

# Workbench Developer Documentation

version 1.0

*Last generated: February 26, 2019*

---



© 2019 KISTI. This is a boilerplate copyright statement... All rights reserved. No part of this publication may be reproduced, distributed, or transmitted in any form or by any means, including photocopying, recording, or other electronic or mechanical methods, without the prior written permission of the publisher, except in the case of brief quotations embodied in critical reviews and certain other noncommercial uses permitted by copyright law.

# Table of Contents

## 개발 환경 구성

Lifeary 6.2 설치 파일 다운로드 .....	2
Lifeary IDE 설치 .....	6
Developer Studion 실행 .....	15
JDK 1.7 설정 .....	17
tomcat 서버 설정 .....	26
ANT 1.9 설정 .....	39
SDK 설정 .....	44

## 워크벤치 편집기 개발

### 워크벤치 분석기 개발

프로젝트 생성 .....	54
프로젝트 기본 설정 .....	67
스크립트 단위의 뷰 설정 .....	72
이벤트 처리 .....	78
초기값 설정 .....	81
파일 탐색기 설정 .....	84
이벤트 처리 함수 .....	91
공용 함수 .....	97

### 워크벤치 이벤트 정의

HANDSHAKE .....	108
REGISTERED .....	110
REQUESTDATA .....	111
RESPONSEDATA .....	112
LOADDATA .....	116
INITIALIZE .....	117
REFRESHOUTPUTVIEW .....	118
DATACHANGED .....	119

# Liferay 설치 파일 다운로드

**Summary:** 본 문서는 워크벤치 기반 전후처리기 개발을 위한 매뉴얼입니다.

EDISON 포털 기반 워크벤치에서 동작하는 전후처리기 개발을 진행하기 위해서는 다음과 같은 파일들을 설치해야 합니다. 해당 파일을 다운받아 개발환경에 맞게 설치해 주시기 바랍니다.

## Liferay 설치 파일 다운로드

Liferay 설치를 위해서는 필수적으로 3가지의 파일이 필요합니다. 해당 파일들은 오픈소스로 누구나 다운받아 개인 사용자 PC에 설치 할 수 있습니다.  
설치해야 하는 리스트는 다음과 같습니다.

- [Liferay Portal Tomcat 6.2.6](#)
- [Liferay Plugin SDK 6.2.6](#)
- [Liferay IDE](#)

## Liferay SDK와 Portal Tomcat 다운로드

다운로드 링크로 가게 되면 다음과 같은 화면을 확인할 수 있습니다.

The screenshot shows the SourceForge page for the Liferay Portal project. A red box highlights the two most relevant files for this guide:

- Liferay-plugin-sdk-6.2-ce-ga6-20171101150212422.zip**: Modified 2017-11-01, Size 3.7 MB, Downloads 31
- Liferay-portal-tomcat-6.2-ce-ga6-20160112152609836.zip**: Modified 2016-01-14, Size 301.0 MB, Downloads 79

A blue callout box on the right side of the screenshot contains the following text:

다운로드  
Liferay Plugin SDK 6.2.6  
Liferay Portal tomcat 6.2.6

현재 EDISON 플랫폼에서 사용하고 있는 워크벤치 기반 전후처리기를 개발하기 위해서는 서로 버전을 통일해야 합니다. 따라서 현재 EDISON 플랫폼에서 사용중인 **Lifeary Plugin SDK 6.2.6** 버전과 **Liferay Portal tomcat 6.2.6** 버전을 다운받아 사용합니다.

### 포틀릿 개발을 위한 개발 도구 다운로드

Liferay IDE 다운로드 링크를 따라가면 아래 그림과 같은 화면으로 이동하게 됩니다.

The screenshot shows the Liferay Portal's 'Files' section. At the top, there are links for 'Download Latest Version' (LiferayProjectSDK-2018.11.4-windows-installer.exe (267 MB)), 'Get Updates', and navigation links for 'Home / Liferay IDE / 3.4.0'. Below this is a table listing files:

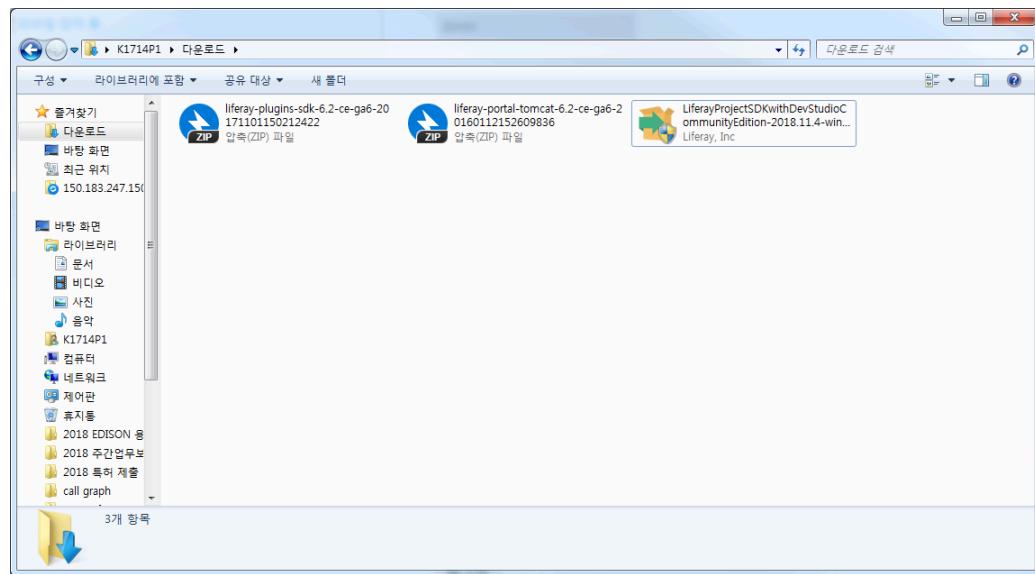
Name	Modified	Size	Downloads / Week
<b>Parent folder</b>			
updateSite	2018-11-06	5	
liferay-ide-updatesite-3.4.0.201811020125.zip	2018-11-06	323.7 MB	64
LiferayProjectSDKwithDevStudioCommunityEdition-2018.11.4-wind...	2018-11-06	905.0 MB	79
LiferayProjectSDKwithDevStudioCommunityEdition-2018.11.4-osx-i...	2018-11-06	906.3 MB	14
LiferayProjectSDKwithDevStudioCommunityEdition-2018.11.4-linux...	2018-11-06	910.0 MB	20
LiferayProjectSDK-2018.11.4-windows-installer.exe	2018-11-06	20.7 MB	285
LiferayProjectSDK-2018.11.4-osx-installer.dmg	2018-11-06	19.6 MB	39
LiferayProjectSDK-2018.11.4-linux-x64-installer.run	2018-11-06	21.0 MB	41
<b>Totals: 8 Items</b>		<b>3.1 GB</b>	<b>547</b>

A blue box on the right contains the Korean text '운영체제에 맞는 IDE 다운로드'.

위의 그림과 같이 Liferay IDE는 윈도우 운영체제, 맥운영체제, 리눅스 운영체제를 지원하고 있습니다. 현재 사용중인 운영체제에 맞는 IDE를 다운받아 사용하면 됩니다.

### 다운로드 파일 확인

현재 다운로드 받은 세 개의 파일을 확인하면 다음 그림과 같습니다.



윈도우 운영체제를 기반으로 설치를 진행하는 화면입니다. 정상적인 파일을 다운받게 되었으면, SDK, tomcat, IDE 설치 파일 세 개의 파일을 다운 받으셔야 합니다.

## JDK 1.7 설치

Liferay 6.2 버전에서 원활한 개발을 위해서는 JDK 1.7 버전에서의 실행 환경을 구성해야 합니다. JDK 설치 파일을 확인하고 운영체제에 맞게 다운받아 설치하시기 바랍니다.

- [JDK 1.7](#)

## ANT 1.9 설치

Liferay 6.2 버전에서 개발된 포틀릿을 웹 포털에 설치(deploy) 하기 위해서는 Ant 버전 1.9를 설치해야 합니다. 해당 ant 1.9를 운영체제에 맞게 다운받아 설치하시기 바랍니다. JDK 설치 파일을 확인하고 운영체제에 맞게 다운받아 설치하시기 바랍니다.

- [ANT 1.9](#)

모든 파일의 다운로드 및 설치가 끝나면 해당 설정하는 방법은 다음 매뉴얼을 참고하시기 바랍니다.

# Liferay IDE 설치

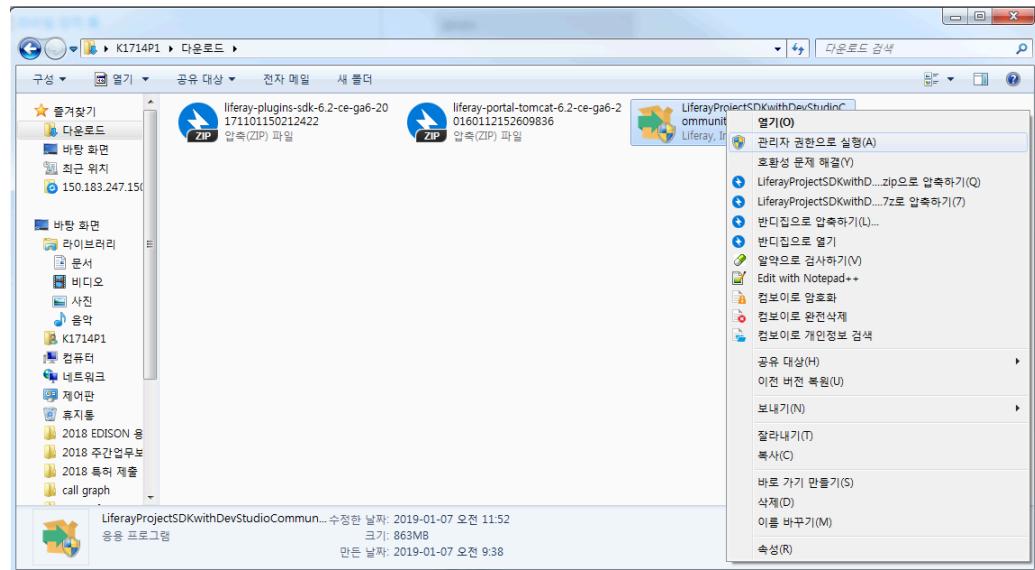
## Liferay IDE 설치

Liferay 기반 개발을 진행하기 위해서는 통합 개발 환경인 IDE(Integrated Development Environment, IDE)를 의미하며, Liferay 개발자를 위해 liferay 플랫폼을 기반으로 하는 개발 환경을 제공합니다. 해당 IDE는 Liferay에서 이클립스를 기반으로 개발 환경을 구축해 놓은 파일을 다운받아 설정을 하고 개발환경을 구성합니다.

이전 매뉴얼에서 확인했듯이 [\[Liferay IDE\] \(<https://sourceforge.net/projects/lportal/files/Liferay%20IDE/3.4.0/>\)](https://sourceforge.net/projects/lportal/files/Liferay%20IDE/3.4.0/)에서 파일을 다운받아 설치합니다. 윈도우 운영체제를 기반으로 설치 방법을 진행하겠습니다.

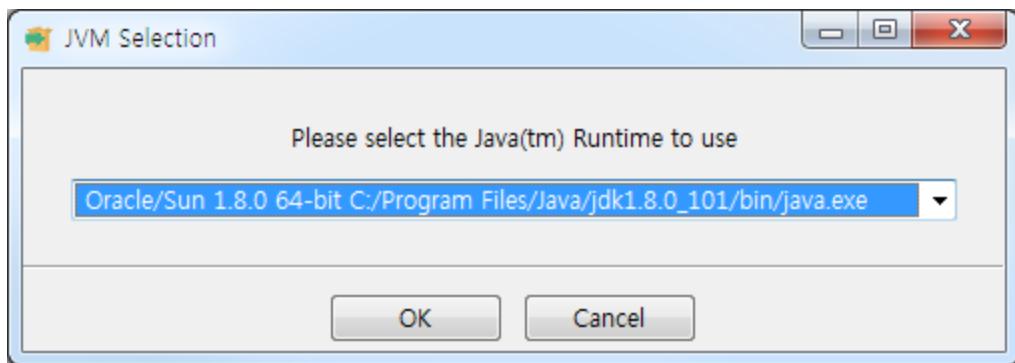
### 1. 관리자 권한으로 실행

다운로드 받은 IDE 설치 파일을 마우스 오른쪽 클릭을 하여 관리자 권한으로 실행합니다.



