FREE eBook

LEARNING liferay

Free unaffiliated eBook created from
Stack Overflow contributors.

#liferay

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: liferay

It is an unofficial and free liferay ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official liferay.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with liferay

Remarks

Liferay Portal CE is a portal software built in Java by Liferay Inc.

Liferay DXP (Digital Experience Platform) is a platform built on top of Liferay Portal for digital solutions, integrating customer and user satisfaction analysis tools and Enterprise-grade quality performances and tooling. It was known as formerly known as *Liferay Portal EE*.

Since version 7.0 it is built using **OSGi** through **Apache Felix**.

Versions

Version	Code name	Release type	Release Date
7.0.1 GA2	Wilberforce	Community Edition	2016-06-10
7.0.10	Wilberforce	Digital Experience Platform	2016-05-04
7.0.0 GA1	Wilberforce	Community Edition	2016-03-31
6.2.3 GA4	Newton	Community Edition	2015-04-17
6.2.2 GA3	Newton	Community Edition	2015-01-15
6.2.1 GA2	Newton	Community Edition	2014-02-28
6.2.10 GA1	Newton	Enterprise Edition	2013-12-03
6.2.0 GA1	Newton	Community Edition	2013-11-01
6.1.2 GA3	Paton	Community Edition	2013-08-23
6.1.30 GA3	Paton	Enterprise Edition	2013-08-16
6.1.1 GA2	Paton	Community Edition	2012-07-31
6.1.20 GA2	Paton	Enterprise Edition	2012-07-31
6.1.10 GA1	Paton	Enterprise Edition	2012-02-15
6.1.0 GA1	Paton	Community Edition	2012-01-01
6.0.12 SP2	Bunyan	Enterprise Edition	2011-11-07
6.0.6	Bunyan	Community Edition	2011-03-04

Version	Code name	Release type	Release Date
6.0.11 SP1	Bunyan	Enterprise Edition	2011-01-13
5.2 SP5	Augustine	Enterprise Edition	2010-10-20
6.0.10	Bunyan	Enterprise Edition	2010-09-10
6.0.5	Bunyan	Community Edition	2010-08-16
6.0.4	Bunyan	Community Edition	2010-07-23
6.0.3	Bunyan	Community Edition	2010-07-20
6.0.2	Bunyan	Community Edition	2010-06-08
5.2 SP4	Augustine	Enterprise Edition	2010-05-19
6.0.1	Bunyan	Community Edition	2010-04-20
5.1 SP5	Calvin	Enterprise Edition	2010-03-12
6.0.0	Bunyan	Community Edition	2010-03-04
5.2 SP3	Augustine	Enterprise Edition	2010-01-07
5.2 SP2	Augustine	Enterprise Edition	2009-11-17
5.1 SP4	Calvin	Enterprise Edition	2009-10-23
5.2 SP1	Augustine	Enterprise Edition	2009-08-07
5.1 SP3	Calvin	Enterprise Edition	2009-07-20
5.2	Augustine	Enterprise Edition	2009-06-01
5.2.3	Augustine	Community Edition	2009-05-12
5.1 SP2	Calvin	Enterprise Edition	2009-05-12
5.2.2	Augustine	Community Edition	2009-02-26
5.1 SP1	Calvin	Enterprise Edition	2009-02-18
5.2.1	Augustine	Community Edition	2009-02-03
5.2.0	Augustine	Community Edition	2009-01-26
5.1 SP	Calvin	Enterprise Edition	2008-12-16
5.1.2	Calvin	Community Edition	2008-10-03

Version	Code name	Release type	Release Date
5.1.1	Calvin	Community Edition	2008-08-11
5.1.0	Calvin	Community Edition	2008-07-17
5.0.1 RC	Luther	Community Edition	2008-04-14
5.0.0 RC	Luther	Community Edition	2008-04-09

Examples

A basic installation for development and tests

Running the latest Liferay CE is straightforward:

1. Go to https://www.liferay.com/downloads.
2. Choose a bundle among the ones listed. For beginners, the Tomcat bundle is a good choice. Click in "Download."

3. Unzip the download package whenever you find fit. The unzipped directory will be the `LIFERAY_HOME` directory.
4. To start Liferay, just run the script `LIFERAY_HOME/tomcat-x.xx.xx/bin/startup.sh`; only on Windows environments run the script `LIFERAY_HOME\tomcat-x.xx.xx\bin\startup.bat`.
5. By default, once Liferay is up, a browser will open its local URL (http://localhost:8080/).
6. To log in, use the email `test@liferay.com`, and the password `test`.
7. To stop Liferay, just run the script `LIFERAY_HOME/tomcat-x.xx.xx/bin/shutdown.sh`; only on Windows environments run the script `LIFERAY_HOME\tomcat-x.xx.xx\bin\shutdown.bat`.

With these steps, you will have Liferay up and running in a "demo" mode. Liferay will use an Hypersonic DB by default, but it is unfit for production. Also, `test@liferay.com` is an administrator account with a default password, so it should be changed eventually. Yet, these steps are good to

get some idea on how Liferay looks like and works.

Read Getting started with liferay online: https://riptutorial.com/liferay/topic/932/getting-started-with-liferay

Chapter 2: Configure Google Tag manager(GTM) in liferay

Introduction

This documentation is not specific to liferay but can be used with reference to any web application.

Liferay provides Google Analytics(referred as GA ahead) by default,after configuring Analytics id GA-##### in Site settings.But this provides limited functionality,only allowing to track page views(Page title and URL).In order to expand it further,we can either embed GA script directly onto the site theme to trigger the required events or use GTM.

Examples

Using GTM to configure GA events

GTM simplifies the whole process of managing tags.In GTM terminology

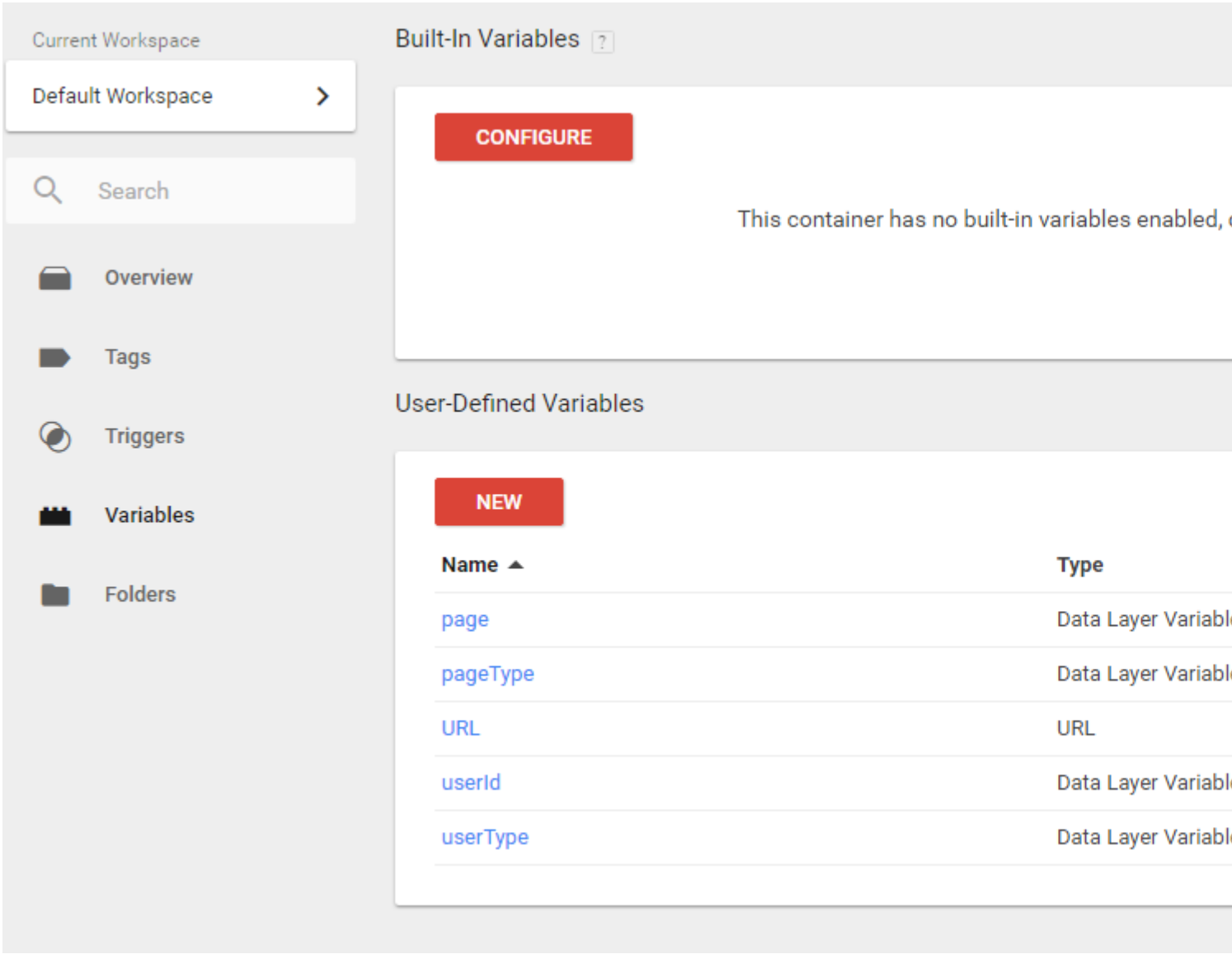
1. We put a GTM javascript snippet on the concerned page,in portal_normal.vm in custom theme in liferay, containing the GTM id and a data layer structure(if needed) to map values from page to variables
2. Corresponding to data layer variables,we need to create Variables at GTM end,which retrieve data from data layer
3. Subsequently,we create tags,which are basically fields which maps variables from data layer to events,which are triggered on certain conditions,leading to events being sent to respective tracking tool(GA,in our case).

