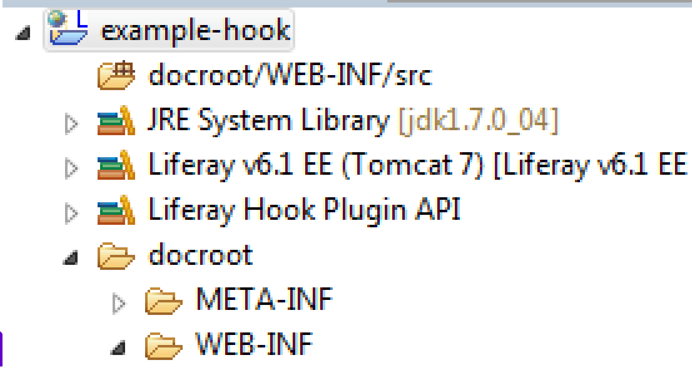
```
<%@ taglib uri="http://liferay.com/tld/util" prefix="liferay-util" %>
<%@ page import="com.liferay.portal.kernel.util.StringUtil" %>
<liferay-util:buffer var="html"> <liferay-util:include page="/html/portlet/blogs/search.portal.jsp" />
</liferay-util:buffer>
<% html = StringUtil.add( html, “Refine the search !", "\n"); %>
<%= html %>
```

Note, that this code introduces dynamic HTML replacement

Reference: Liferay 6.1 Developers Guide

dean@JavaSchool.com

Hook Project *Example: Trigger Custom Login Action


example-hook
docroot/WEB-INF/src
JRE System Library [jdk1.7.0_04]
Liferay v6.1 EE (Tomcat 7) [Liferay v6.1 EE
Liferay Hook Plugin API
docroot
META-INF
WEB-INF

In the WEB-INF directory create the folder **src/com/liferay/sample/hook**

In the sample/hook folder create the file **LoginAction.java**

```
package com.liferay.sample.hook;

import com.liferay.portal.kernel.events.Action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class LoginAction extends Action {
    public void run(HttpServletRequest req, HttpServletResponse res) {
        // the custom login action
    }
}
```

Create the file **WEB-INF/src/portal.properties** with this content:

```
login.events.pre=com.liferay.sample.hook.LoginAction
```

In the file: WEB-INF/liferay-hook.xml find the line: <custom-jsp-dir> and add the line above:

```
<portal-properties>portal.properties</portal-properties>
```

Note, we hooked the login event and instructed to perform the custom action before (pre) the standard one

Reference: [Liferay 6.1 Developers Guide]()

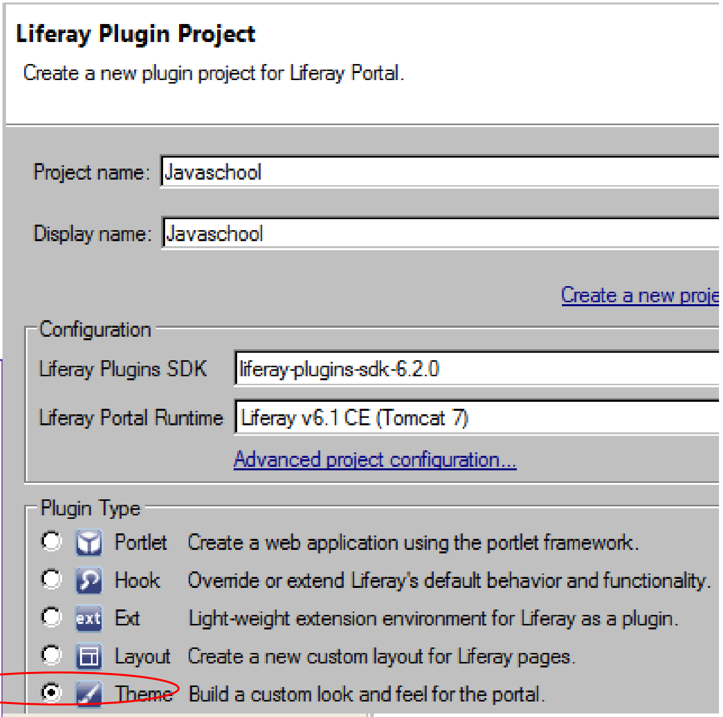
Theme Project Example
Modify the look and feel
With the Javaschool Theme