### 2. JDK 1.8 기반으로 실행

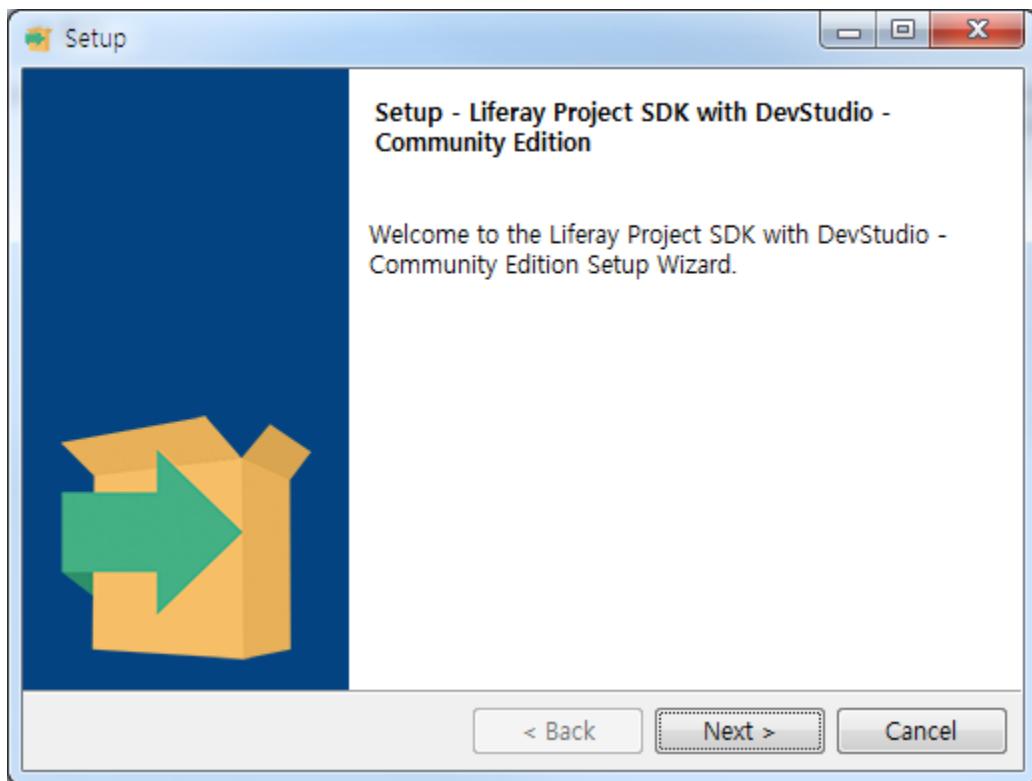
개발 환경 설치를 시작하면 설치되는 IDE의 실행 환경인 JVM의 버전을 설치합니다.



실제 개발 환경을 설정할 때에는 개발 IDE 내부의 개발 환경을 JDK 1.7 버전으로 수정하게 됩니다.

### 3. 설치 내용

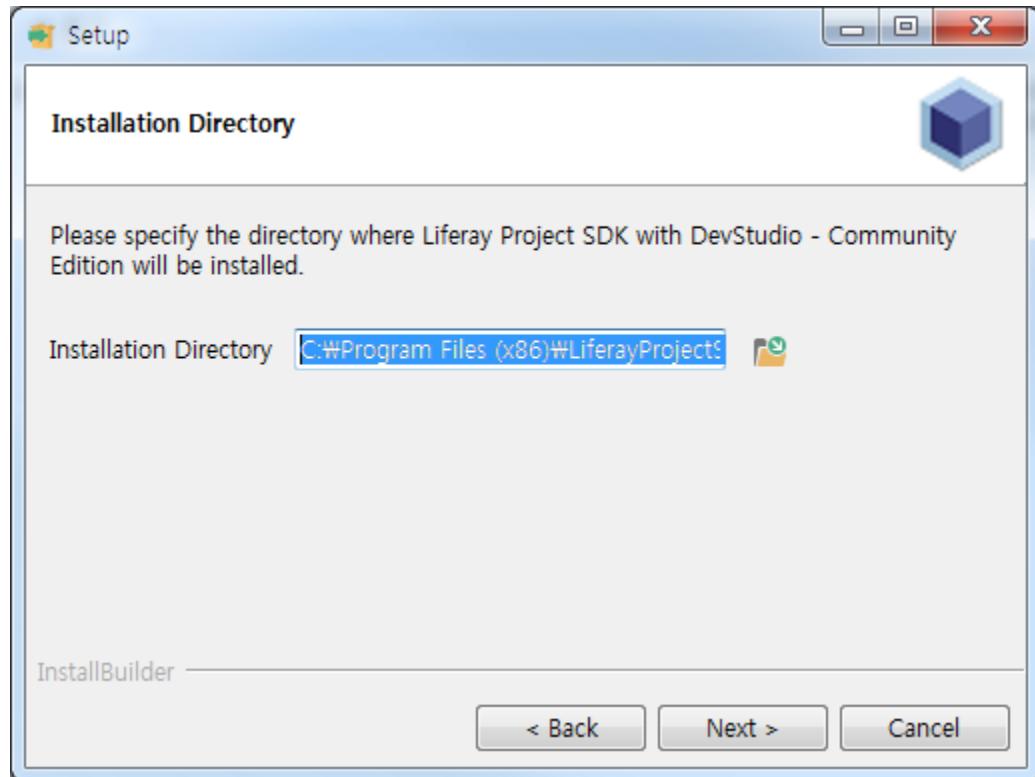
설치를 시작하면 아래 그림과 같은 화면을 확인할 수 있습니다. Nxet를 선택합니다.



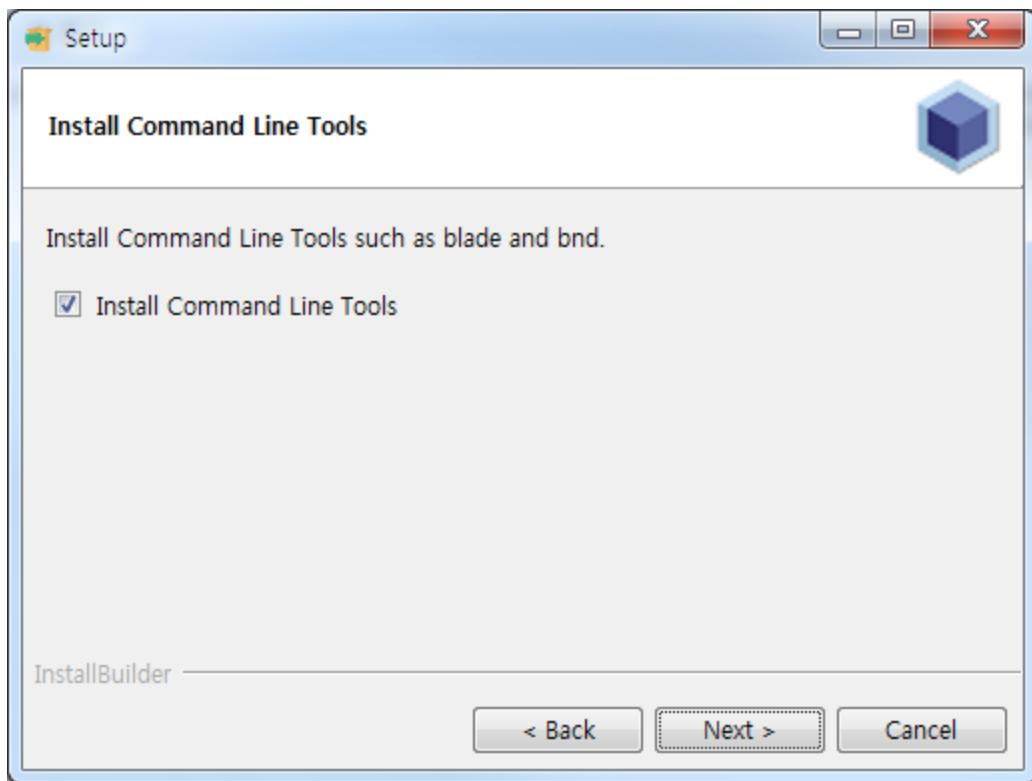
다음으로는 설치할 위치를 설정합니다. 기본적으로 윈도우 운영체제에서는

C:\Program Files

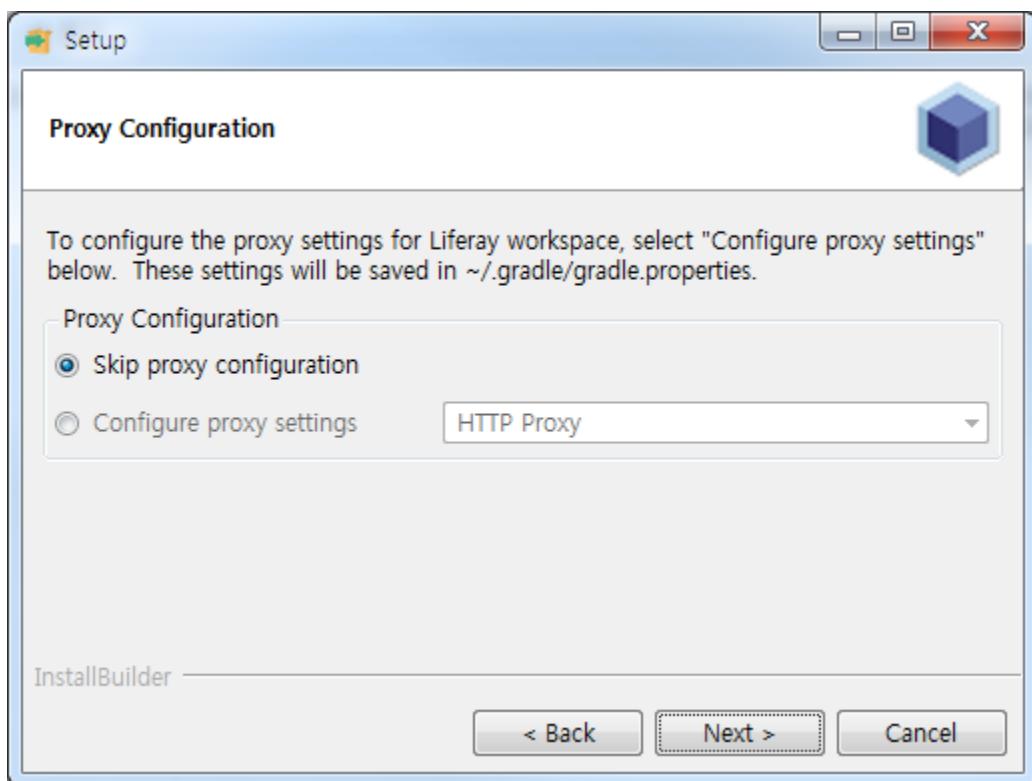
(x86)\LiferayProjectSDKwithDevStudioCommunityEdition 폴더가 기본으로 설정되어 있습니다. 크게 변경되는 내용이 없으면 바로 Nxet 버튼을 선택합니다.