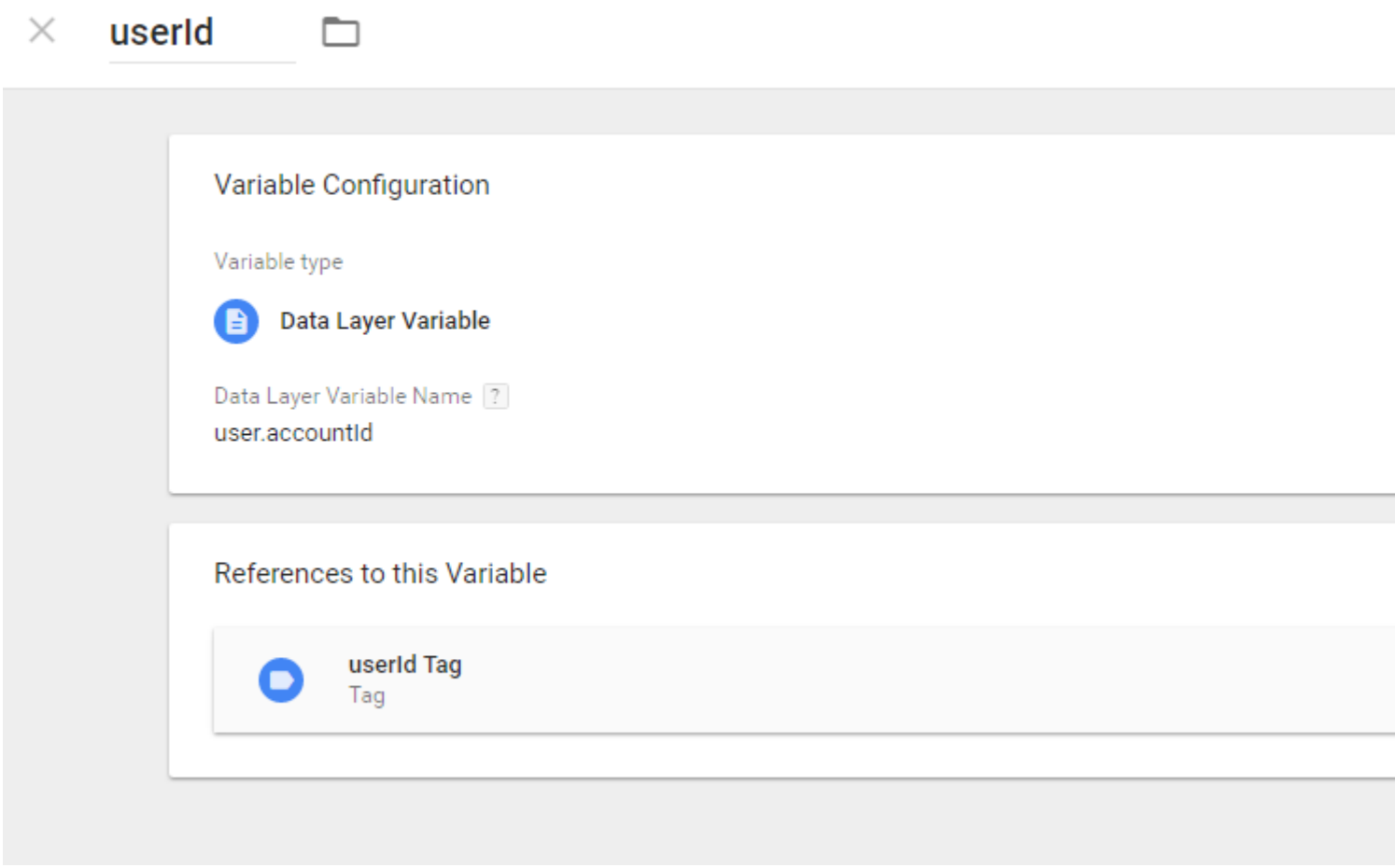
Below is a sample of GTM javascript snippet embedded on a page,

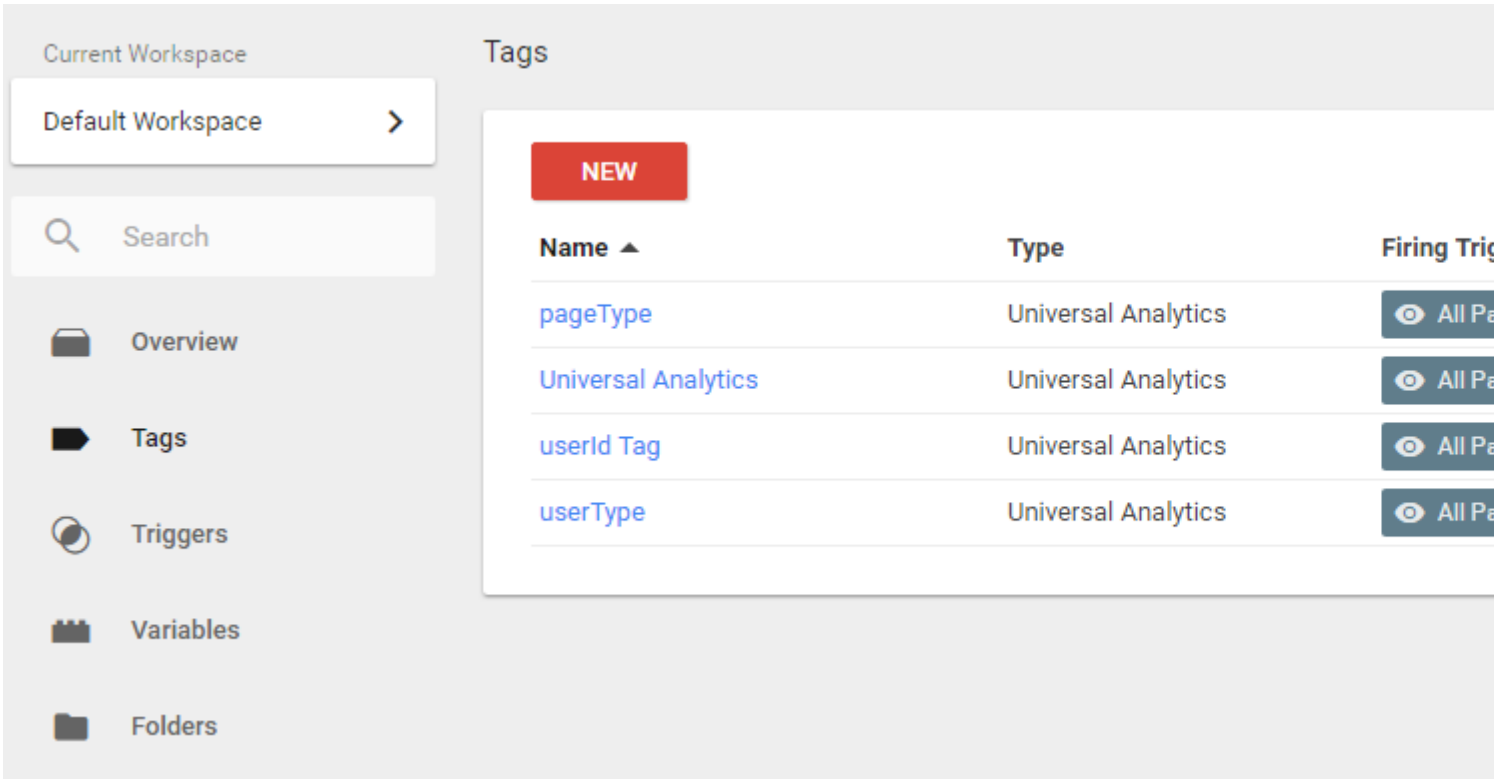
```
<body>
<!-- 1) Data layer section -->
<script type="text/javascript">
    dataLayer = [{
            "page" : "<? Virtual path of the page ?>"
            ,"pageType" : "<? Page type ?>"
            ,"user" : {
                        "type" : "<? User type ?>"
                        ,"userId" : "<? Logged user id ?>"
                        ,"country" : "<? Logged user country ?>"
                        ,"userRole" : "<? Role of user ?>"
                    }
        }];
  </script>
<!-- 2) GTM Container -->
<noscript><iframe src="//www.googletagmanager.com/ns.html?id=GTM-PK9HK8"
height="0" width="0" style="display:none;visibility:hidden"></iframe></noscript>
<script>(function(w,d,s,l,i){w[l]=w[l]||[];w[l].push({'gtm.start':
new Date().getTime(),event:'gtm.js'});var f=d.getElementsByTagName(s)[0],
```

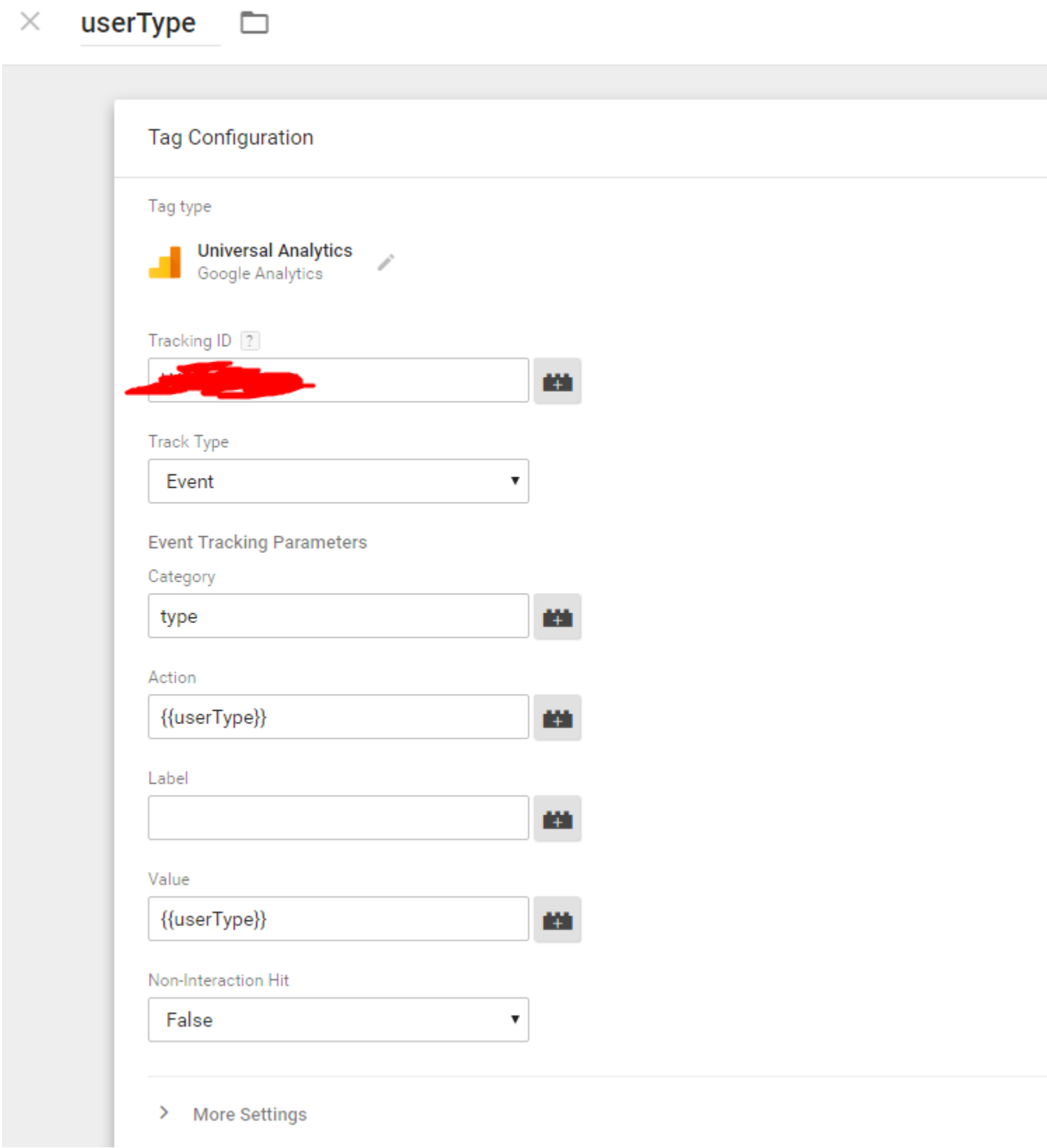
```
j=d.createElement(s),dl=l!='dataLayer'?'&l='+l:'';j.async=true;j.src=
'//www.googletagmanager.com/gtm.js?id='+i+dl;f.parentNode.insertBefore(j,f);
})(window,document,'script','dataLayer','<GTM-ID>');</script>
<!-- End Google Tag Manager -->
```

Post including this script in page,we need to configure the respective variables and tags from GTM end.