1. Create the new theme by customizing one of the existing themes.
How:
- Modify the line in the build.xml
- <property name="theme.parent" value="_styled" />
- Replace "_styled" with the selected theme name

2. The theme is built of the files in the following directories:
css, images, js, templates

Modify selected files without touching original source
How:
- In the docroot/_diffs (created by the IDE) create the folders for the selected modified files
- For example, we'd like to modify the portal_normal.vm
- So, we create the directory **templates** under the docroot/_diffs
- Then copy and modify the file portal_normal.vm

3. Redeploy the project (right-mouse-click on the project – **Redeploy**)

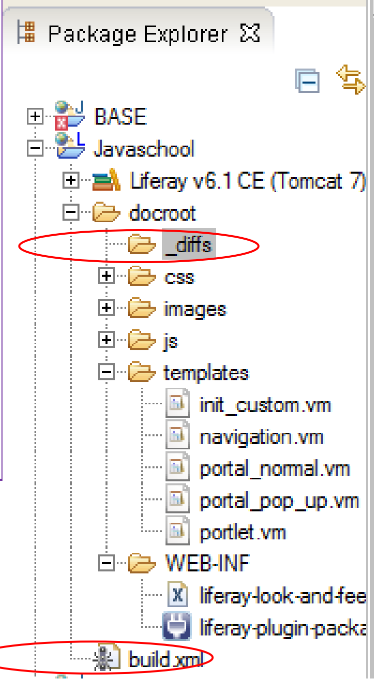

Liferay Plugin Project
Create a new plugin project for Liferay Portal.

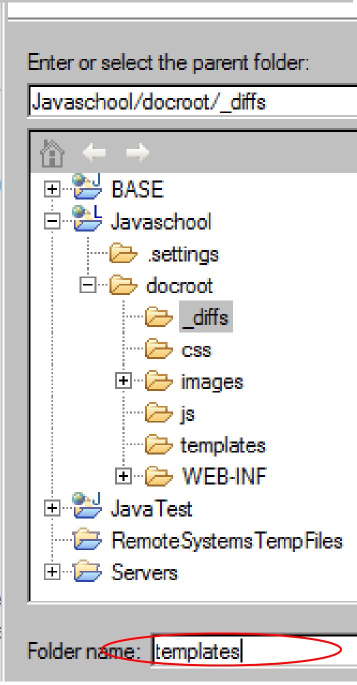
Project name: Javaschool
Display name: Javaschool

Create a new proje

Configuration
Liferay Plugins SDK: liferay-plugins-sdk-6.2.0
Liferay Portal Runtime: Liferay v6.1 CE (Tomcat 7)
Advanced project configuration...

Plugin Type
- Portlet — Create a web application using the portlet framework.
- Hook — Override or extend Liferay's default behavior and functionality.
- Ext — Light-weight extension environment for Liferay as a plugin.
- Layout — Create a new custom layout for Liferay pages.
- [✓ Theme] — Build a custom look and feel for the portal.


```
Package Explorer
BASE
Javaschool
  Liferay v6.1 CE (Tomcat 7)
  docroot
    _diffs
    css
    images
    js
    templates
      init_custom.vm
      navigation.vm
      portal_normal.vm
      portal_pop_up.vm
      portlet.vm
    WEB-INF
      liferay-look-and-fee
      liferay-plugin-packa
  build.xml
```


Enter or select the parent folder:
Javaschool/docroot/_diffs

```
BASE
Javaschool
  .settings
  docroot
    _diffs
    css
    images
    js
    templates
    WEB-INF
JavaTest
RemoteSystemsTempFiles
Servers
```