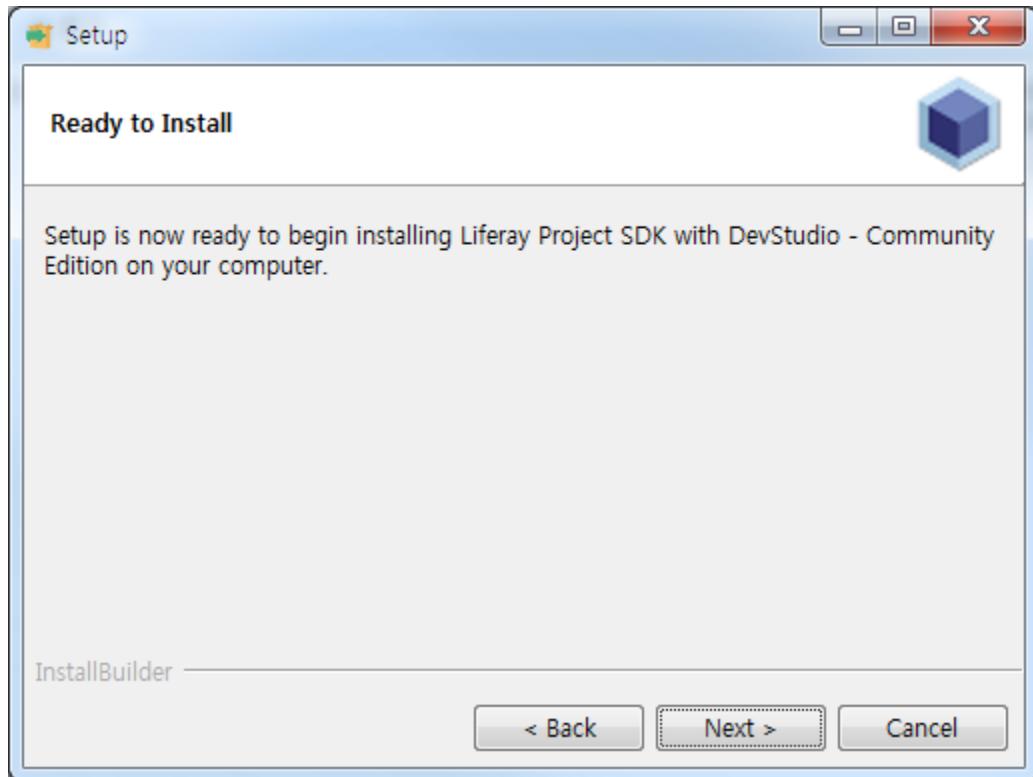
Install Command Like Tools 체크를 한 상태로 Next 버튼을 선택합니다.



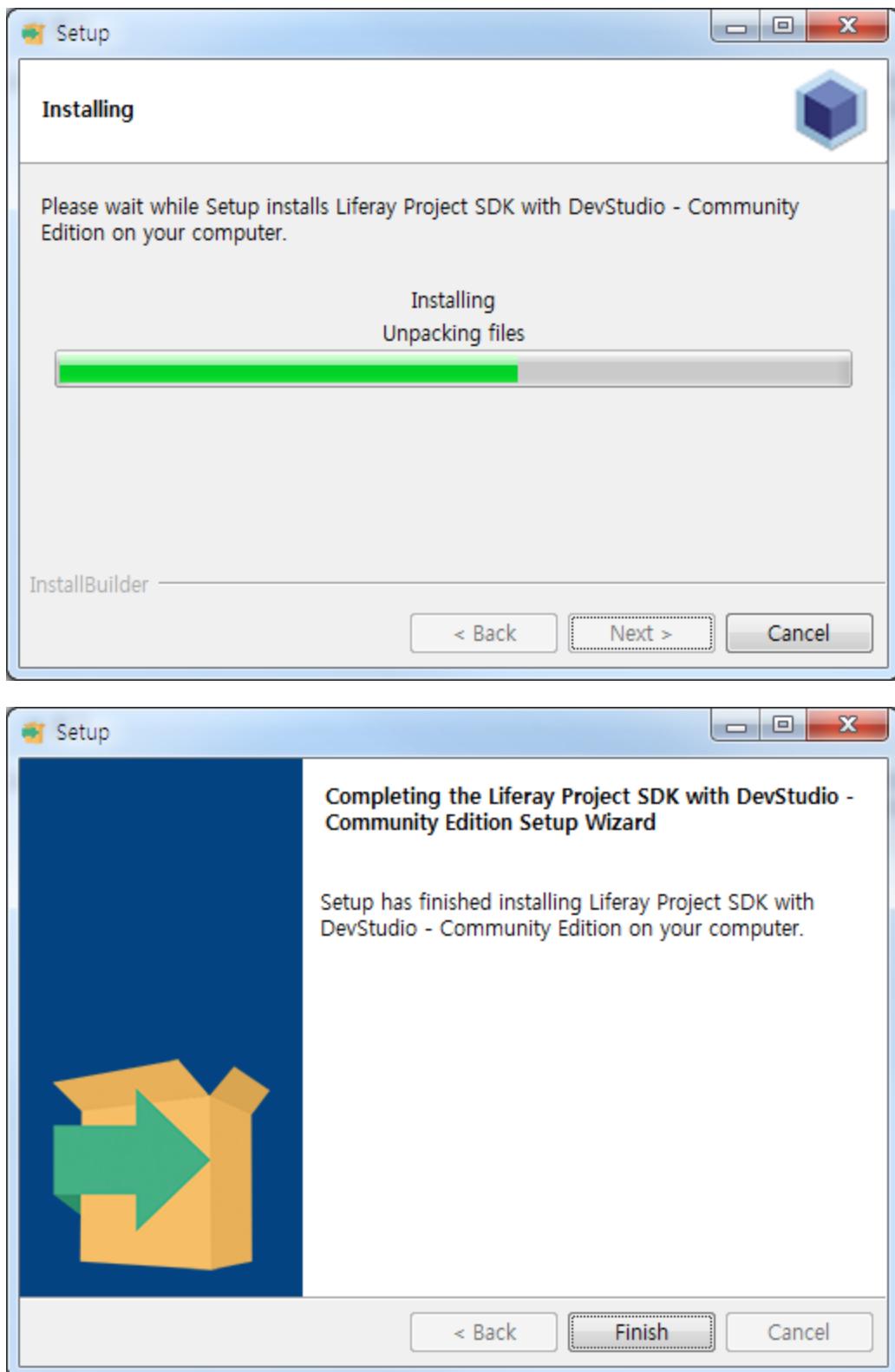
Skip proxxt configuration 체크를 한 상태로 Next 버튼을 선택합니다.



다음으로 모든 설치 준비가 완료되었다는 화면을 확인할 수 있습니다. 다시 설정하고 싶은 내용이 없으면 Next 버튼을 선택합니다.



설치하고 완료되면 아래와 같은 화면을 확인할 수 있습니다.



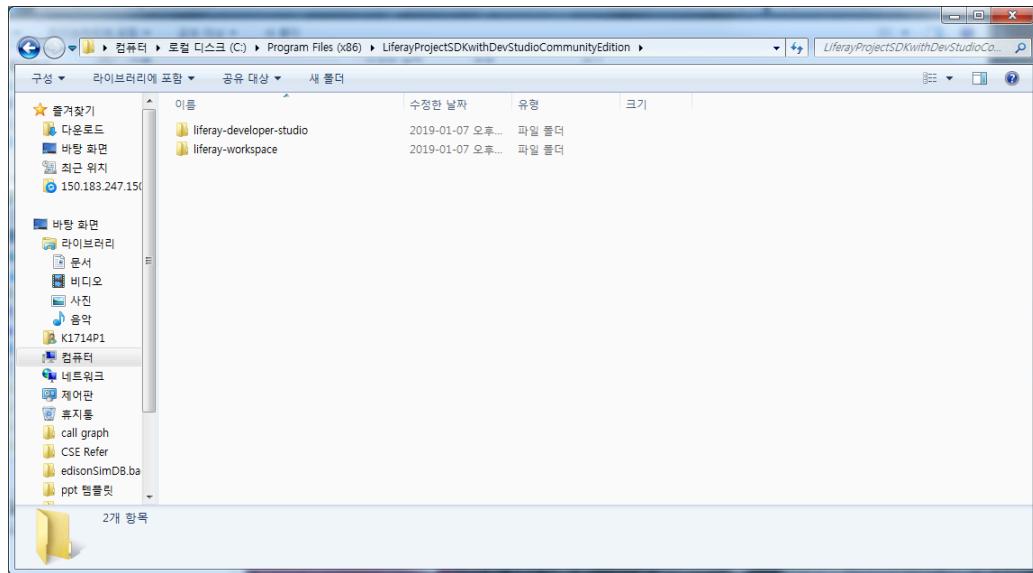
위와 같이 설치가 완료되면 Finish 버튼을 선택하여 설치를 종료합니다.

#### 4. 설치 확인

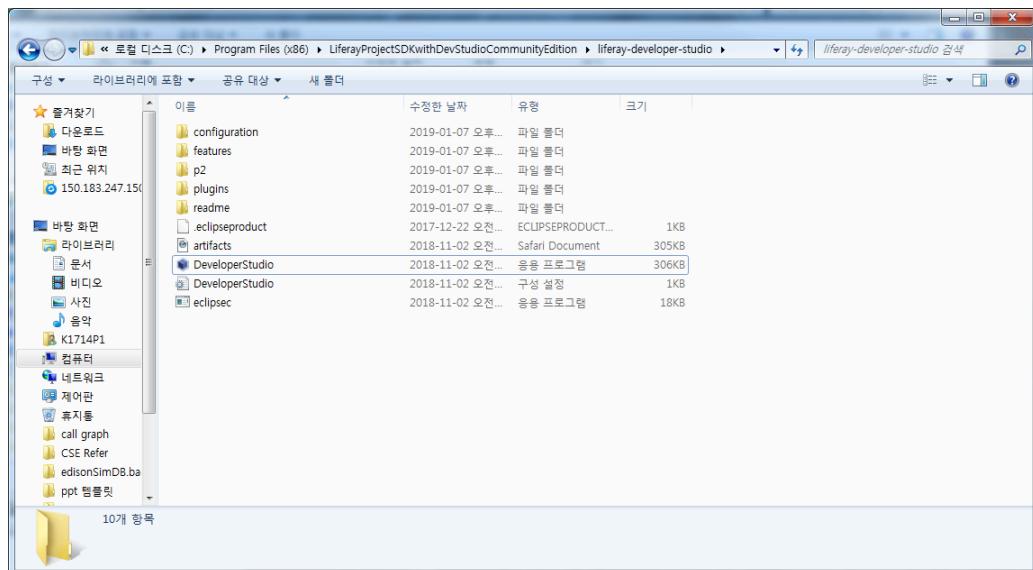
실제 설치된 위치인 **C:\Program Files**

**(x86)\LiferayProjectSDKwithDevStudioCommunityEdition** 폴더에 가서 설치

내용을 확인하면 아래 그림과 같습니다.



다운받은 Liferay IDE 설치 파일을 이용하여 설치를 진행하면 위의 그림과 같이 두개의 폴더가 설치된 것을 확인할 수 있습니다. 두 개의 폴더 중 IDE 환경은 **liferay-developer-studio** 의 파일에 설치되어 있습니다. 해당 폴더를 들어가면 아래와 같은 설치된 파일들을 확인할 수 있습니다.



## IDE 실행 환경 설정

처음 실행 환경을 설정어할 때, 기본 IDE 파일 실행 메모리가 매우 작아 원활한 실행 환경 설정을 위해 실행 환경 값을 변경합니다. 폴더 내 파일중 **DeveloperStudio.ini** 파일의 내용을 수정합니다.