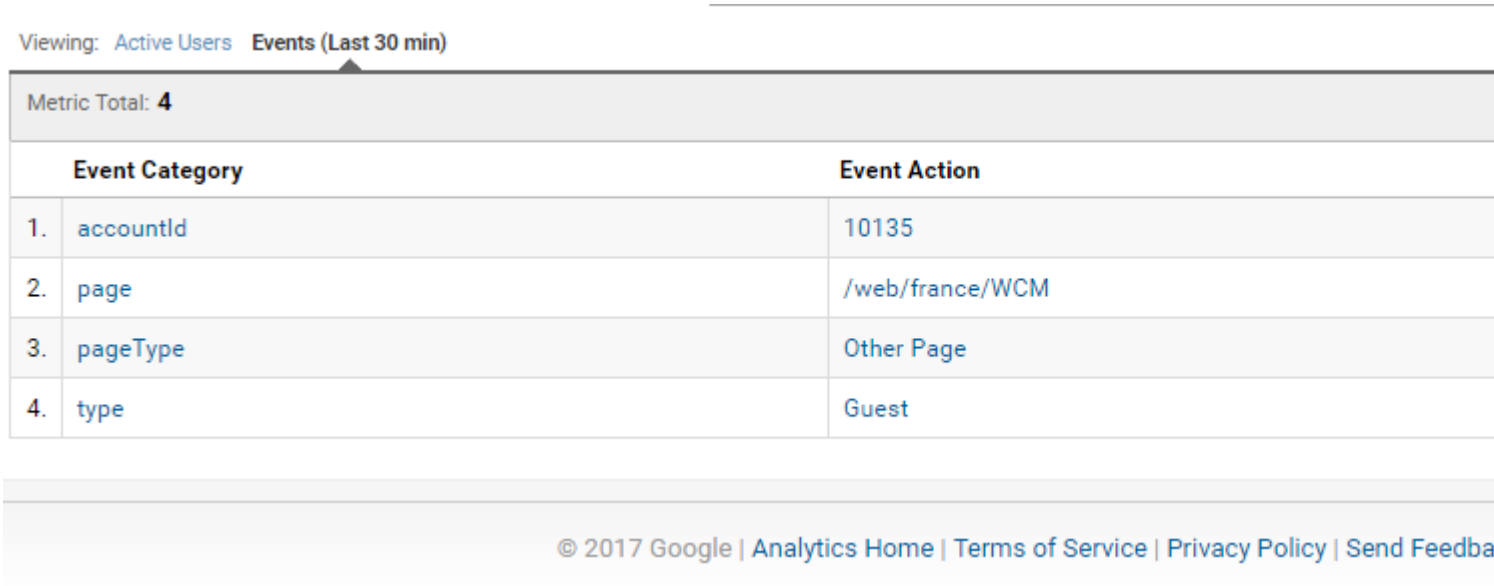



userId
Variable Configuration
Variable type
Data Layer Variable
Data Layer Variable Name
user.accountId
References to this Variable
userId Tag
Tag


Current Workspace
Default Workspace
Search
Overview
Tags
Triggers
Variables
Folders
Tags
NEW
Name
Type
Firing Trig
pageType
Universal Analytics
Universal Analytics
Universal Analytics
userId Tag
Universal Analytics
userType
Universal Analytics



Post we have configured the required fields,we can view events on GA console upon a user view.



Viewing: Active Users **Events (Last 30 min)**

Metric Total: **4**

	Event Category	Event Action
1.	accountId	10135
2.	page	/web/france/WCM
3.	pageType	Other Page
4.	type	Guest

In order to view the data sent from portal to GA,we can use Google Analytics Debugger plugin,to view events being sent to GA via browser console.

Read Configure Google Tag manager(GTM) in liferay online: https://riptutorial.com/liferay/topic/8723/configure-google-tag-manager-gtm--in-liferay

Chapter 3: Create a Quartz scheduler in liferay

Remarks

A scheduler serves to perform background tasks at certain defined intervals.

As per Liferay portlet DTD

<!- The scheduler-entry element contains the declarative data of a scheduler. ->

!ELEMENT scheduler-entry (scheduler-description?, scheduler-event-listener-class, trigger)

<!- The scheduler-description value describes a scheduler. ->

!ELEMENT scheduler-description (#PCDATA)

<!- The scheduler-event-listener-class value must be a class that implements com.liferay.portal.kernel.messaging.MessageListener. This class will receive a message at a regular interval specified by the trigger element. ->

!ELEMENT scheduler-event-listener-class (#PCDATA)

<!- The trigger element contains configuration data to indicate when to trigger the class specified in scheduler-event-listener-class. ->

!ELEMENT trigger (cron | simple)

Examples

Create a quartz scheduler to display some information

In order to create a scheduler,the entry needs to be created in

```
liferay-portlet.xml
```

provding scheduler class and trigger value for timing of scheduler triggering

```
<portlet-name>GetSetGo</portlet-name>
        <icon>/icon.png</icon>
        <scheduler-entry>
        <scheduler-description>This scheduler logs User count from portal</scheduler-
description>
            <scheduler-event-listener-class>com.example.scheduler.SchedulerSample</scheduler-
event-listener-class>
            <trigger>
```

```
                <simple>
                  <simple-trigger-value>
                    5
                  </simple-trigger-value>
                <time-unit>minute</time-unit>
            </simple>
            </trigger>
        </scheduler-entry>
```

The given entry provides

1. Scheduler description
2. Class name,which implements MessageListener class
3. Trigger,which provides intervals for defining trigger point for scheduler

    -Using Cron

    -Using Simple trigger value

In the given example,the scheduler will trigger after every 5 mins.

Next up we need to create scheduler class

```
package com.example.scheduler;

import com.liferay.portal.kernel.exception.SystemException;
import com.liferay.portal.kernel.log.Log;
import com.liferay.portal.kernel.log.LogFactoryUtil;
import com.liferay.portal.kernel.messaging.Message;
import com.liferay.portal.kernel.messaging.MessageListener;
import com.liferay.portal.kernel.messaging.MessageListenerException;
import com.liferay.portal.service.UserLocalServiceUtil;

public class SchedulerSample implements MessageListener {

    @Override
    public void receive(Message arg0) throws MessageListenerException {
        Log log=LogFactoryUtil.getLog(SchedulerSample.class);

        try {
            log.info("User Count for portal:"+UserLocalServiceUtil.getUsersCount());
        } catch (SystemException e) {

            log.info("User count is currently unavailable");
        }

    }

}
```

This scheduler simply displays output portal user count after every trigger interval to server console.

Create a dynamic quartz scheduler programmatically

There are specific scenarios where in we might need to create a Quartz scheduler,based on user input on when a scheduler should be triggered,apart from we can handle cases,where we have certain pre-defined functionalities,which need to be triggered based on user action,at a certain period.

This example receives user input on trigger timing,to trigger a scheduler.Here `ScheduledJobListener` class `imlements MessageListener`,which contains business logic to be executed on triggering the scheduler.The job is scheduled using `SchedulerEngineHelperUtil`class to trigger the job,after configuring the required params:

1. Trigger(using the cron text string and job name)
2. Message(using implementation for MessageListener class and portletId)
3. Scheduler storage types(which is MEMORY_CLUSTERED by default,can be set as PERSISTED to be stored in DB)
4. DestinationNames(which is SCHEDULER_DISPATCH for Liferay) which decides Message Bus destination to be used

The below snippet is part of action phase of the portlet interacting with user,to create and schedule a quartz job.

```
   //Dynamic scheduling
    String portletId= (String)req.getAttribute(WebKeys.PORTLET_ID);

    String jobName= ScheduledJobListener.class.getName();

    Calendar startCalendar = new GregorianCalendar(year , month, day, hh, mm, ss);
    String jobCronPattern = SchedulerEngineHelperUtil.getCronText(startCalendar, false);
                               //Calendar object & flag for time zone sensitive calendar

    Trigger trigger=new
CronTrigger(ScheduledJobListener.class.getName(),ScheduledJobListener.class.getName(),
jobCronPattern);

    Message message=new Message();
    message.put(SchedulerEngine.MESSAGE_LISTENER_CLASS_NAME,jobName);
    message.put(SchedulerEngine.PORTLET_ID, portletId);

    try {
          SchedulerEngineHelperUtil.schedule(

trigger,StorageType.PERSISTED,"Message_Desc",DestinationNames.SCHEDULER_DISPATCH,
                message,0);
         } catch (SchedulerException e)
                {
                    e.printStackTrace();
                }
```

Here,in order to create cron text,params are retrieved from user input For the cron text,we can also use the given reference for creating the cron pattern

```
    1. Seconds
    2. Minutes
    3. Hours
    4. Day-of-Month
```

```
    5. Month
    6. Day-of-Week
    7. Year (optional field)
    **Expression**     **Meaning**
    0 0 12 * * ?     Fire at 12pm (noon) every day
    0 15 10 ? * *     Fire at 10:15am every day
    0 15 10 * * ?     Fire at 10:15am every day
    0 15 10 * * ? *     Fire at 10:15am every day
    0 15 10 * * ? 2005     Fire at 10:15am every day during the year 2005
    0 * 14 * * ?     Fire every minute starting at 2pm and ending at 2:59pm, every day
    0 0/5 14 * * ?     Fire every 5 minutes starting at 2pm and ending at 2:55pm, every day
    0 0/5 14,18 * * ?     Fire every 5 minutes starting at 2pm and ending at 2:55pm, AND fire
every 5 minutes starting at 6pm and ending at 6:55pm, every day
    0 0-5 14 * * ?     Fire every minute starting at 2pm and ending at 2:05pm, every day
    0 10,44 14 ? 3 WED     Fire at 2:10pm and at 2:44pm every Wednesday in the month of March.
    0 15 10 ? * MON-FRI     Fire at 10:15am every Monday, Tuesday, Wednesday, Thursday and
Friday
    0 15 10 15 * ?     Fire at 10:15am on the 15th day of every month
    0 15 10 L * ?     Fire at 10:15am on the last day of every month
    0 15 10 L-2 * ?     Fire at 10:15am on the 2nd-to-last last day of every month
    0 15 10 ? * 6L     Fire at 10:15am on the last Friday of every month
    0 15 10 ? * 6L     Fire at 10:15am on the last Friday of every month
    0 15 10 ? * 6L 2002-2005     Fire at 10:15am on every last friday of every month during
the years 2002, 2003, 2004 and 2005
    0 15 10 ? * 6#3     Fire at 10:15am on the third Friday of every month
    0 0 12 1/5 * ?     Fire at 12pm (noon) every 5 days every month, starting on the first day
of the month.
    0 11 11 11 11 ?     Fire every November 11th at 11:11am.
```

and directly create a crontext string to be used based on user input

```
 String jobCronPattern="0 */5 * * * ?";
```

Here in this case,it fires after every five minutes.