Folder name: templates

Check out this existing library of themes: https://github.com/liferay/liferay-plugins/tree/master/themes

Services Example

1. Add services to the existing its-portlet project.
How:
- Right mouse on the project – File-New-Liferay Service Buider
- Select existing com.its.services package path
- Provide the namespace (<u>must be a valid keyword, like **Event**</u>)
- Click Finish and Liferay IDE creates WEB-INF/services.xml

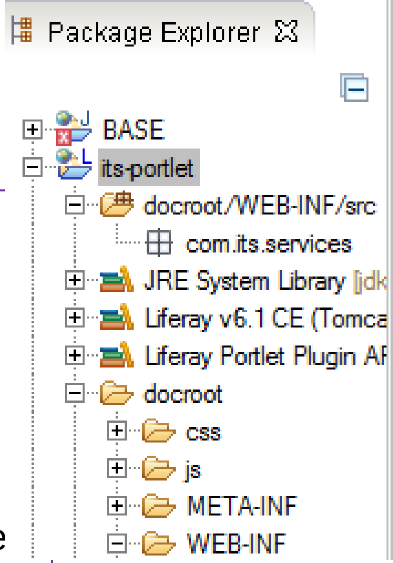
2. Modify the file services.xml via the GUI (on the right) or the source

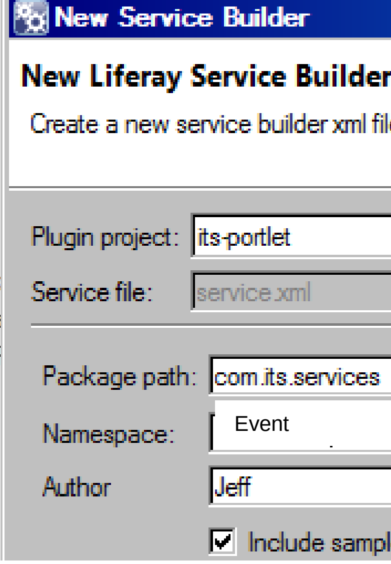
How:
- Use the Entities control to provide data structures

Service Builder uses the service *namespace* in naming the database tables and generates the following SQL scripts under the docroot/WEB-INF/sql folder:

indexes.sql
sequences.sql
tables.sql

3. Create the services (see the next slide)



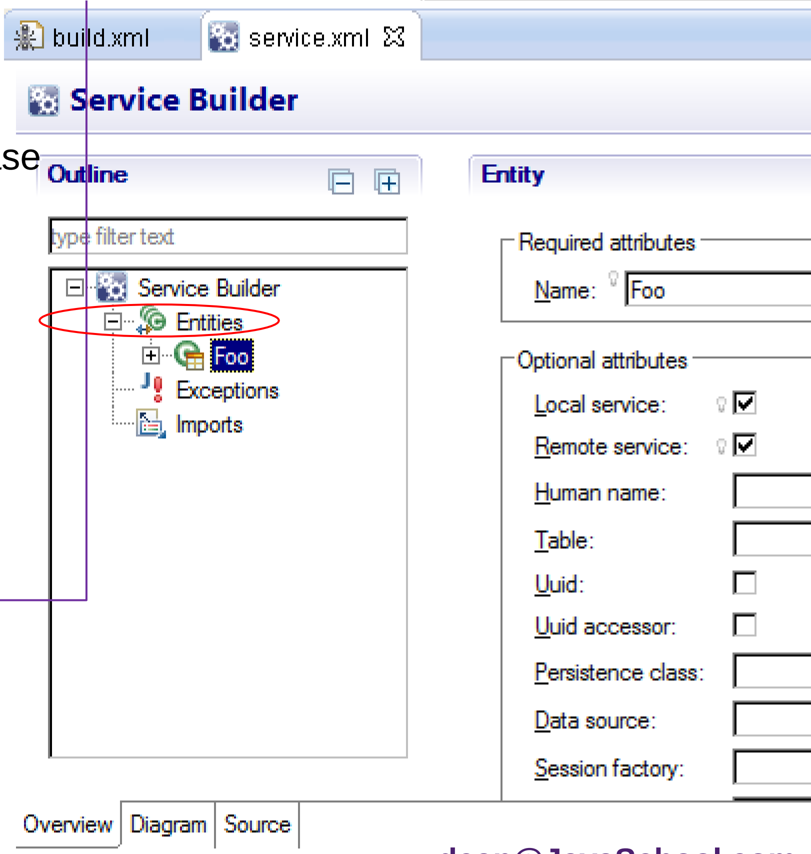

New Service Builder

New Liferay Service Builder

Create a new service builder xml fil

Plugin project:	its-portlet
Service file:	service.xml
Package path:	com.its.services
Namespace:	Event
Author	Jeff

☑ Include sampl



Entity

Required attributes

Name: Foo

Optional attributes

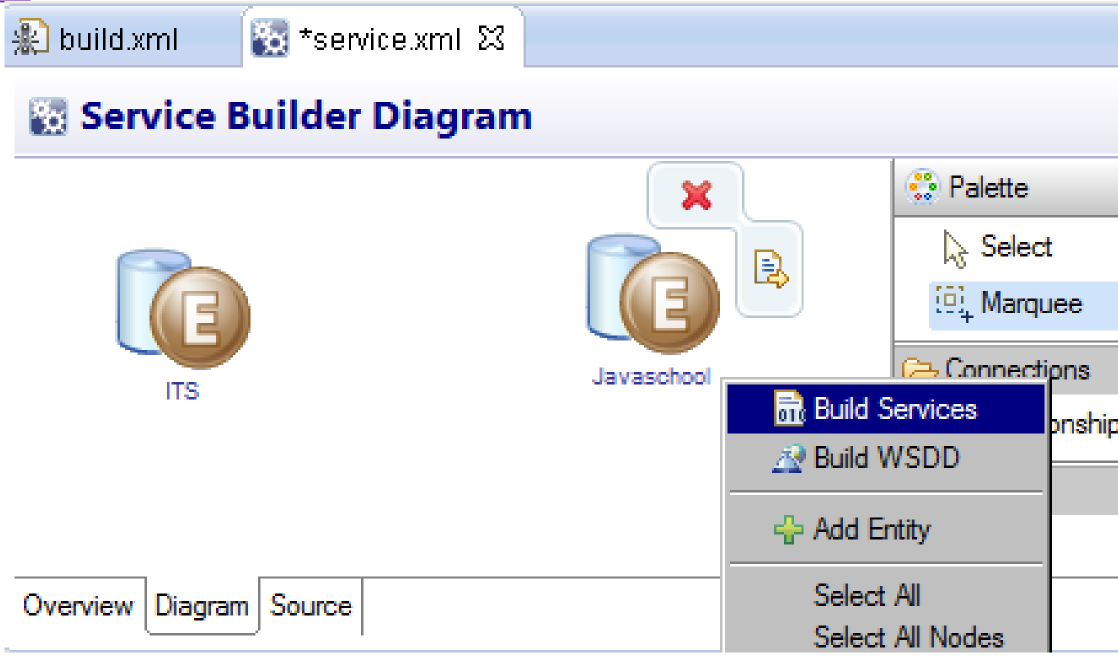
Local service:	☑
Remote service:	☑
Human name:	
Table:	
Uuid:	☐
Uuid accessor:	☐
Persistence class:	
Data source:	
Session factory:	

Services Example Continue With ITS and Javaschool Entities

3. Create **ITS** and **Javaschool** Entities, Open the Diagram tab and select **Build Services** for the **Javaschool** Entity