- 기본 환경 설정

처음 기본으로 설정된 구성 환경 파일은 아래와 같습니다.

```
-startup  
plugins/org.eclipse.equinox.launcher_1.4.0.v20161219-135  
6.jar  
--launcher.library  
plugins/org.eclipse.equinox.launcher.win32.win32.x86_6  
4_1.1.551.v20171108-1834  
-product  
com.liferay.ide.studio.ui.product  
--launcher.defaultAction  
openFile  
--launcher.XXMaxPermSize  
384M  
-showsplash  
com.liferay.ide.studio.ui  
-vmargs  
-Dosgi.requiredJavaVersion=1.8  
-Xms40m  
-Xmx1024m  
-Dsun.java2d.noddraw=true
```

- 수정된 환경 설정

수정해야 하는 내용은 아래와 같습니다.

```
-startup  
plugins/org.eclipse.equinox.launcher_1.4.0.v20161219-135  
6.jar  
--launcher.library  
plugins/org.eclipse.equinox.launcher.win32.win32.x86_6  
4_1.1.551.v20171108-1834  
-product  
com.liferay.ide.studio.ui.product  
--launcher.defaultAction  
openFile  
--launcher.XXMaxPermSize  
512M  
-showsplash  
com.liferay.ide.studio.ui  
-vmargs  
-Dosgi.requiredJavaVersion=1.8  
-Xms1024m  
-Xmx2048m  
-Dsun.java2d.noddraw=true
```

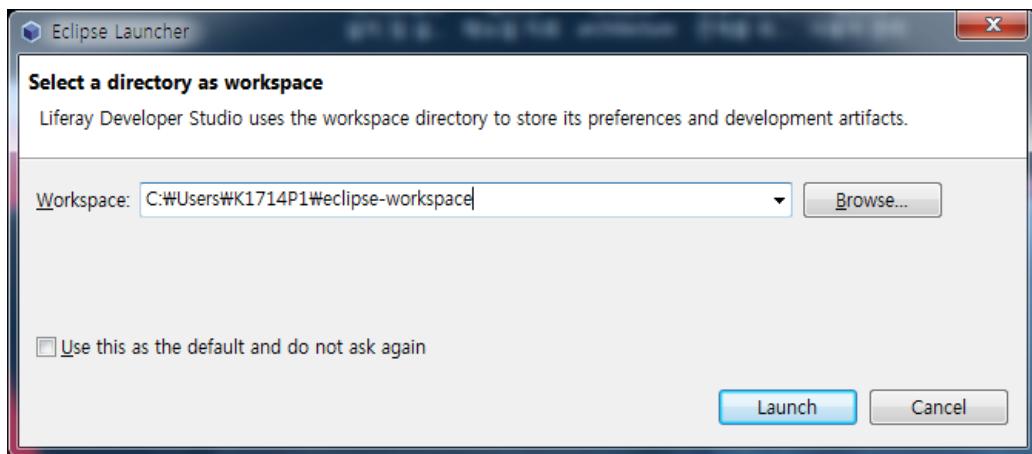
수정되는 내용은 메모리 사용량을 약간 늘리는 내용입니다. 각 컴퓨터의 사양에 맞춰 적당한 크기로 늘려 주시기 바랍니다. **--launcher.XXMaxPermSize 512M**, **-Xms1024m**, **-Xmx2048m**의 내용을 수정하였으며, 원활한 개발을 위한 최소한의 메모리 사용량을 기준으로 수정하였습니다.

# Developer Studio 실행

**Summary:** 본 문서는 워크벤치 기반 전후처리기 개발을 위한 매뉴얼입니다.

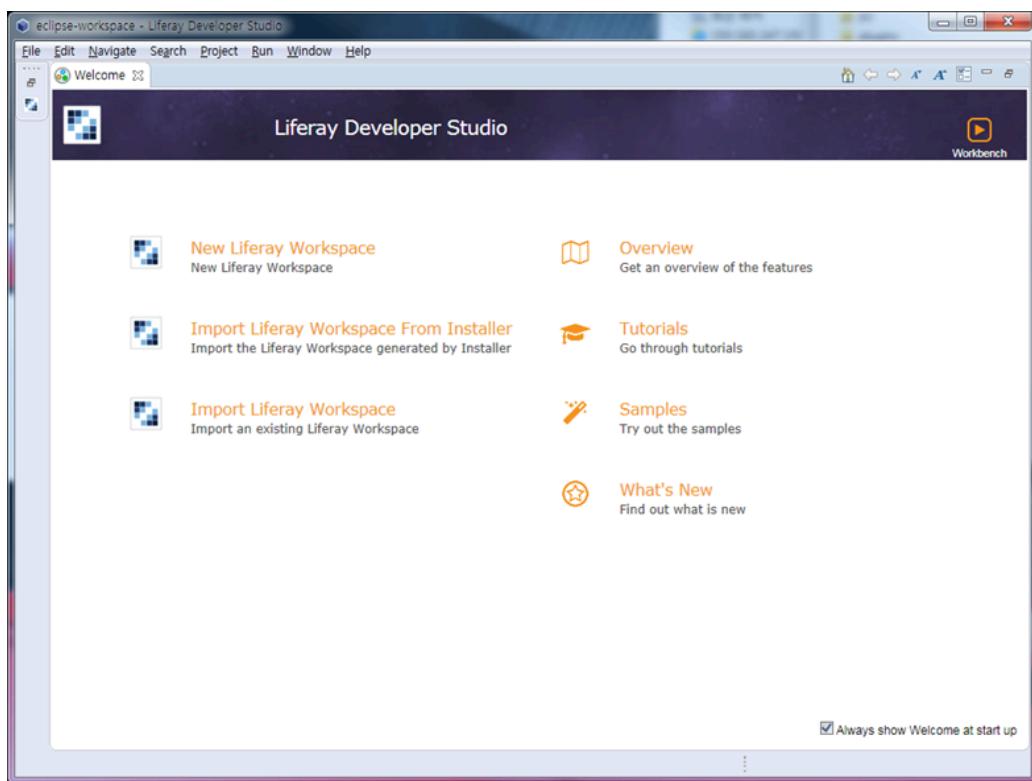
## Developer Studio 실행

이전 매뉴얼에 따라 기본적인 Developer Studio의 설정을 마치고, 다음으로 DeveloperStudio 응용프로그램을 실행시키면 다음과 같은 화면을 확인할 수 있습니다.



개발을 진행할 개발자의 개발 공간(Workspace)의 위치를 지정하는 화면입니다. 기본적으로 설정된 위치가 아니라 특정 위치를 지정하고 싶으면 [Browse...](#) 버튼을 클릭하여 새로운 위치를 지정할 수 있습니다. 특정 Workspace의 위치를 지정하고 [Launch](#) 버튼을 클릭하여 DeveloperStudio를 실행시킬 수 있습니다. 실행 버튼(Launch)을 선택하면 아래 그림과 같이 실행 되는 것을 확인 할 수 있습니다.



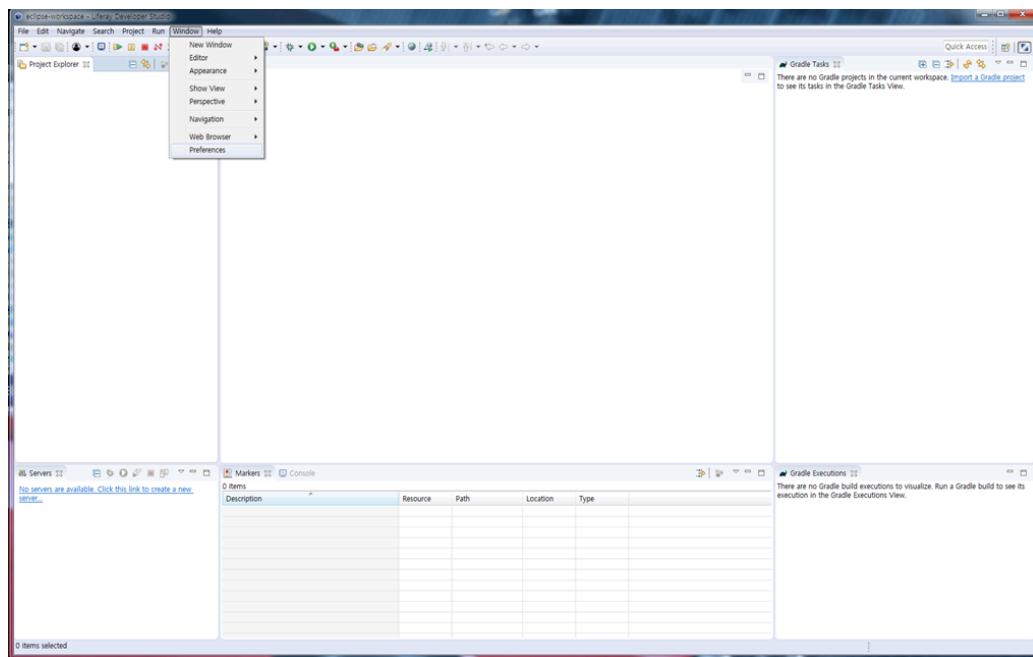


# JDK 1.7 설정

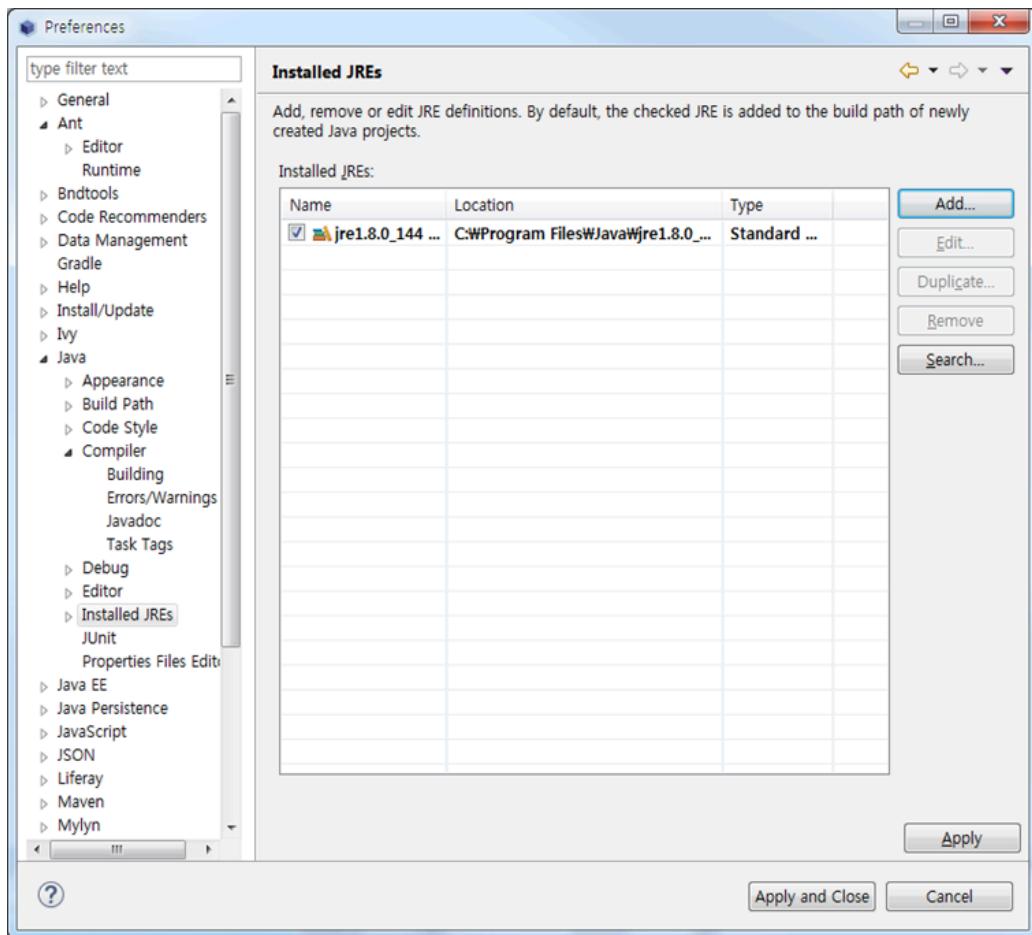
**Summary:** 본 문서는 워크벤치 기반 전후처리기 개발을 위한 매뉴얼입니다.

## JDK 1.7 설정

DeveloperStudio에서 JDK 버전 설정을 위해 상위 메뉴중 **Window** 메뉴를 선택하고 **Preferences** 를 선택하여 Preferences설정 창을 엽니다.