References:

1. Dynamic scheduler creation
2. Scheduler application
3. Quartz FAQs

Read Create a Quartz scheduler in liferay online: https://riptutorial.com/liferay/topic/7998/create-a-quartz-scheduler-in-liferay

Chapter 4: Debug remote liferay server via Eclipse

Examples

Debug remote liferay server via Eclipse(without Liferay Remote IDE connector eclipse plugin)

To debug a server instance, start in debug mode. To do so, configure these parameters to be passed to the server:

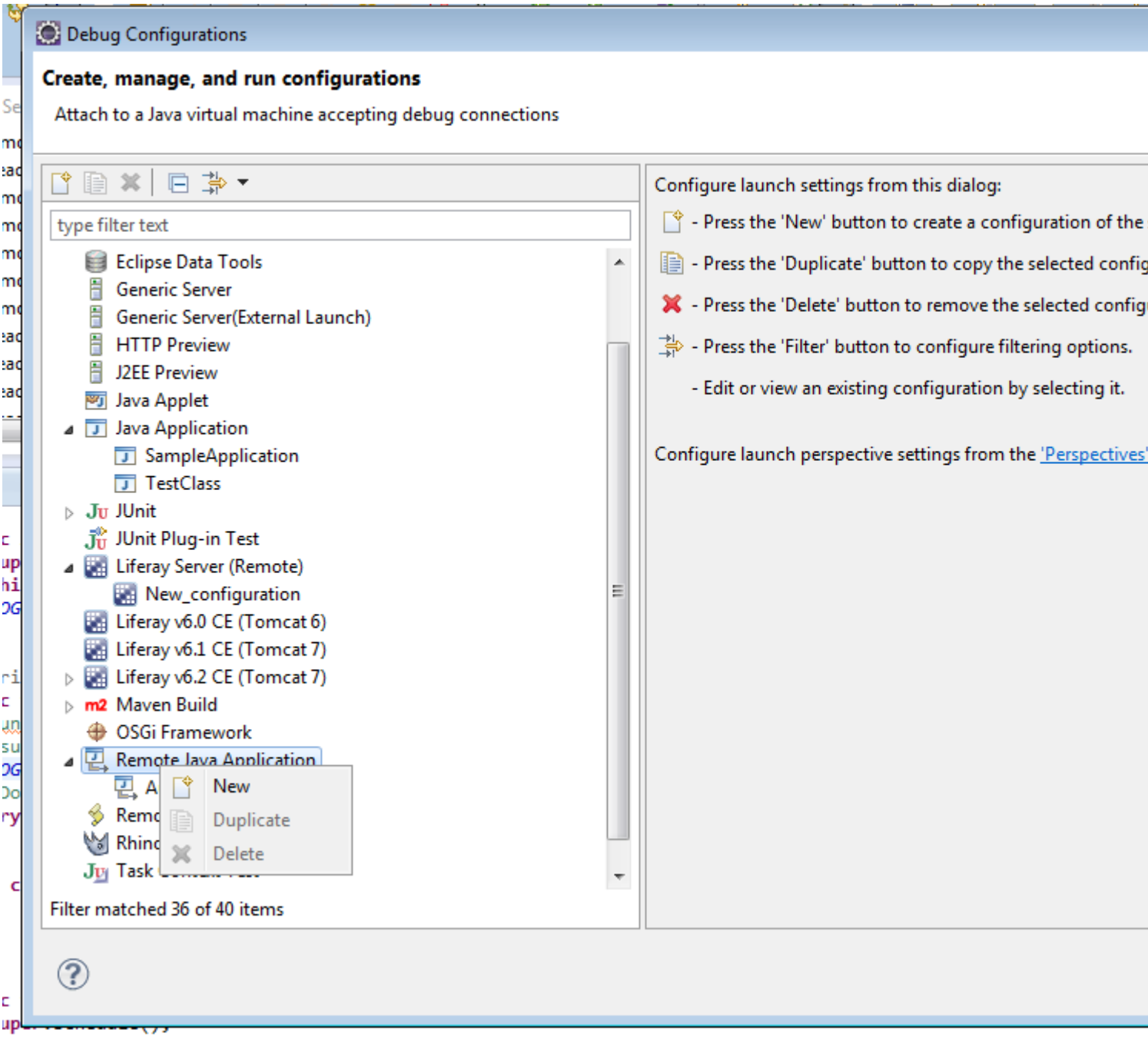
```
-Xdebug -Xrunjdwp:transport=dt_socket,address=8000,server=y,suspend=n
```

to setenv.bat(Windows) or setenv.sh(Unix)

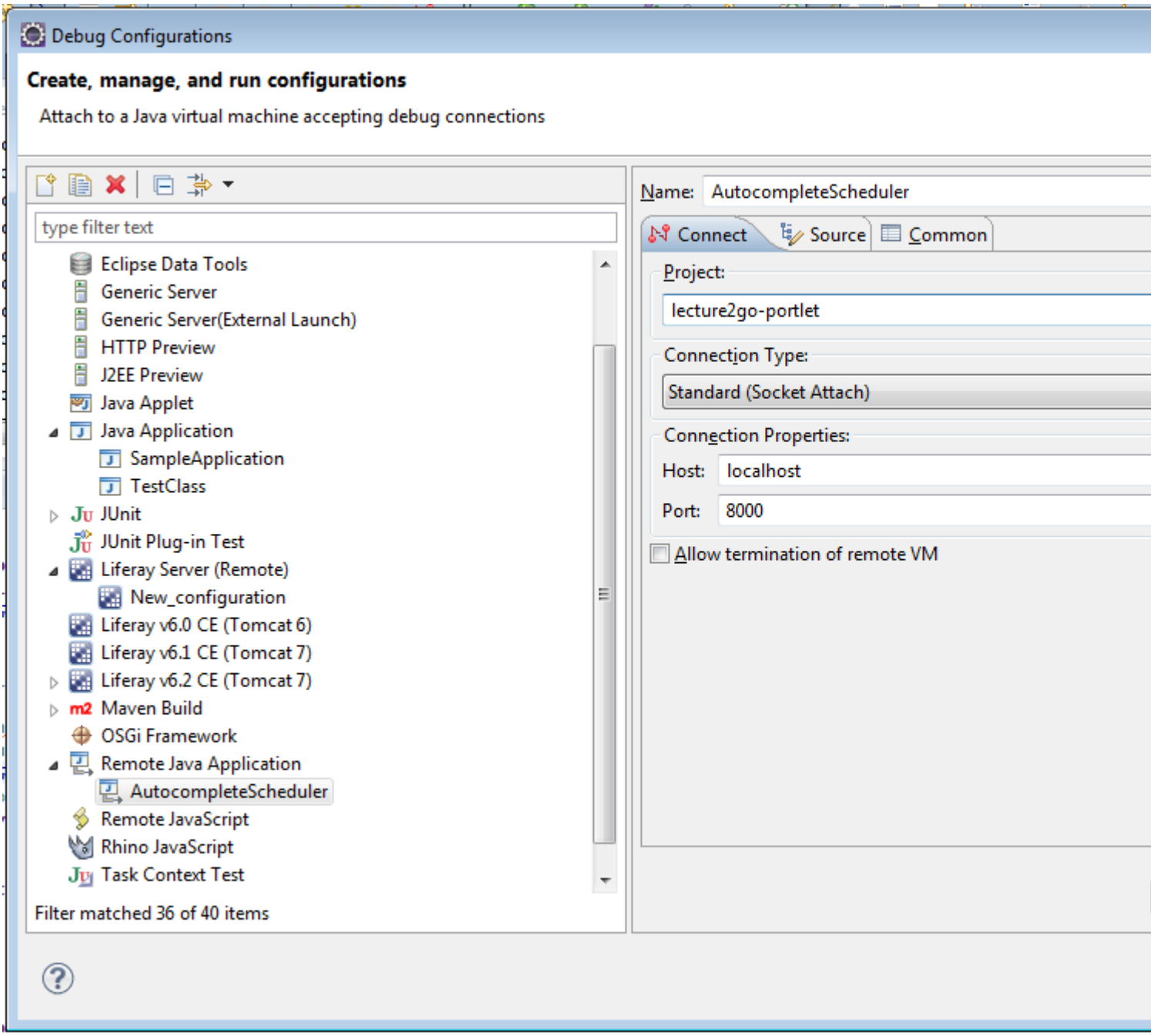
These initialize the server in debug mode, and listen for debug requests on the given port. Start the server and post the config.

In eclipse, the remote debug config needs to be configured to attach the source to the remote server. Follow the given steps:

1. Go to *Run->Debug Configurations->Remote java application*:



2. Create a new configuration from Remote Java Application:



3. Enter the given details:

```
Host name:localhost(For local instance)or Ip of the machine
Port:8000(By default)
```

4. Click *Debug* to intitiate attachments to the server instance.

Read Debug remote liferay server via Eclipse online:
https://riptutorial.com/liferay/topic/7891/debug-remote-liferay-server-via-eclipse

Chapter 5: Deploying a Plugin

Examples

Deploying to Glassfish

So, you make first a .war file let's say a portlet of name `<YOUR PLUGIN>.war`. You wanna have it running on a glassfish domain under Liferay portal.

Steps to success:

1. Navigate to Control Panel -> Plugins Installation on Liferay
2. Hit **Install new portlets**
3. Hit **Configuration**
4. Fill in to Deploy Directory a new place for deployment let's say `<YOUR DOMAIN>/autodeploy2`
5. Check that in the next line target is `<YOUR DOMAIN>/autodeploy` (it is the Glassfish default deployment directory)
6. Hit **Save**

Now deployment will be done by copy pasting files to that new directory `<YOUR DOMAIN>/autodeploy2`. The rest of it is handled automatically. Setting takes action immediately.

Done with deployment: Make a victory jig and enjoy :)

..you stop dancing and face a bug. You want a new revision to be deployed.. In this case, continue reading.

So, you have built your war again and want to re-deploy. Do the following:

1. undeploy old stuff from `<YOUR DOMAIN>/autodeploy` folder by deleting the war file. Don't delete any other file.
2. result is that `<YOUR PLUGIN>.war_UnDeployed` file will appear.
3. deploy new file by copying the newly built war in `<YOUR DOMAIN>/autodeploy2` folder.
4. result is that `<YOUR PLUGIN>.war_deployed` will appear in `<YOUR DOMAIN>/autodeploy` folder.

Make a dance again :)

Read Deploying a Plugin online: https://riptutorial.com/liferay/topic/1708/deploying-a-plugin

Chapter 6: Hooks in Liferay

Remarks

This works with Liferay Portal up to version 6.2.

Examples

JSP Hook

JSP hooks are a special liferay plugin that allow to modify core portlet jsp-s, lets say you want to modify the login portlet to show `Welcome in my custom login!`.

The minimal structure for a Hook Plugin is as follows:

```
[project-name]-hook/
 └── docroot/
     ├── WEB-INF/
     │   ├── src/
     │   ├── lib/
     │   ├── liferay-hook.xml
     │   ├── liferay-plugin-package.properties
     │   └── web.xml
     └── META-INF/
         ├── custom_jsps/
         └── MANIFEST.MF
```

`liferay-hook.xml` is the file that distiguishes the type of hook you're using, here you define inside the hook tag the proper parameter for the hook, for JSP hook:

```
<?xml version="1.0"?>
<!DOCTYPE hook PUBLIC "-//Liferay//DTD Hook 6.2.0//EN" "http://www.liferay.com/dtd/liferay-hook_6_2_0.dtd">

<hook>
    <custom-jsp-dir>/custom_jsps</custom-jsp-dir>
</hook>
```

`login.jsp` is found in Liferay in `/docroot/html/portlet/login/login.jsp`, to make a hook of it we need to add a jsp with the same name and path in our `custom_jsps` folder.

When the hook is deployed, Liferay will look in the `liferay-hook.xml` for the `custom-jsp-dir` value and will replace all the portal JSPs with the ones found in this directory. The original jsp's are saved with name `<orginal name>.portal.jsp` to be restored in case of hook undeployment.

We can even call the original JSPs in the new modified JSP if we want to keep the code making this adaptable to updates or upgrades of the underlying Liferay platform version. To do this, in your custom JSP use the following pattern:

```
<liferay-util:buffer var="contentHtml">
    <liferay-util:include page="/html/{ JSP file's path }" />
</liferay-util:buffer>
```

where `{ JSP file's path }` in this case will be `portlet/login/login.portal.jsp`. Doing this is called *extending the original jsp*.