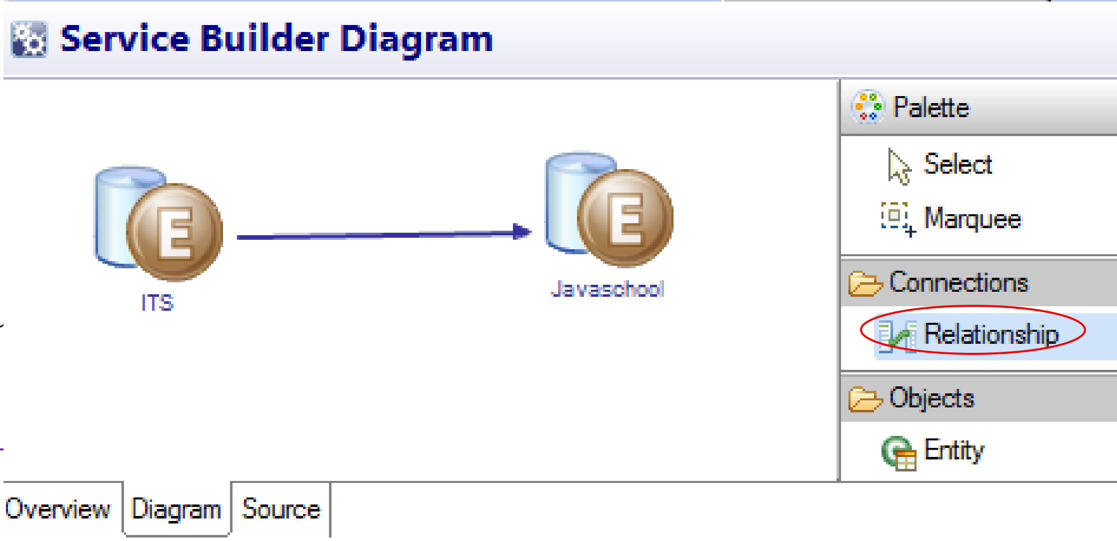
The Liferay IDE will create a set of Java-based services and store them under the liferay-plugins-sdk\portlets\project-name – directory

4. Then select the ***Relationship*** option and connect **ITS** and **Javaschool** Entities. Save the service.xml file.



5. Check the source of the service.xml file and modify as necessary. For the X-service:
Service Builder will create an XLocalService class which is the interface for the local service.
It contains the signatures of every method in XLocalServiceBaseImpl and XLocalServiceImpl.

XLocalServiceBaseImpl contains a few automatically generated methods providing common functionality. Since the XLocalService class is generated, you should never modify it. If you do, your changes will be overwritten the next time you run Service Builder. Instead, all custom code should be placed in XLocalServiceImpl.



For full source example check out the portlets/event-listing-portlet in the https://github.com/liferay/liferay-docs/tree/6.1.x/devGuide/code/devGuide-sdk.

dean@JavaSchool.com

Inter - Portlet Communications

The JSR-286 (Portlet 2.0) specification provides the following ways for sharing data between portlets from ACTION to VIEW phase and VIEW-VIEW Phase.

.

1. Portlet session

2. IPC Mechanisms

Public Render Parameters

Event

Client-Side IPC

3. Cookies

--

1. Portlet session: Liferay provides a mechanism to share session data across WARs.

<u>liferay-portlet.xml</u>

```
<portlet> <private-session-attributes>false</private-session-attributes> </portlet>
```

<u>Set Session in Portlet 1</u>

```
PortletSession session = renderRequest.getPortletSession();
session.setAttribute("sessionValue", some-value ,PortletSession.APPLICATION_SCOPE);
```

<u>Get Session in Portlet 2</u>

```
PortletSession ps = renderRequest.getPortletSession();
String tabNames = (String)ps.getAttribute("sessionValue ",PortletSession.APPLICATION_SCOPE);
```