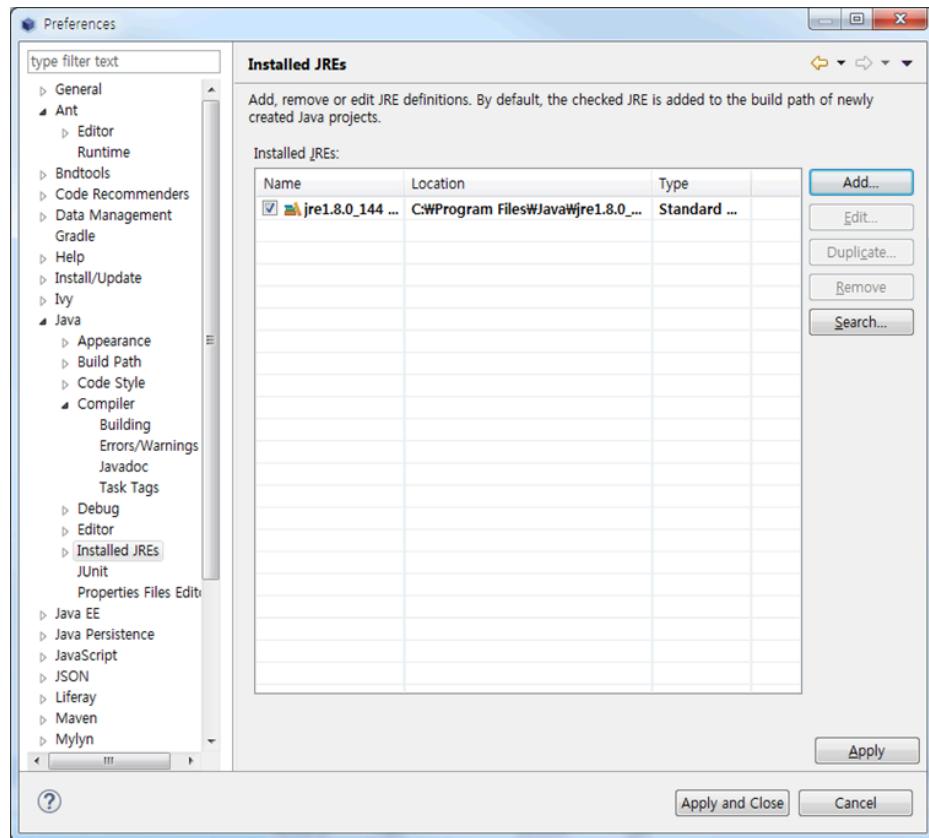


Preferences창에서 **java** 탭 내에 **Installed JREs** 에서 아래 그림과 같이 설정을 합니다.

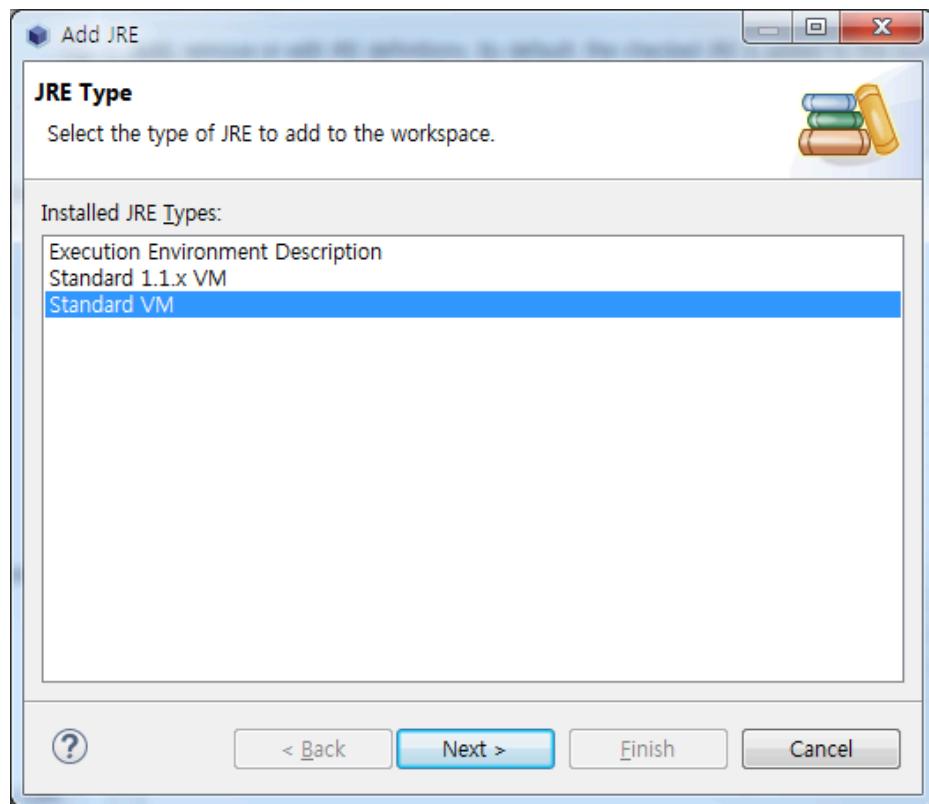


기본적인 셋팅으로 JRE 1.8 버전으로 DeveloperStudio가 구동되게 되어 있습니다. 위 크벤치 기반 전후처리기 개발을 위해서 JDK 1.7 기반으로 설정을 바꿔야 합니다. 해당 내용은 아래와 같습니다.