Then we can add content to it with:

```
<%
contentHtml = StringUtil.add("Stuff I'm adding BEFORE the original content",
contentHtml,"\n");
contentHtml = StringUtil.add(contentHtml,"Stuff I'm adding AFTER the original content","\n");
%>
<%= contentHtml %>
```

Struts Action Hooks

This type of Hook can be used to override core portal (e.g `c/portal/login`) and portlet struts actions (e.g `/login/forgot_password`), this actions for Liferay Portal are specified in a `struts-config.xml` file in its `WEB-INF` folder.To override an action:

1. in `liferay-hook.xml` file of your hook plugin under `docroot/WEB-INF`, add a `struts-action` element within the hook element.
2. Inside `struts-action` element, add `struts-action-path` that specifies the action path you're overriding and `struts-action-impl` that specifies your custom action class.This looks like:

```
 <struts-action-path>/login/login</struts-action-path>
    <struts-action-impl>
    com.myhook.action.ExampleStrutsPortletAction
    </struts-action-impl>
</struts-action>
```

3. Create a Struts portlet action class that extends `BaseStrutsPortletAction`. An example of this class is:

```
public class ExampleStrutsPortletAction extends BaseStrutsPortletAction {

        public void processAction(StrutsPortletAction originalStrutsPortletAction,
                PortletConfig portletConfig, ActionRequest actionRequest,
                ActionResponse actionResponse) throws Exception {

            System.out.println("Custom Struts Action");

            originalStrutsPortletAction.processAction(originalStrutsPortletAction,
                    portletConfig, actionRequest, actionResponse);
        }

    public String render(StrutsPortletAction originalStrutsPortletAction,
            PortletConfig portletConfig, RenderRequest renderRequest,
            RenderResponse renderResponse) throws Exception {

        System.out.println("Custom Struts Action");
```

```
        return originalStrutsPortletAction.render(null, portletConfig,
                renderRequest, renderResponse);
    }
}
```

Calling the method being overridden, like `originalStrutsPortletAction.processAction`, is not obligatory but a best practice to keep the behavior from the Action unchanged in regards of Liferay Portal. This type of hook can be used to add new Struts Actions also, it's the same as modifying an existing action, in this case `liferay-hook.xml` would be:

```
<struts-action>
    <struts-action-path>/my/custom/path</struts-action-path>
    <struts-action-impl>
    com.myhook.action.ExampleStrutsAction
    </struts-action-impl>
</struts-action>
```

Hello User "Name" with hooks

This example will show how to make a simple "Hello User [name]" after the login. The example is based on performing a custom action using a hook

From your command line terminal, navigate to your Plugins SDK's hooks folder. To create a hook project, you must execute the create script. Here's the format to follow in executing the script:

> create.[sh|bat] [project-name] "[Hook Display Name]"

On Linux and Mac OS X, you'd enter a command similar to the one in this example:

> ./create.sh Hello-user "Hello User"

On Windows, you'd enter a command similar to the one in this example:

> create.bat Hello-user "My Hook"

Liferay IDE's New Project wizard and the create scripts generate hook projects in your Plugin SDK's hooks folder. The Plugins SDK automatically appends "-hook" to your project name.

Whether you created your hook project from the Liferay IDE or from the command line, you end up with the same project structure (see before).

- Determine the event on which you want to trigger your custom action. Look in the **portal.properties** documentation to find the matching event property. Hint: the event properties have .event in their name. There are session, startup, shutdown, and portal event properties in the following sections of the **portal.properties** documentation: Session - Startup Events - Shutdown Events - Portal Events
- In your hook project, create a Java class that extends the **com.liferay.portal.kernel.events.Action class.** Override the **Action.run(HttpServletRequest, HttpServletResponse)** method.

```
import com.liferay.portal.kernel.events.Action;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.liferay.portal.model.User;
import com.liferay.portal.util.PortalUtil;

public class HelloUser extends Action {
    public void run(HttpServletRequest req, HttpServletResponse res) {
        User user = PortalUtil.getUser(req);
        System.out.println("Hello User "+user.getScreenName());
    }
}
```

Important: If your action access the HttpServletRequest object, extend com.liferay.portal.kernel.events.Action; otherwise, extend com.liferay.portal.struts.SimpleAction.

- Create a properties file, **portal.properties**, inside your hook project's docroot/WEB-INF/src folder. Then add the name of the portal event property that corresponds to the event on which you want to perform your action. Specify your action class' fully qualified name as the property's value.

  ```
  `login.events.post=HelloUser`
  ```

  For example, to perform a class' action just prior to the portal logging in a user, you'd specify the login.events.pre property with your action class as its value. It could look like this property setting.

  Important: Since portal properties like login.events.pre accept multiple values, you must append your values to the existing values. You can repeatedly modify the properties from additional hooks.

  Only modify a portal property that accepts a single value from a single hook plugin. If you modify a property's value from multiple plugins, Liferay won't know which value to use.

- Edit your **docroot/WEB-INF/liferay-hook.xml** file and add your hook's portal properties file name as the value for the `<portal-properties>...</portal-properties>` element within your hook's `<hook>...</hook>` element. For example, if your hook's properties file name is **portal.properties**, you'd specify this element:

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

- Deploy your hook, go to your hook path and enter `ant clean deploy`you will see the .war in the dist folder.

Now if you login into liferay, you will see in the server log a message like "Hello user Admin".

Model Listener Hook

Background

Model Listener hook's are a type of Liferay plugin that listens for events taken on an model and executes code in response. Model Listener hooks are similar to Custom Struts Action hooks in that they respond to an action taken in the portal. However while Struts actions respond to an action taken by a user, a Model Listener responds (before or after) an event involving a Liferay model.

Differences

Here are a few examples of Struts Actions v. Model Listeners for comparison.

- **Struts Action**
  - User Login
  - Account Creation
  - Extend Session
  - Move Folder
- **Model Listener**
  - After folder is created
  - When user information is updated
  - After bookmark is removed
  - Before a role association is made

  The best resource for learning Liferay's architecture is through their source code. All their source files are located on GitHub and by viewing their JavaDocs. You can see all of the core portal models on the JavaDocs and all of the Struts Actions on the GitHub.

Example

In this tutorial we are going to develop a Model Listener that sends an email to a User after their account is first created. To do this we are going to write a class called **UserModelListener** which will extend Liferay's **BaseModelListener**. We will briefly go over hook creation and will cover the necessary modifications to the following config files

- **portal.properties**
- **liferay-hook.xml**

Getting Started

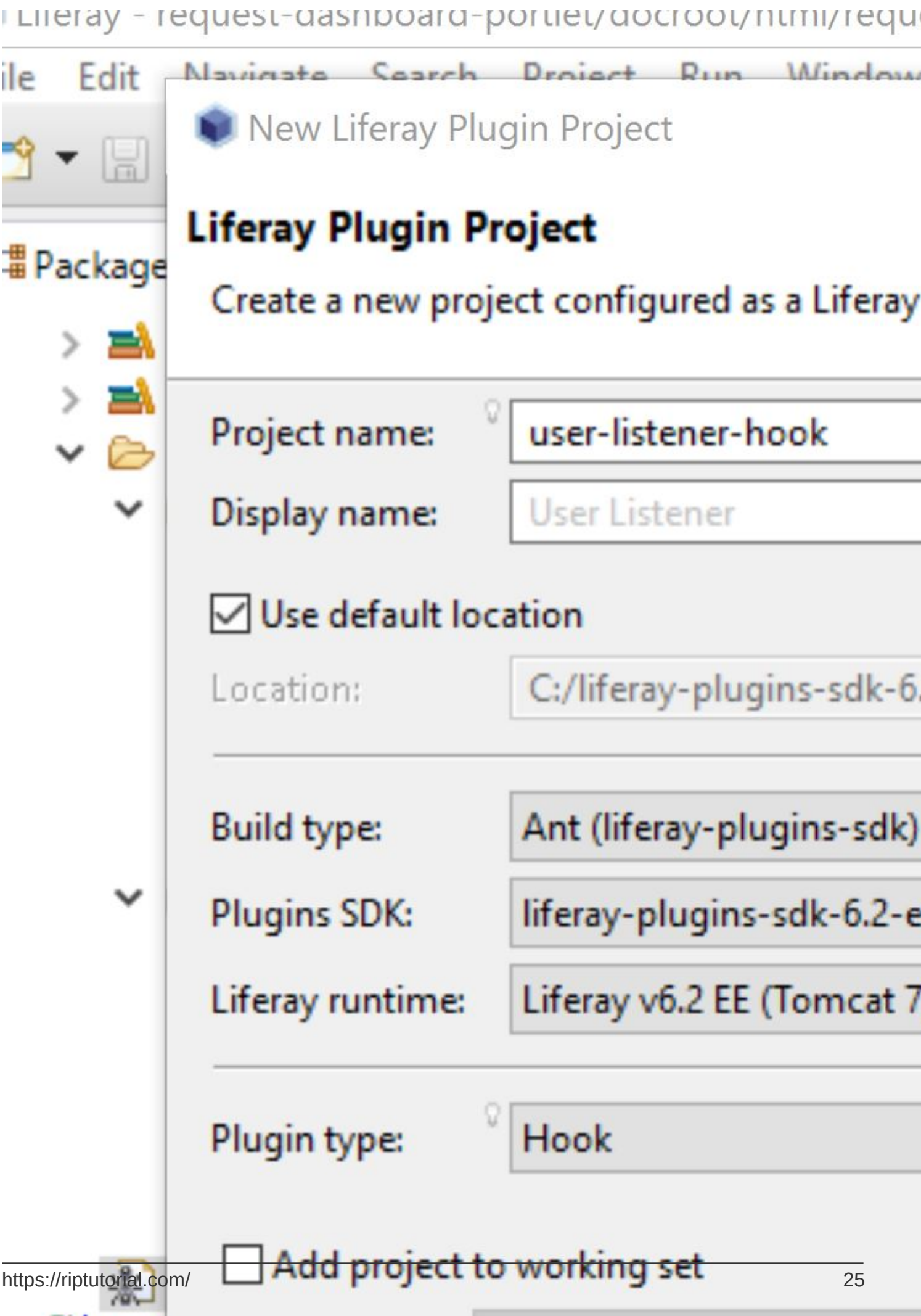
To begin developing your Model Listener hook you must first launch your Liferay IDE or Liferay Developer Studio application.

Both the Liferay IDE and the Liferay Developer Studio are customized **Eclipse** development environments. They are strikingly similar and one set of directions should be sufficient for both environments.

Inside your development environment execute the following steps.