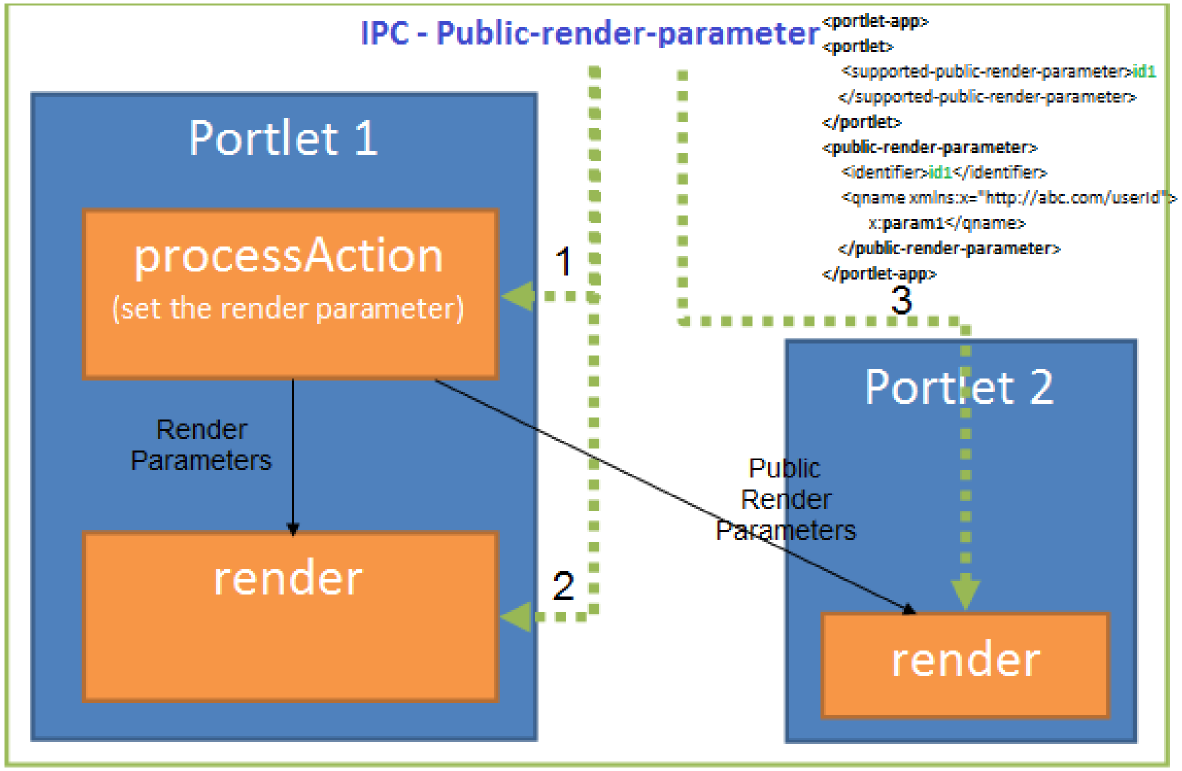
Inter - Portlet Communications 2.1

2. IPC Mechanisms

2.1 Public Render Parameters

2.1.1 Provide in "Sender-Portlet" portlet.xml

```
<portlet-app>
 <portlet>
  <supported-public-render-parameter>id1
  </supported-public-render-parameter>
 </portlet>
 <public-render-parameter>
  <identifier>id1</identifier>
  <qname xmlns:x="http://abc.com/userId">
x:param1</qname>
 </public-render-parameter>
</portlet-app>
```



```
<portlet-app>
<portlet>
  <supported-public-render-parameter>id1
  </supported-public-render-parameter>
</portlet>
<public-render-parameter>
  <identifier>id1</identifier>
  <qname xmlns:x="http://abc.com/userId">
     x:param1</qname>
  </public-render-parameter>
</portlet-app>
```

2.1.2 Set the render parameter in the processAction() method by using **id1** as the key.

```
public void processAction(ActionRequest request,
ActionResponse response) throws IOException,
PortletException { ........
response.setRenderParameter("id1",
“someIdValue”); ........ }
```

2.1.3 Receiver Portlet Portlet "portlet.xml" #

```
<portlet-app> <portlet> <portlet-name>PortletB</portlet-name> <supported-public-render-parameter>id1</supported-public-render-parameter> </portlet> <public-render-parameter> <identifier>id1</identifier> <qname xmlns:x="http://abc.com/userId">x:param1</qname> </public-render-parameter> </portlet-app>
```