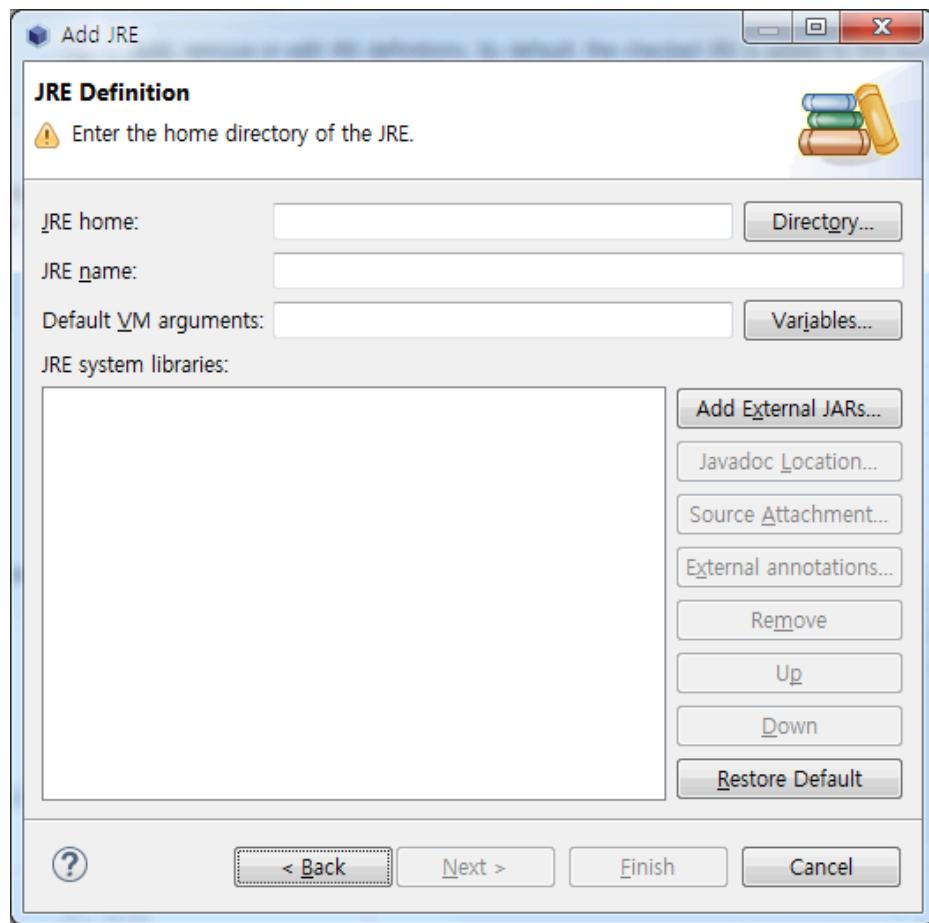
1. Add 버튼 선택



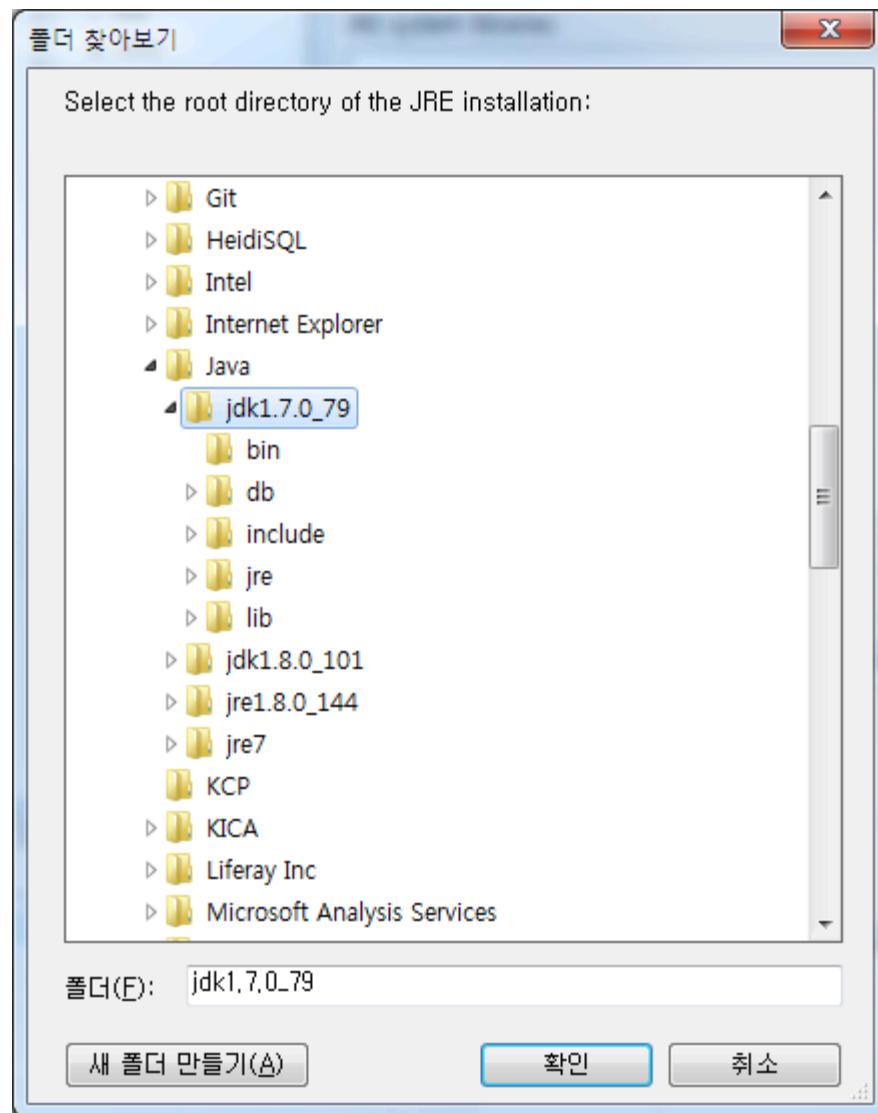
## 2. Standard VM 선택



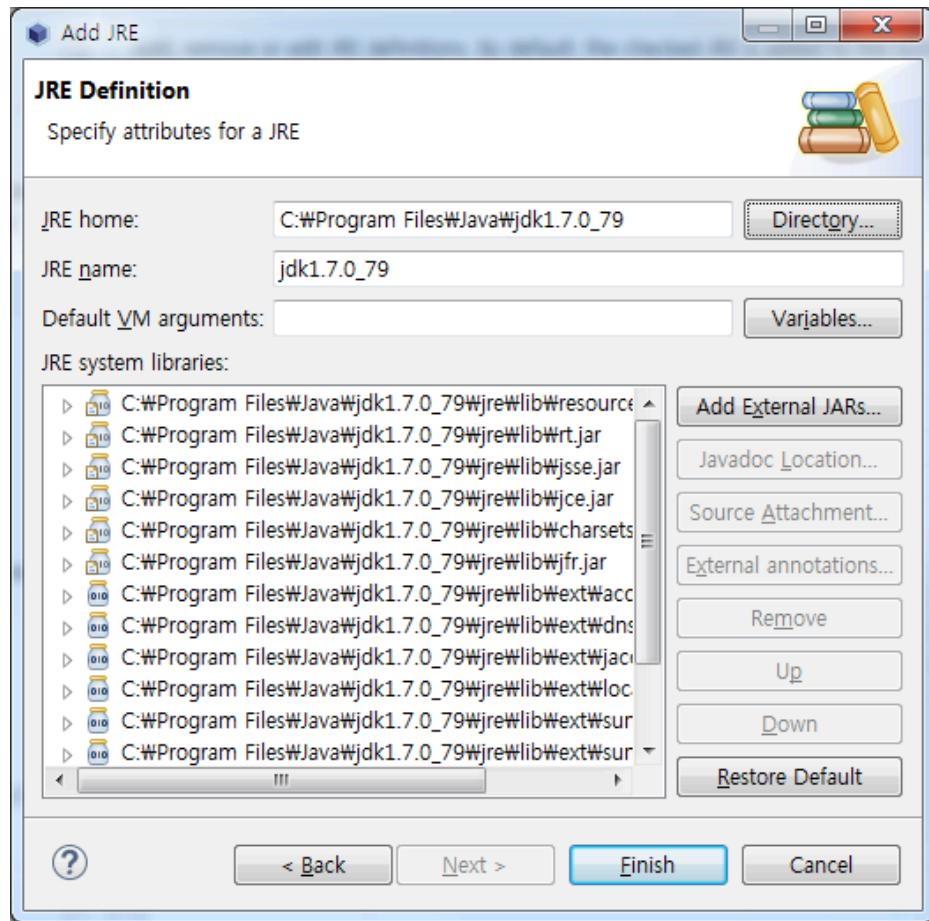
### 3. JRE home Directory 선택



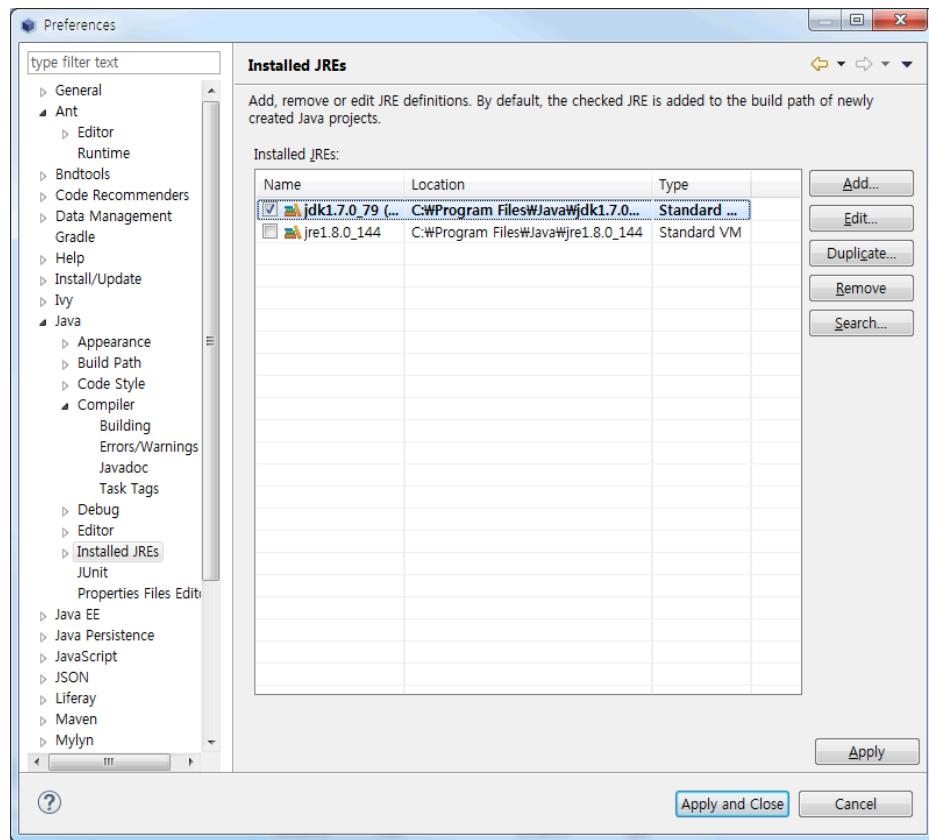
#### 4. JDK 1.7 버전이 설치된 디렉토리 선택



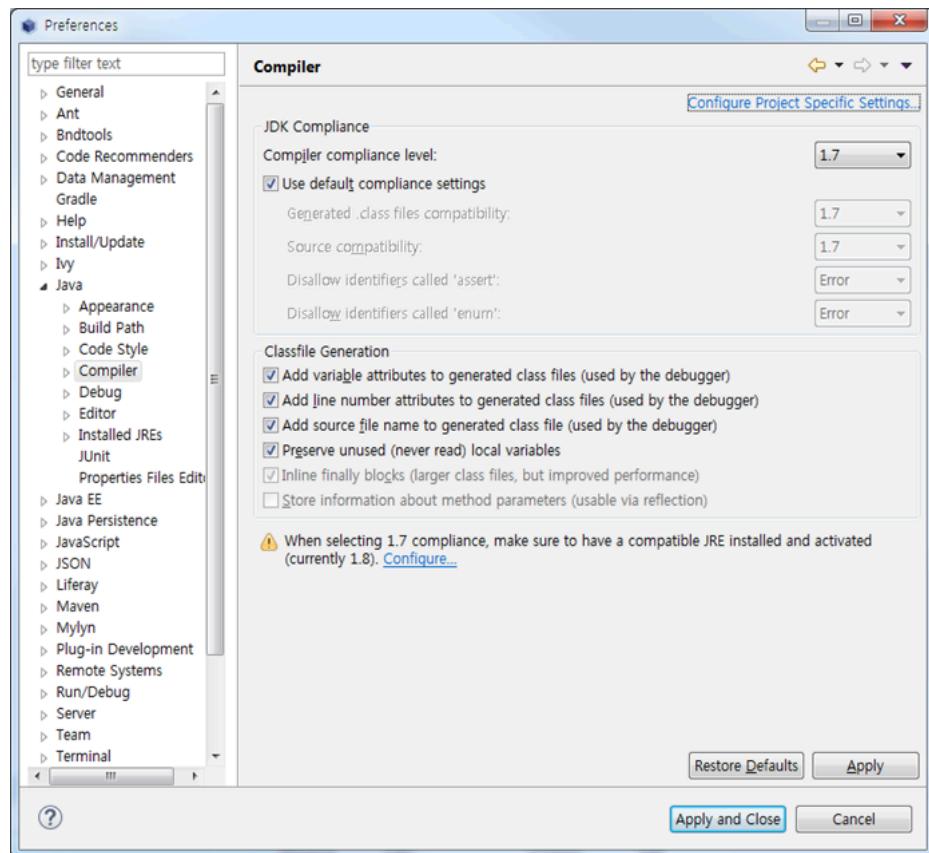
##### 5. JDK 1.7 설정 Finish 선택



6. 설치된 JRE 리스트 중 JDK 1.7 버전 체크 및 Apply 버튼 클릭



7. Java Compiler 탭에서 아래 그림과 같이 설정

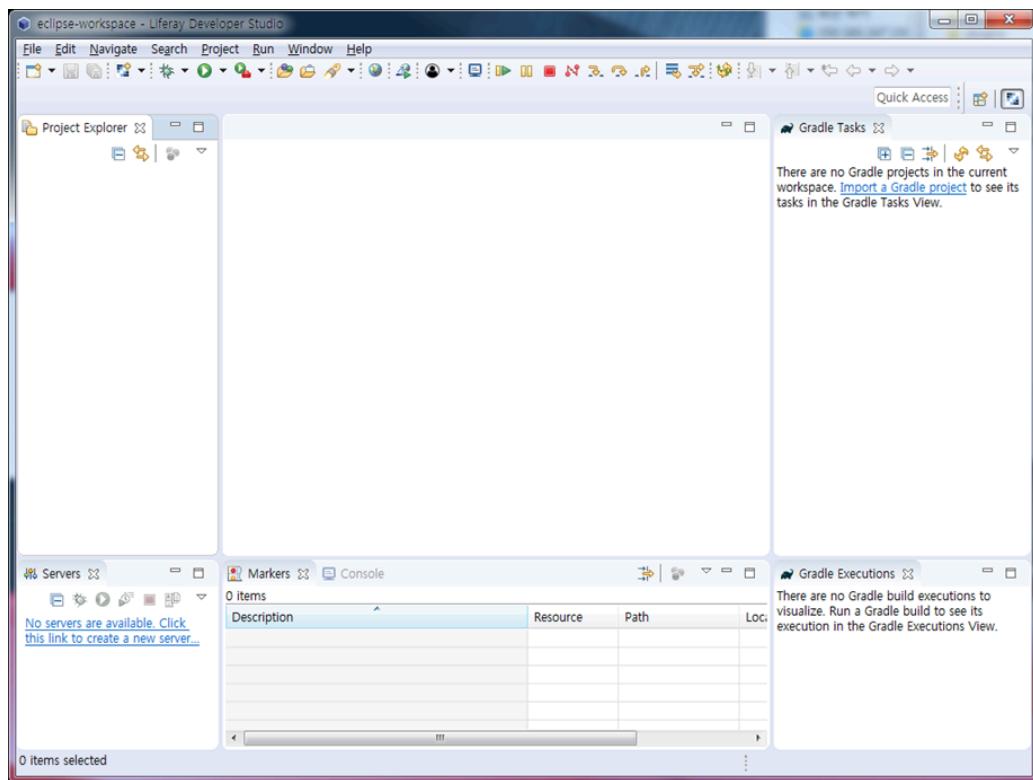


# tomcat 서버 설정

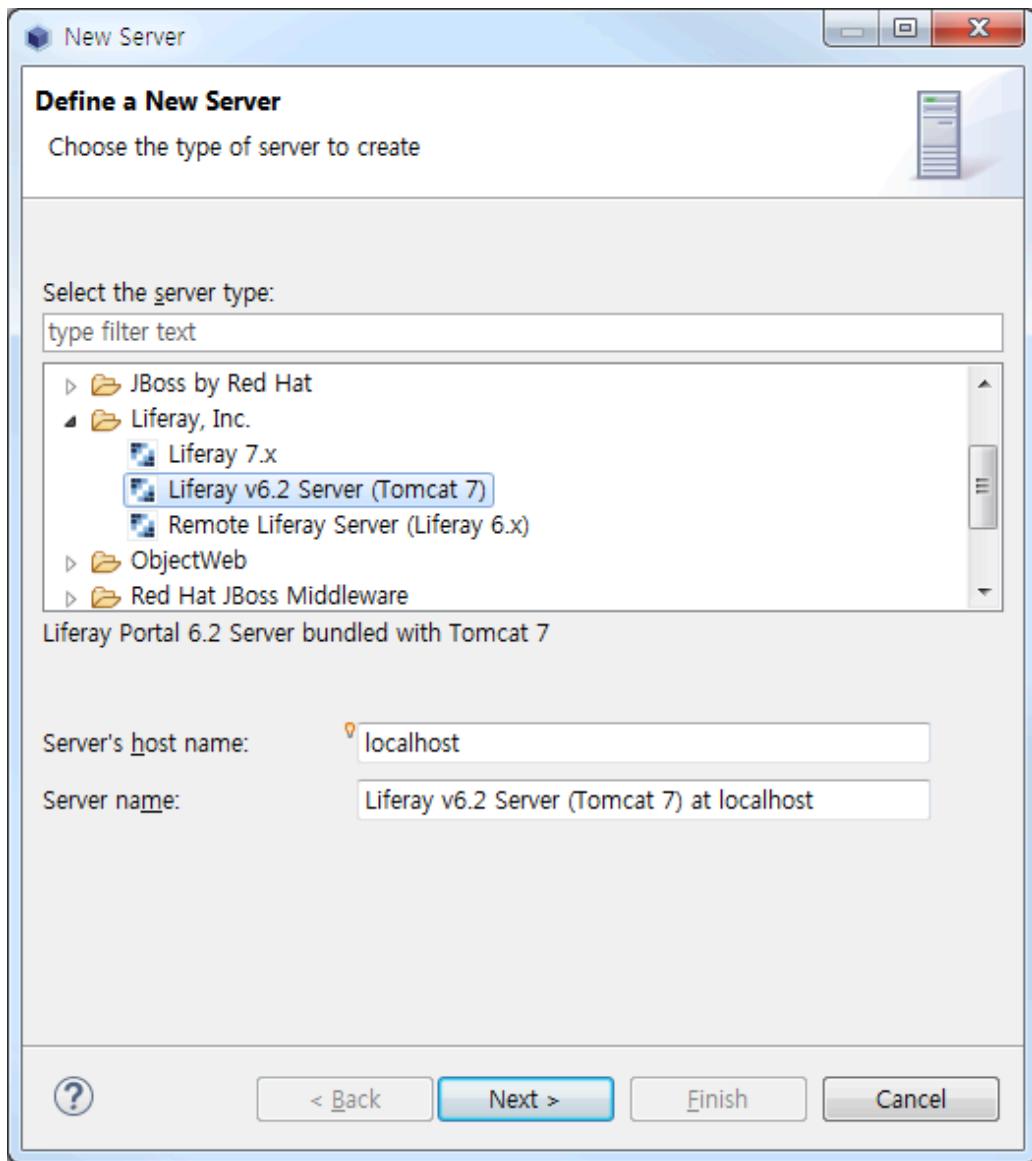
**Summary:** 본 문서는 워크벤치 기반 전후처리기 개발을 위한 매뉴얼입니다.