1. In the top left corner click **File**
2. Hover your mouse over **New**
3. Click **Liferay Plugin Project**

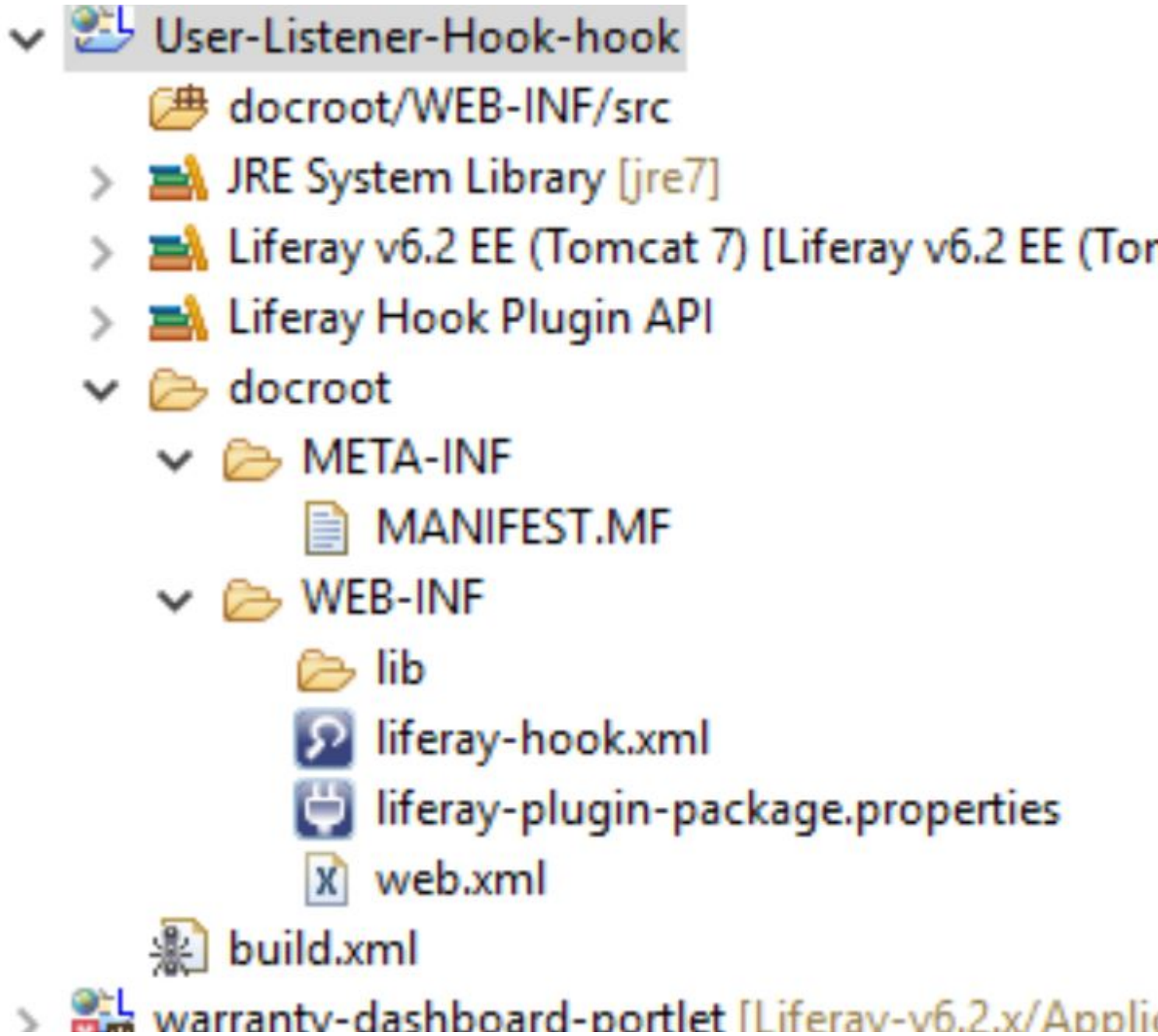
You will spawn this window.


Liferay - request-dashboard-portlet/docroot/html/requ
File Edit Navigate Search Project Run Window
New Liferay Plugin Project
Liferay Plugin Project
Create a new project configured as a Liferay
Package
Project name:
user-listener-hook
Display name:
User Listener
Use default location
Location:
C:/liferay-plugins-sdk-6.
Build type:
Ant (liferay-plugins-sdk)
Plugins SDK:
liferay-plugins-sdk-6.2-e
Liferay runtime:
Liferay v6.2 EE (Tomcat 7
Plugin type:
Hook
Add project to working set

- Select **Use Default Location**
- Build Type: **Ant**
- Plugin type: **Hook**

  Make sure that your project is located inside your **Liferays Plugins SDK** Hook directory. You will need to select your **SDK** and your **Runtime** accordingly.

In your **Package Explorer** perspective you will see the following directory structure.



Listener Development

Now that you have created your hook you will need to create your custom **UserModelListener** class. This class will extend Liferay's BaseModelListener class.

Liferay's BaseModelListener class is an abstract class that implements the ModelListener interface. You do not want to implement the ModelListener interface directly as it will require you to override all of it's methods.

The following methods are provided to you by the **ModelListener** interface through the **BaseModelListener** abstract class.

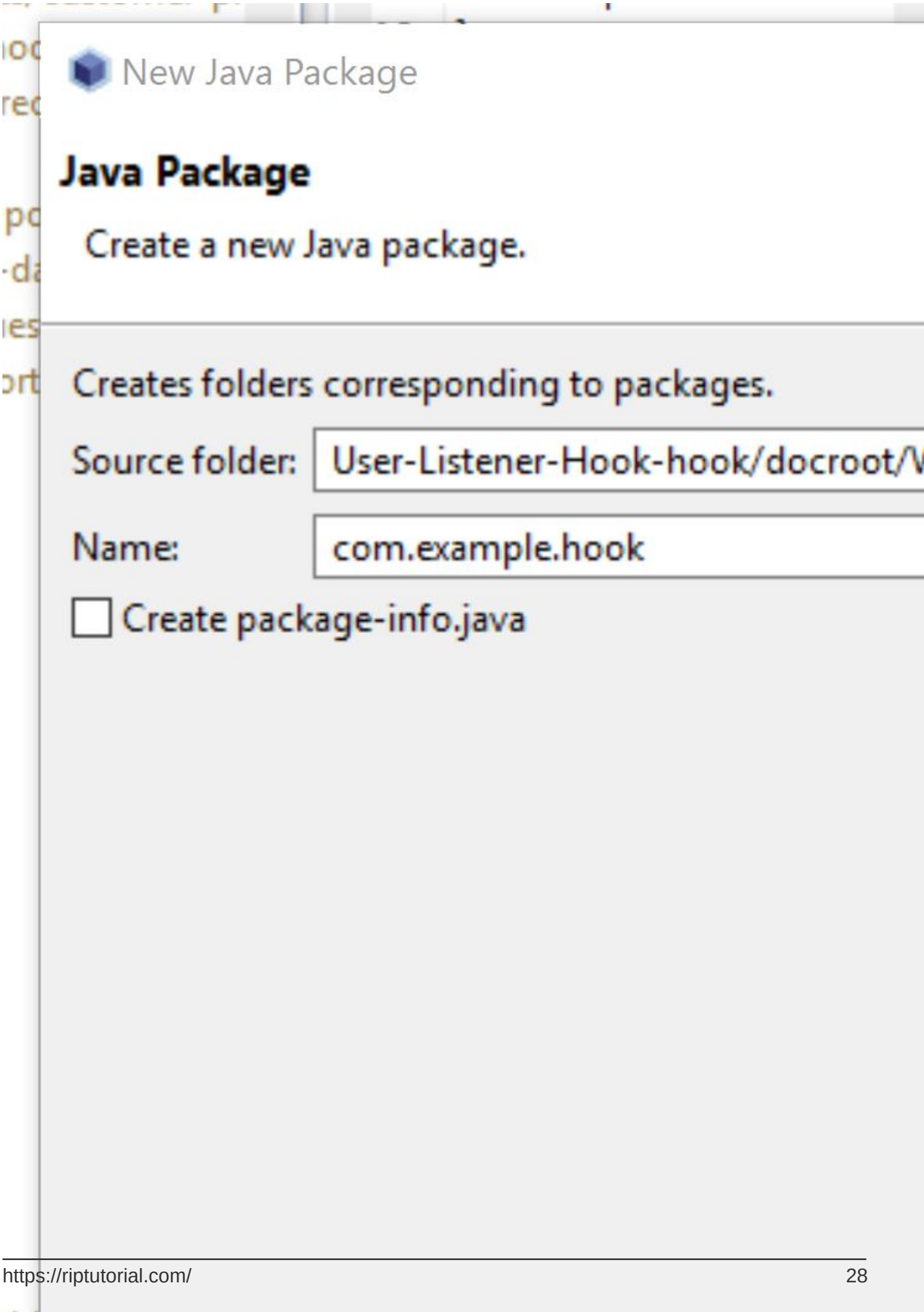
- onAfterAddAssociation
- onAfterCreate
- onAfterRemove
- onAfterRemoveAssociation
- onAfterUpdate
- onBeforeAddAssociation
- onBeforeCreate
- onBeforeRemove
- onBeforeRemoveAssociation
- onBeforeUpdate

Create your **UserModelListener** class inside the following directory. To create the class via the GUI simply execute the following commands

- Click **File** in the top left corner
- Hover your mouse over **New**
- Click **Class**

```
docroot/
     WEB-INF/
           src/
```

Enter the information shown below


New Java Package
Java Package
Create a new Java package.
Creates folders corresponding to packages.
Source folder:
User-Listener-Hook-hook/docroot/
Name:
com.example.hook
Create package-info.java

class, packaged inside **com.example.code**, for the **DLFolder** model we would have the following property

```
value.object.listener.com.liferay.portal.model.DLFolder =
com.example.code.CustomerDLFolderModelListener
```