2.1.4 A portlet can read public render parameter

```
request.getParameter(“id1”);
```

Inter - Portlet Communications 2.2

2. IPC Mechanisms

2.2 Event Inter-portlet Communications

Enable communications by adding to the portal-ext.properties:
portlet.event.distribution=ALL_PORTLETS

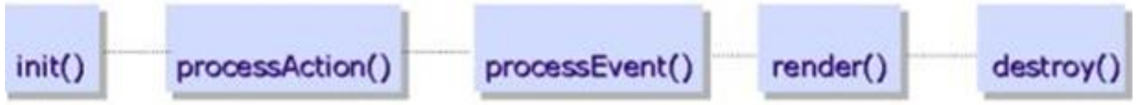
2.2.1. Sender Portlet : portlet.xml
<portlet-app> <portlet> <supported-publishing-event xmlns:x='http://liferay.com'> <qname>**x:empinfo**</qname> </supported-publishing-event>> </portlet> <event-definition xmlns:x='http://liferay.com'> <qname>**x:empinfo**</qname> <value-type>java.lang.String</value-type> </event-definition> </portlet-app>

2.2.2. Set the event in sender's process action
javax.xml.namespace.QName qName = new QName("http://liferay.com", "**empinfo**", "x");
response.setEvent(qName, "Replace with the meaningful event string");

2.2.3. Listener Portlet : portlet.xml
<portlet-app><portlet> <supported-processing-event xmlns:x='http://liferay.com'> <qname>x:empinfo</qname> </supported-processing-event> </portlet> <event-definition xmlns:x='http://liferay.com'> <qname>x:empinfo</qname> <value-type>java.lang.String</value-type> </event-definition> </portlet-app>

2.2.4: get the EVENT in the Portlet - Listener

```
public void handleProcessempinfoEvent(javax.portlet.EventRequest request, javax.portlet.EventResponse response) throws
javax.portlet.PortletException, java.io.IOException {
  javax.portlet.Event event = request.getEvent();
  String value = (String) event.getValue();
  System.out.print("Event value: " + value);
  response.setRenderParameter("empInfo", value);
}
```



Inter - Portlet Communications 2.3

2. IPC Mechanisms

2.3 Client-Side IPC:There are 2 APIs for client side IPC.

2.3.1 Event generation (call from Portlet-A):

```
Liferay.fire( '<eventName>', { name: value } );
```

E.g.

```
Liferay.fire( ‘startQuiz', { quizName: ‘Java', quizLevel: ‘light' } );
```

2.3.2 Event Listener (call from Portlet-B):

```
Liferay.on( '<eventName>',
function(event) { // your code } );
```

E.g.

```
Liferay.on( '<eventName>',
function(event) {
  showNews('', event.origin);
  showNews('', event.destination);
} );
```

ITS, Inc.

Inter - Portlet Communications 3

3. Cookies: Store and share data between the portlets (on different pages) via **Cookies**.
Limitations for cookies: **size < 4KB of data; number of cookies < 20 per server**

Seting the Cookies through jQuery:

```
<script src="/html/js/jquery/cookie.js" type="text/javascript" >
</script>
function setCookie(docURL) {
jQuery.cookie("cookieParam",docURL);
}
```

Setting the Cookies through java/jsp:

```
HttpServletResponse response = PortalUtil.getHttpServletResponse(actionResponse);
Cookie cookieParam = new Cookie(“user", user);
response.addCookie(cookieParam);
```

To get the Cookies through jQuery:

```
jQuery.cookie(“user");
```

To get the Cookie through java/jsp:

```
String userName = "";
Cookie[] cookies = request.getCookies();
if (cookies != null) {
  for (Cookie cookie : cookies) {
    if (cookie.getName(equals(“user")) {
      userName = cookie.getValue();
    }
 }
}
```

Refrence: Liferay Wiki