## tomcat 서버 생성

이전 설정이 끝나고 다운 받은 Liferay 6.2 버전의 서버를 현재 DeveloperStudio와 연결해야 합니다. 아래의 그림과 같은 화면에서 아무런 서버가 연결되어 있지 않다면 **No servers are available. Click this link to create a new server...** 의 문구를 왼쪽 하단부에서 확인할 수 있습니다.



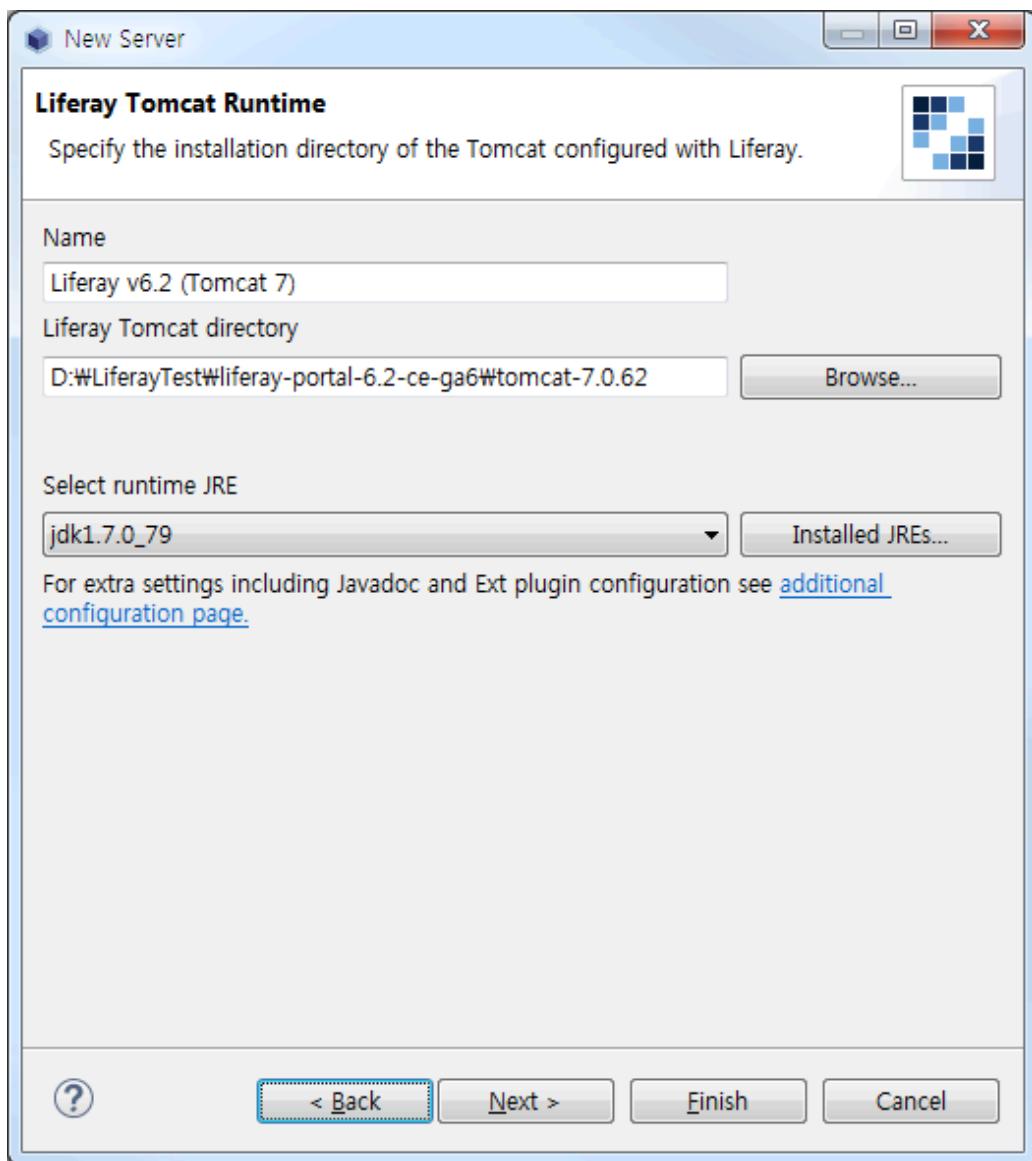
해당 링크를 선택하면 새로운 서버를 연결시킬 수 있는 팝업 화면이 뜨게 됩니다.

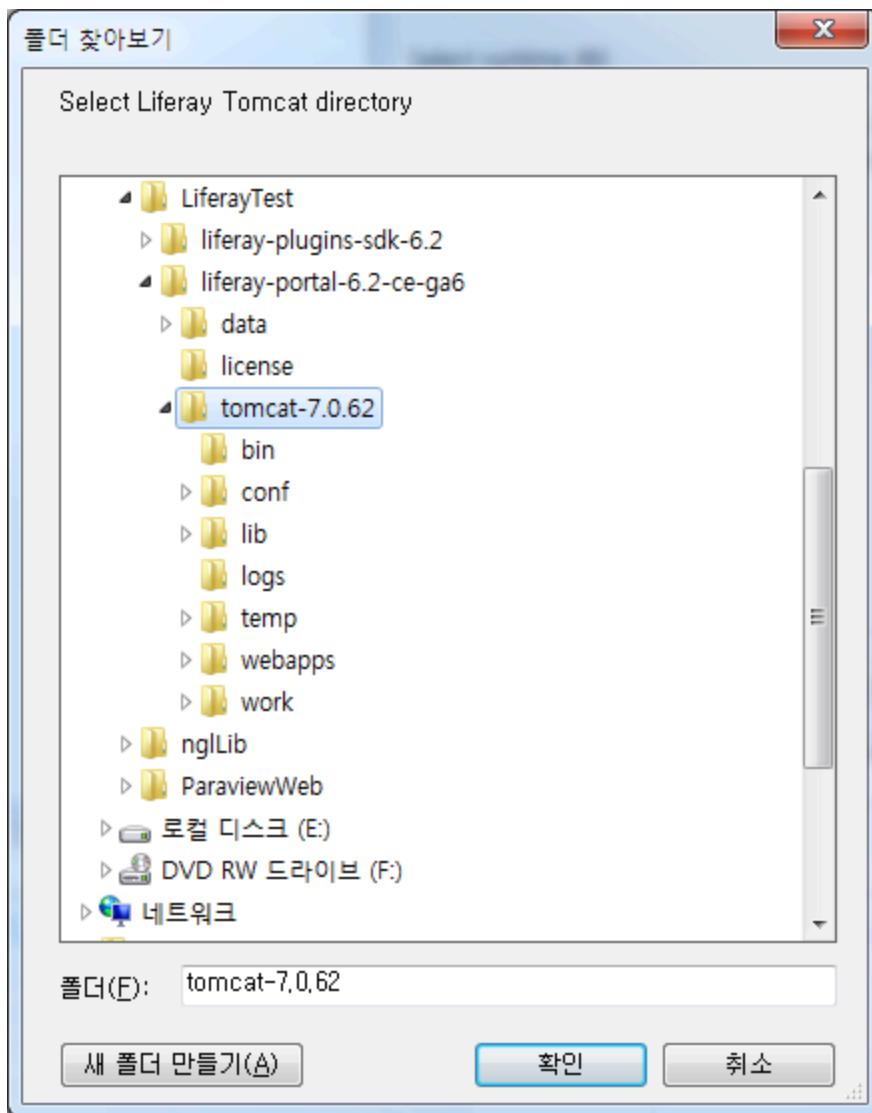


위의 그림에서와 마찬가지로 EDISON 플랫폼 기반 전후처리기를 개발하기 위해서는 Liferay 6.2 버전의 서버를 설치해야 합니다. 따라서 **Liferay v6.2 Server (Tomcat 7)** 을 선택하고 Next 버튼을 선택합니다.

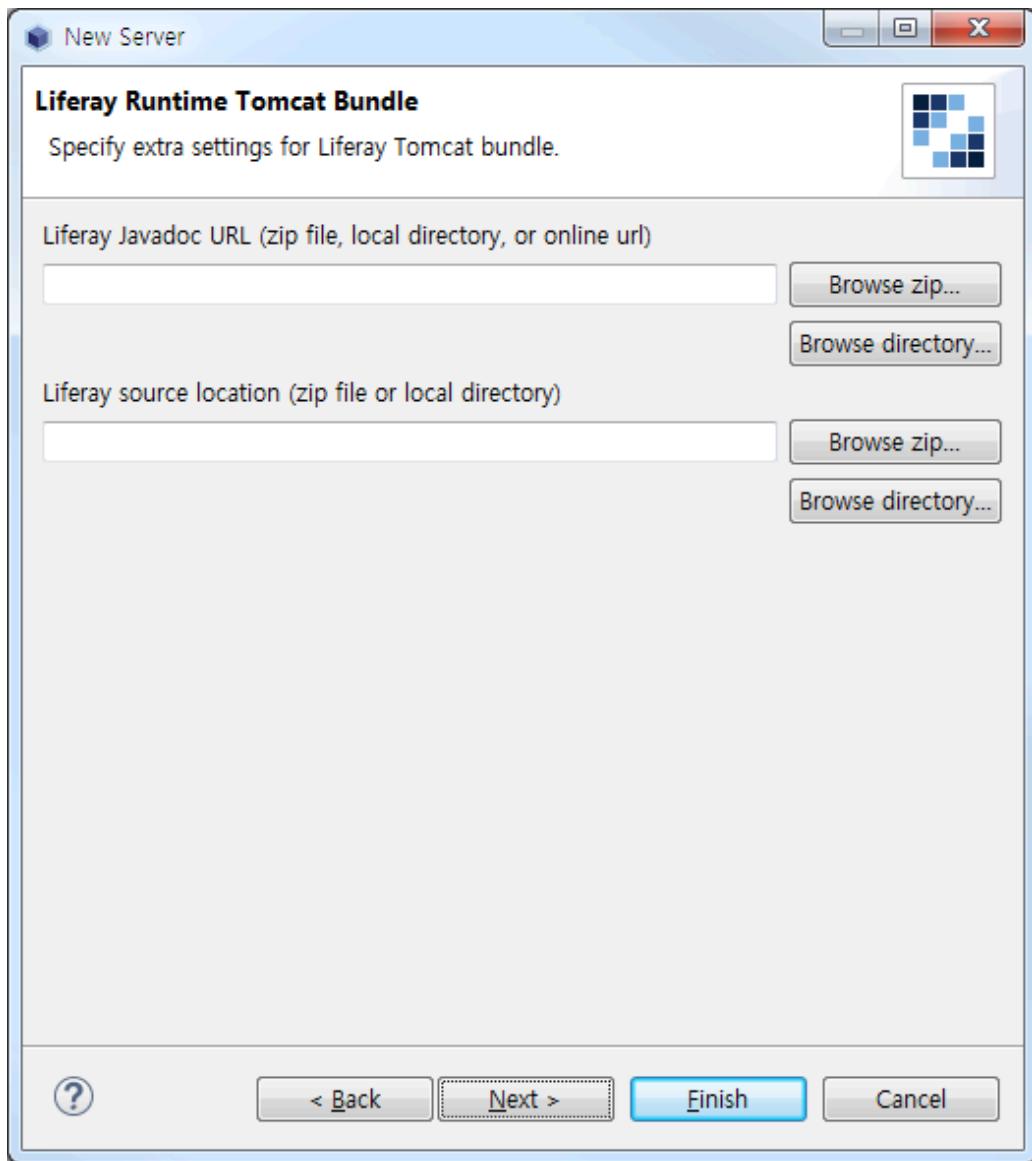
#### tomcat 디렉토리 연결

미리 다운받아놓은 파일 중에서 **liferay-portal-6.2-ce-ga6** 의 압축을 풀면 해당 폴더 내에 포털(서버) 실행을 위한 파일들이 존재합니다. 해당 폴더 내 **tomcat-7.0.62** 폴더를 선택하여 Liferay Tomcat directory로 설정합니다. 또한 서버 실행을 위한 자바 환경 파일은 이전에 설치되어 있는 jdk 1.7 버전인 **jdk1.7.0\_79** 을 선택합니다. 해당 설정이 모두 끝나면 Next 버튼을 선택합니다.