Lastly, locate your liferay-hook.xml file. In **Source** view, write the following.

```
<?xml version="1.0"?>
<!DOCTYPE hook PUBLIC "-//Liferay//DTD Hook 6.2.0//EN" "http://www.liferay.com/dtd/liferay-
hook_6_2_0.dtd">

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

Explanation

1. Line one is an optional **prolog** which specifies the document version and (in some cases) the character set.
2. Line 2 is a formal **DocType Definition** (DTD) which explicitly defines which elements and attributes are valid
3. Line 3 and 5 consist of the parent **Hook element** (one of the valid elements supported by this DTD)
4. Line 4 overrides and extends the **portal.properties** file in **${liferay.home}**

   To see what other elements can be used in this XML file you can reference the URL within the DocType Definition. This is standard for all **XML** and **SGML** files with a **DTD**. Another example of a **Liferay XML** file with a **DTD** is **service.xml** (Liferay's **ORM** implementation based on **Hibernate**).

Build and Deploy

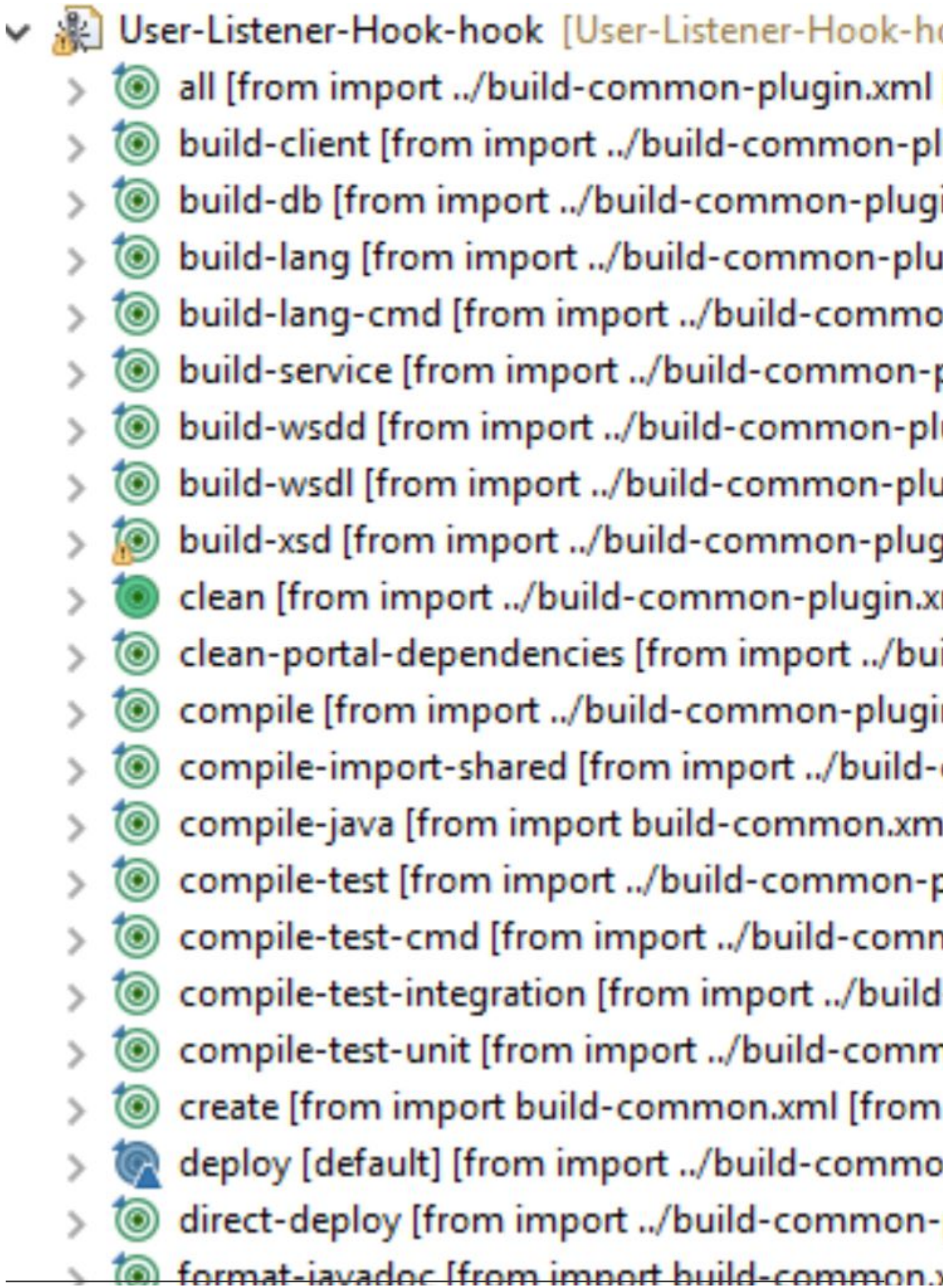
Building and deploying hooks is a simple process. Liferay Plugin development supports build and dependency automation with

- Ant
- Ivy
- Maven
- Gradle

In our example we utilized **Ant** for build automation. The **build.xml** file contains the build commands (known as **targets** in **Ant**). To build your hook simply execute the following commands.

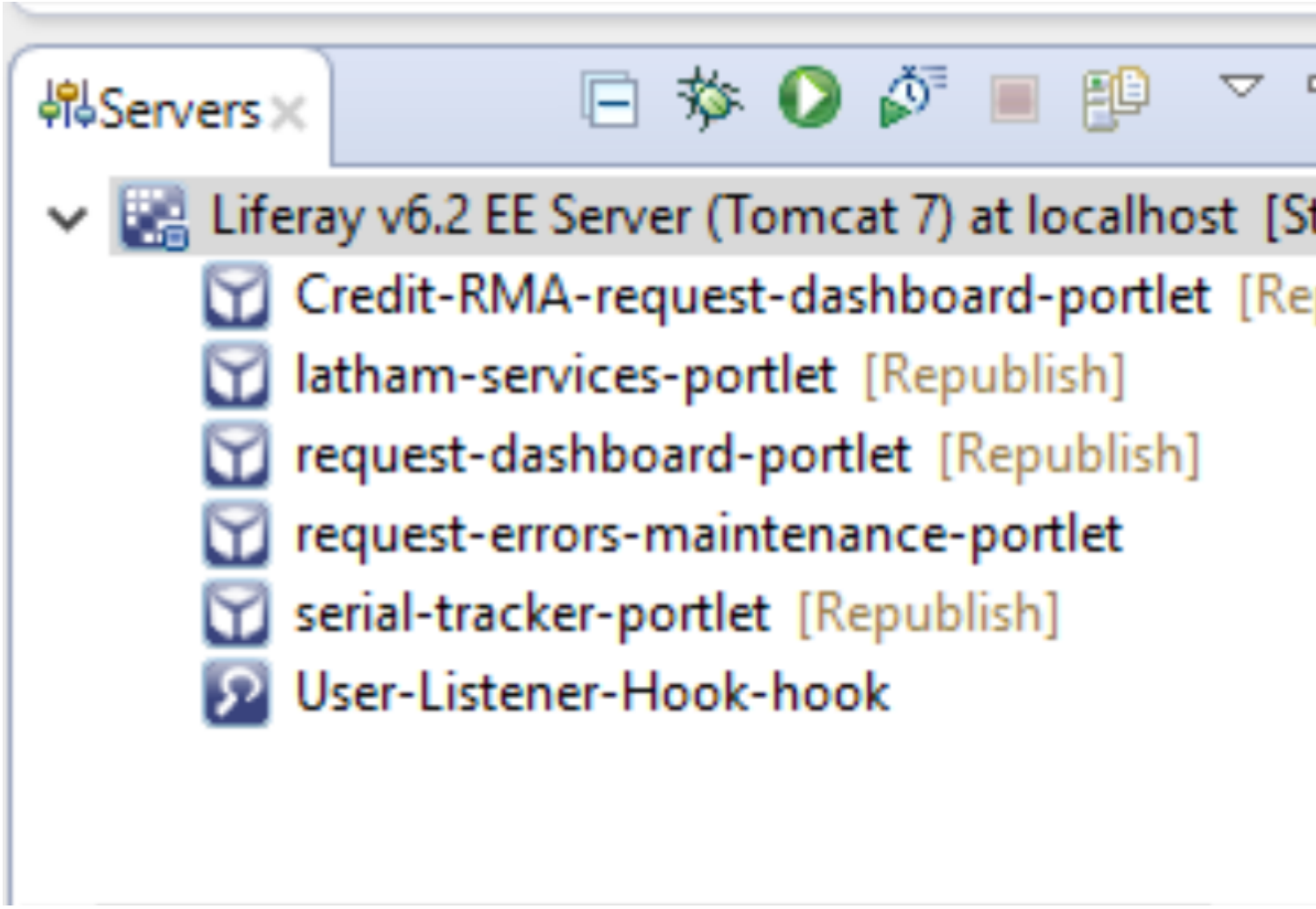
1. Located your **build.xml** file
2. In your IDE, drag the **build.xml** file into the **Ant** perspective

3. Expand the file and run the **all** target



In your console view you should see something similar to the following

2. Locate **User-Listener-Hook** under the **Available** selection
3. Once highlighted click the **Add** button and click **OK**
4. Click the **Play** button in the **Server** perspective



Please let me know if you have any questions, comments, concerns, etc. All constructive feedback is greatly appreciated!

Read Hooks in Liferay online: https://riptutorial.com/liferay/topic/3712/hooks-in-liferay

Chapter 7: Inter portlet communication

Introduction

This manual contains the various ways in which portlet can co-ordinate or communicate amongst each other and the various scenarios for which a particular approach is used.

Remarks

References:

1. Public render param
2. JSR 286 specs
3. Portlet session

Examples

Using Public render parameter

This approach was introduced in JSR 286.

In JSR 168,render parameters set in *processAction* of a portlet were available only in that portlet.With the Public Render Parameters feature, the render parameters set in the *processAction* of one portlet will be available in render of other portlets also.In order to configure this,for all the portlets supporting this:

Add `<supported-public-render-parameter>` tag ,just before the portlet tag ends in `portlet.xml`

```
<security-role-ref>
    <role-name>user</role-name>
</security-role-ref>
<supported-public-render-parameter>{param-name}</supported-public-render-parameter>
</portlet>
```

Add `<public-render-parameter>` tag just before the `<portlet-app>` tag ends

```
  <public-render-parameter>
    <identifier>{param-name}</identifier>
    <qname xmlns:x="localhost">x:{param-name}</qname>
  </public-render-parameter>
</portlet-app>
```

In the `processAction` method,the param value needs to be set in the response

```
res.setRenderParameter({param-name},{param-value});
```

Post we are done with configuring this for all the required portlet,after executing the action phase of the concerned portlet,the param should be available in render phase for all supporting portlets on the page,irrespective of being part of same or different application(war).

Using Portlet session

This is one approach which has been there since JSR 168.It allows us to share attributes using portlet session.A portlet session can have different types of scopes:

1. Portlet scope(attributes available only within portlet)
2. Application scope(attributes available within whole application[war])

In order to use this approach,we do not need to make any entries in portlet configuration,as portlet session is readily available in portlet request:

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

or

```
PortletSession session = renderRequest.getPortletSession();
session.setAttribute("attribute-name","attribute-value", PortletSession.PORTLET_SCOPE);
```

The attribute can only be retrieved from the respective scope only.Like for attribute set in portlet scope,we need to fetch it using

```
PortletSession session = renderRequest.getPortletSession();
String attributeValue = (String) session.getAttribute("attribute-name",
PortletSession.PORTLET_SCOPE);
```

The major limitation of this approach is lack of sharing among other portlet,outside of application scope.In order to overcome this,there is liferay specific approach to add `<private-session-attributes>` to `liferay-portlet.xml`

```
    <private-session-attributes>false</private-session-attributes>
    <header-portlet-css>/css/main.css</header-portlet-css>
    <footer-portlet-javascript>/js/main.js</footer-portlet-javascript>
    <css-class-wrapper>{portlet-name}</css-class-wrapper>
</portlet>
```

for all portlets,where the attributes are set and retrieved.

Using eventing feature

The eventing mechanism is an extended version of the public render param,with additonal feature to pass custom objects to other portlets,but with an overhead of event phase.

To achieve this,this mechanism consists of

1. Publisher portlet
2. Processor(consumer) portlet,where both may be part of different portlet applications.

To start with,

Add `<supported-publishing-event>` tag to the publisher portlet in `portlet.xml`

```
    <security-role-ref>
        <role-name>user</role-name>
    </security-role-ref>
    <supported-publishing-event>
         <qname xmlns:x="http:sun.com/events">x:Employee</qname>
    </supported-publishing-event>
  </portlet>
```

Add `<supported-processing-event>` tag to the processor portlet in `portlet.xml`

```
<security-role-ref>
        <role-name>user</role-name>
    </security-role-ref>
    <supported-processing-event>
        <qname xmlns:x="http:sun.com/events">x:Employee</qname>
     </supported-processing-event>
</portlet>
```

Add `<event-definition>`tag to both the portlets,defining event-name and type in `portlet.xml`

```
<event-definition>
  <qname xmlns:x="http:sun.com/events">x:Employee</qname>
  <value-type>com.sun.portal.portlet.users.Employee</value-type>
</event-definition>
   </portlet-app>
```

Next we need to create class for the event type(in case of custom type)

```
public class Employee implements Serializable {
  public Employee() {
  }
  private String name;
  private int userId;

  public String getName() {
    return name;
  }

  public void setName(String name) {
    this.name = name;
  }

  public int getUserId() {
    return userId;
  }
  public void setUserId(int id)
  {
    this.userId = id;
  }
```

}

Now,in the publisher portlet,the event needs to be published in the action phase

```
    QName qname = new QName("http:sun.com/events" , "Employee");
    Employee emp = new Employee();
    emp.setName("Rahul");
    emp.setUserId(4567);
    res.setEvent(qname, emp);
```

Post we have published the event,it needs to be processed by the publisher portlet in the event phase.

The event phase was introduced in JSR 286 and is executed before render phase of the portlet,when applicable

```
@ProcessEvent(qname = "{http:sun.com/events}Employee")
public void processEvent(EventRequest request, EventResponse response) {

    Event event = request.getEvent();
    if(event.getName().equals("Employee")){
      Employee payload = (Employee)event.getValue();
      response.setRenderParameter("EmpName",
      payload.getName());
    }

}
```

which can then be retrieved from the render parameter via render request.

Read Inter portlet communication online: https://riptutorial.com/liferay/topic/8370/inter-portlet-communication

Chapter 8: Setting up SSL

Remarks

Make sure you have a valid ssl certificate provided by a third party. You can also use a selfsigned certificate, but for dev only. Letsencrypt provides free certificates that can be used in production....

Use keytool to import the certificate to the keystorechain of java.

Examples

How to enable SSL on Tomcat and Liferay

Make sure your tomcat configurations file, server.xml has this line:

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11Protocol"
              maxHttpHeaderSize="8192" SSLEnabled="true"
              maxThreads="150" minSpareThreads="25" maxSpareThreads="75"
              enableLookups="false" disableUploadTimeout="true"
              acceptCount="100" scheme="https" secure="true"
              clientAuth="false" useBodyEncodingForURI="true"
              sslEnabledProtocols="TLSv1.2"
              keystorePass="passwordtokeystore"
              keystoreFile="/path/to/.keystoreChain"
              truststoreFile="%JAVA_HOME%/jdk1.8.0_91/jre/lib/security/cacerts"
              />
```

Its important to choose the right sslprotocols, you can add more sslprotocols with a comma seperation inbetween the ssl protocols like this:

```
sslEnabledProtocols="TLSv1, TLSv1.1, TLSv1.2"
```

Then make sure that your portal-ext.properties file in Liferay have this configuration lines:

```
web.server.protocol=https
```

Read Setting up SSL online: https://riptutorial.com/liferay/topic/4320/setting-up-ssl

Chapter 9: Using Dynamic and custom SQL query in Liferay

Introduction

There are scenarios when dealing with service layer in liferay, when we need to query database with too many clauses or dealing with multiple tables.In such cases,we use either of:

1)Dynamic query(wrapper on Hibernate criteria API)

2)Custom SQL queries

Remarks

References:

1. Custom SQL
2. Dynamic query

Examples

Using Dynamic query in Liferay

For most of the scenarios involving entities from service layer,we can make do with the default service calls,with some help from the finders as well.For simple scenarios involving multiple entities,we move towards using Dynamic query API.This is a wrapper API for the Criteria API used in Hibernate.It can be used for cases,where we need to generate dynamic query,which is not very complex in nature,using several constructs from the API. To start with,some of the most commonly used constructs are: `DynamicQueryFactoryUtil`-Used for constructing query

`RestrictionsFactoryUtil`-Used for providing restrictions i.e.fields for comparison with a certain value to narrow down the results matching a certain value or within a range,etc

`ProjectionFactoryUtil`-Used for providing projections to get fields which will be part of search result i.e. instead of providing the whole entity,will provide only certain fields or apply aggregration function(such as min.max,avg) on the same.

`PropertyFactoryUtil`-Used for comparison of some property from the entity class to mostly do comparsion with other fields from a query

The implementation of these classes are present in dao.orm.jpa package with all the available methods

Read Using Dynamic and custom SQL query in Liferay online:
https://riptutorial.com/liferay/topic/10863/using-dynamic-and-custom-sql-query-in-liferay

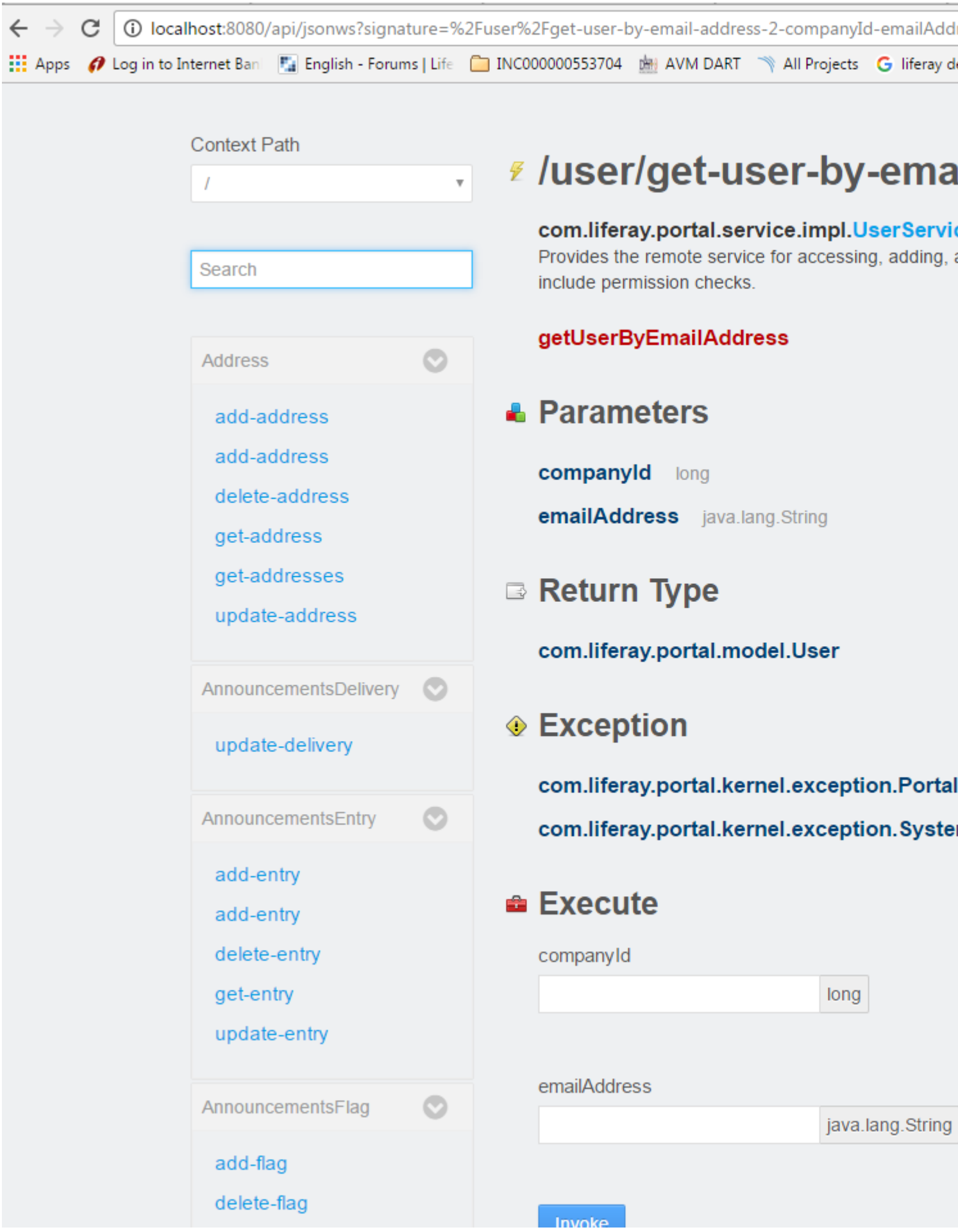
Chapter 10: Using Restful web service in Liferay

Examples

Consume Liferay JSON service for GET requests

Liferay exposes many default and custom services available to other systems via JSON. To explore services on a particular liferay instance, use a given URL - A local instance in this case:

```
http://localhost:8080/api/jsonws/
```



Select the required service, consume the service with the given syntax and parameters:

```
/user/get-user-by-email-address
```

Use `companyId and emailAddress` to retrieve the user with the expected datatypes, as well as possible exceptions to be handled by the consumer.

The following example consumes this service from a portlet. The given utility class method makes a call to the webservice, passing the necessary arguments:

```
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;

import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import sun.misc.BASE64Encoder;

import com.liferay.portal.kernel.util.StringUtil;
import com.liferay.portal.theme.ThemeDisplay;

public class WebServiceUtil {

public static String requestWebService(ThemeDisplay themeDisplay) {

    String url="http://localhost:8080/api/jsonws/user/get-user-by-email-address/company-
id/{company-id}/email-address/{email-address}";

    String groupId= Long.toString(themeDisplay.getCompanyId());
    String userEmail="test@liferay.com";

    String[] searchList={"{company-id}","{email-address}"};
    String[] replList={groupId,userEmail};

    //Path params are replaced with args to make web service call
    url=StringUtil.replace(url, searchList, replList);

    System.out.println(url);
    StringBuilder sb = new StringBuilder();
    JSONObject jsonObject=new JSONObject();
    try
    {
        URL urlVal = new URL(url);
        HttpURLConnection conn = (HttpURLConnection) urlVal.openConnection();

        //The user credentials are directly used here only for the purpose of example,always
fetech these details from an external props file.

        String uname ="test@liferay.com";
        String pswd="test";
        String authStr=uname+":"+pswd;

        //Encoding username+pswd to be added to request header for making web service
call
        String authStrEnc=new BASE64Encoder().encode(authStr.getBytes());

        /*Authorization type is set to consume web service
```

```
        and encoded combination is set in header to autheticate caller*/

        conn.setRequestMethod("GET");
        conn.setRequestProperty("Accept", "application/json");
        conn.setRequestProperty("Authorization", "Basic "+authStrEnc);

        BufferedReader brf = new BufferedReader(new InputStreamReader(conn.getInputStream()));

        JSONParser json=new JSONParser();
        jsonObject=(JSONObject)json.parse(brf);


        int cp;
        while ((cp = brf.read()) != -1) {
          sb.append((char) cp);
        }
    }
    catch(IOException e)
    {
        System.out.println("Something went wrong while reading/writing in stream!!");
    }
    catch (ParseException e) {
        System.out.println("Parse error");
    }

    //For purpose of simplicity we have fetched one of the fields from JSON response
    return (String)jsonObject.get("firstName");

}

}
```

Read Using Restful web service in Liferay online: https://riptutorial.com/liferay/topic/7821/using-restful-web-service-in-liferay

Credits

S. No	Chapters	Contributors
1	Getting started with liferay	brandizzi, Community, Pier Paolo Ramon, Pierpaolo Cira, rp.
2	Configure Google Tag manager(GTM) in liferay	Shivam Aggarwal
3	Create a Quartz scheduler in liferay	Shivam Aggarwal
4	Debug remote liferay server via Eclipse	4444, Shivam Aggarwal
5	Deploying a Plugin	mico, Pier Paolo Ramon, rp.
6	Hooks in Liferay	Chris Maggiulli, El0din, KLajdPaja, Pier Paolo Ramon, rp.
7	Inter portlet communication	a_horse_with_no_name, Shivam Aggarwal
8	Setting up SSL	El0din, rp., Tofik Sahraoui
9	Using Dynamic and custom SQL query in Liferay	Shivam Aggarwal
10	Using Restful web service in Liferay	4444, Shivam Aggarwal