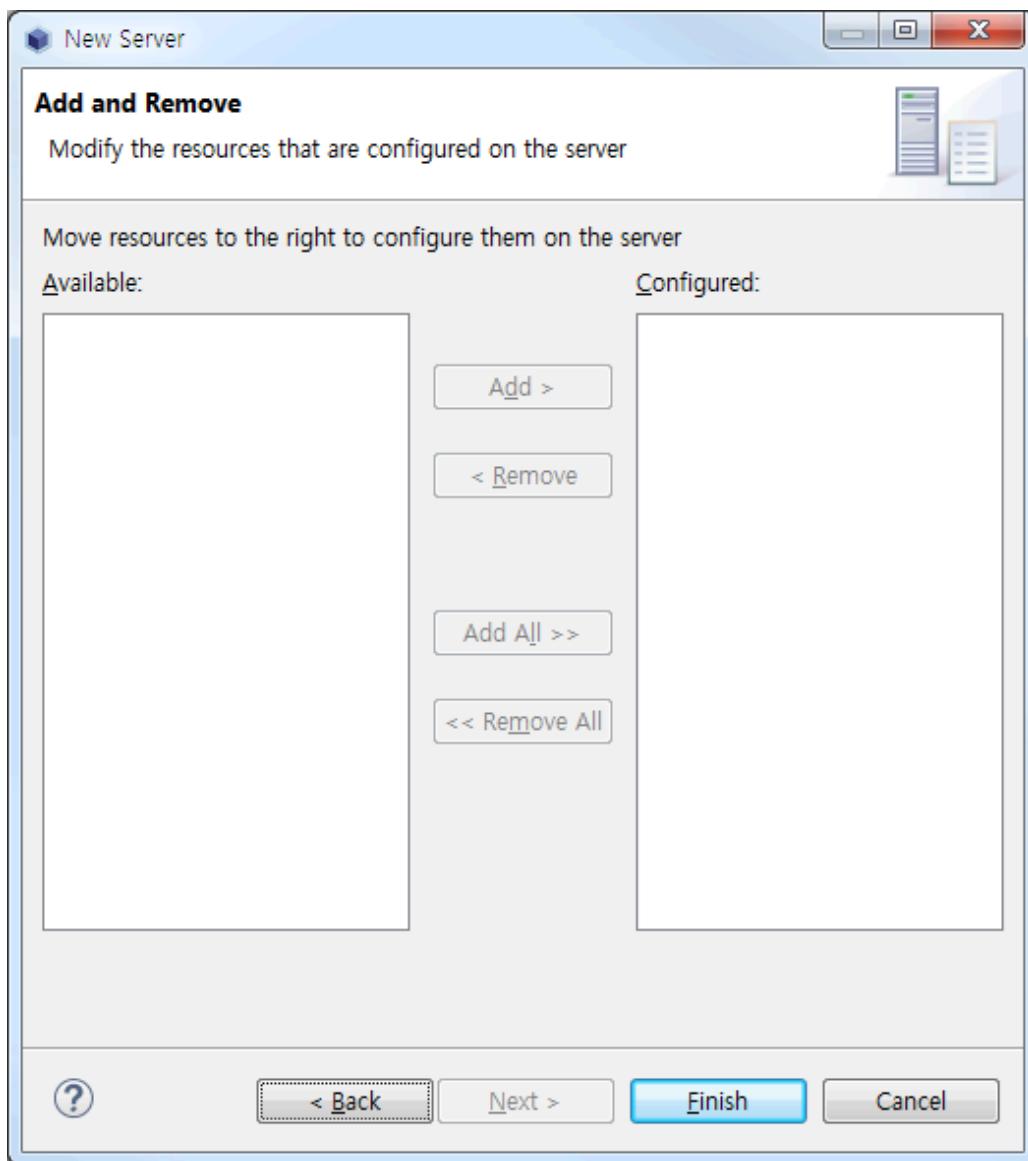




크게 설정할 내용은 아직 없으므로 Next 버튼을 선택합니다.

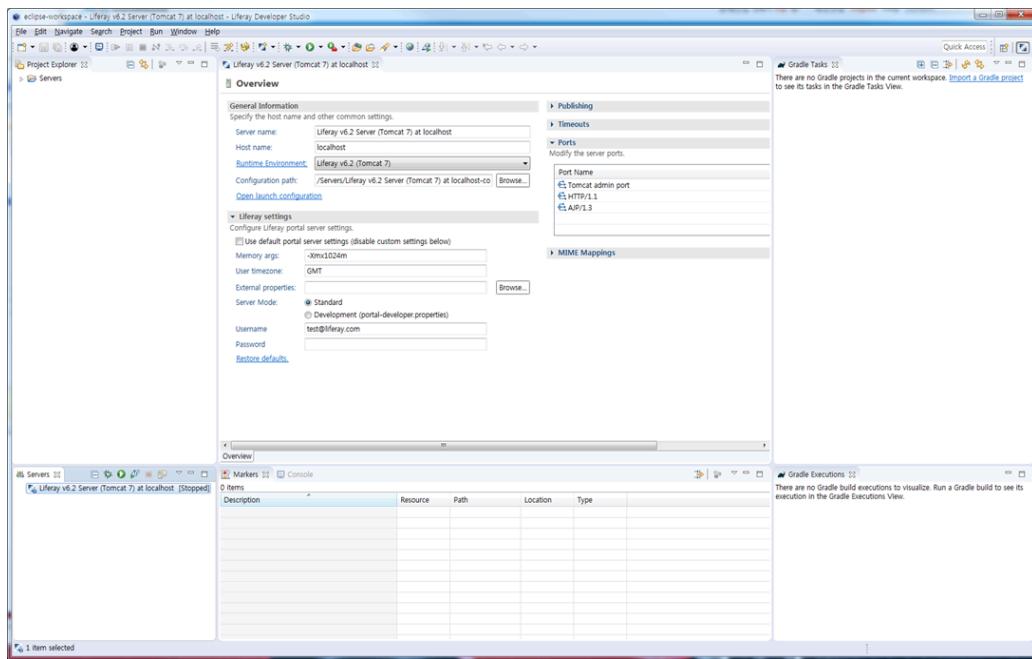


따로 설정할 내용은 없으므로 Finish 버튼을 선택하여 서버 생성을 종료합니다.



### 서버 Overview 설정

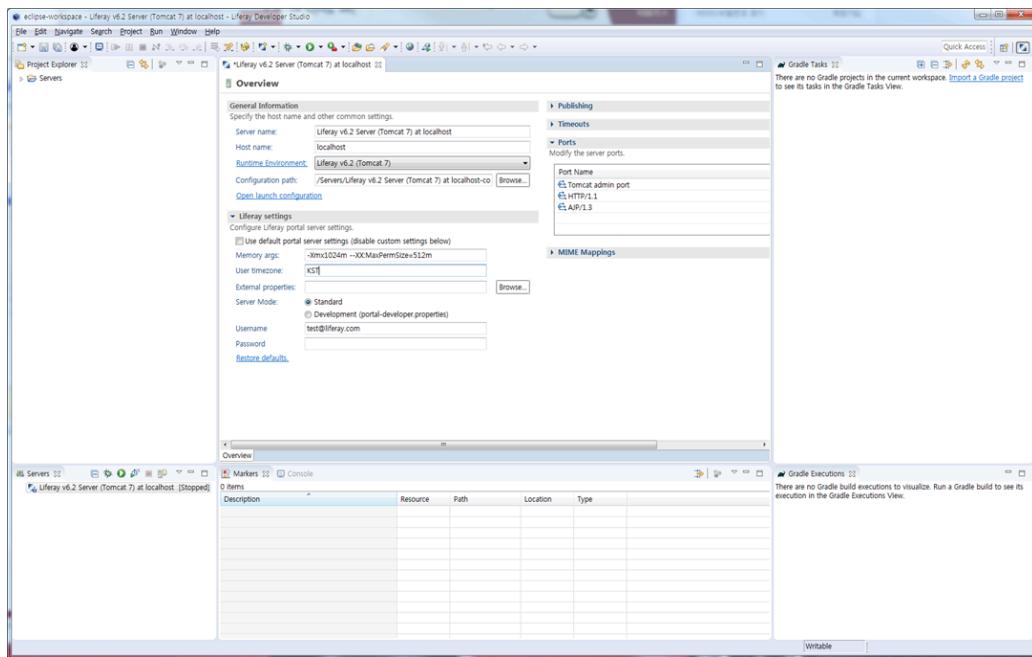
서버 생성을 종료하면 아래 그림과 같이 서버가 생성된 화면을 확인할 수 있습니다.



그림에서 왼쪽 하단부 Server 부분의 생성된 Liferay v6.2의 서버를 더블 클릭하면 현재 생성된 서버의 오버뷰(Overview)를 확인할 수 있습니다. 오버뷰에서 수정해야 하는 내용은 다음과 같습니다.

수정 내용	수정 전	수정 후
Memory args	-Xmx1024m	-Xmx1024m -XX:MaxPermSize=512m
User timezone	GMT	KST

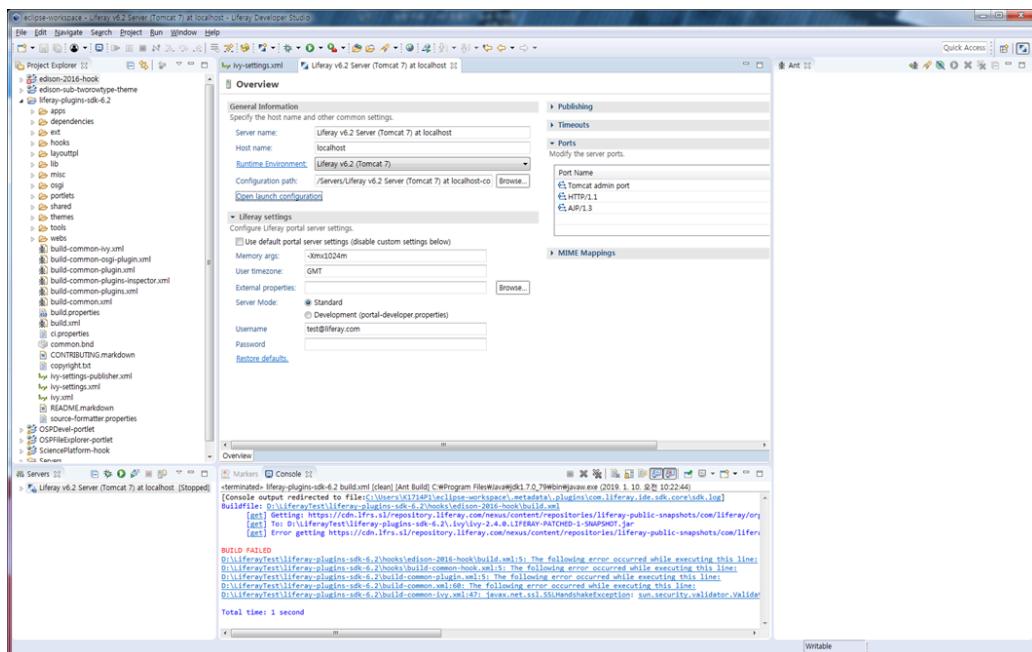
수정이 완료된 후 서버 Overview 내용은 다음 그림과 같습니다.

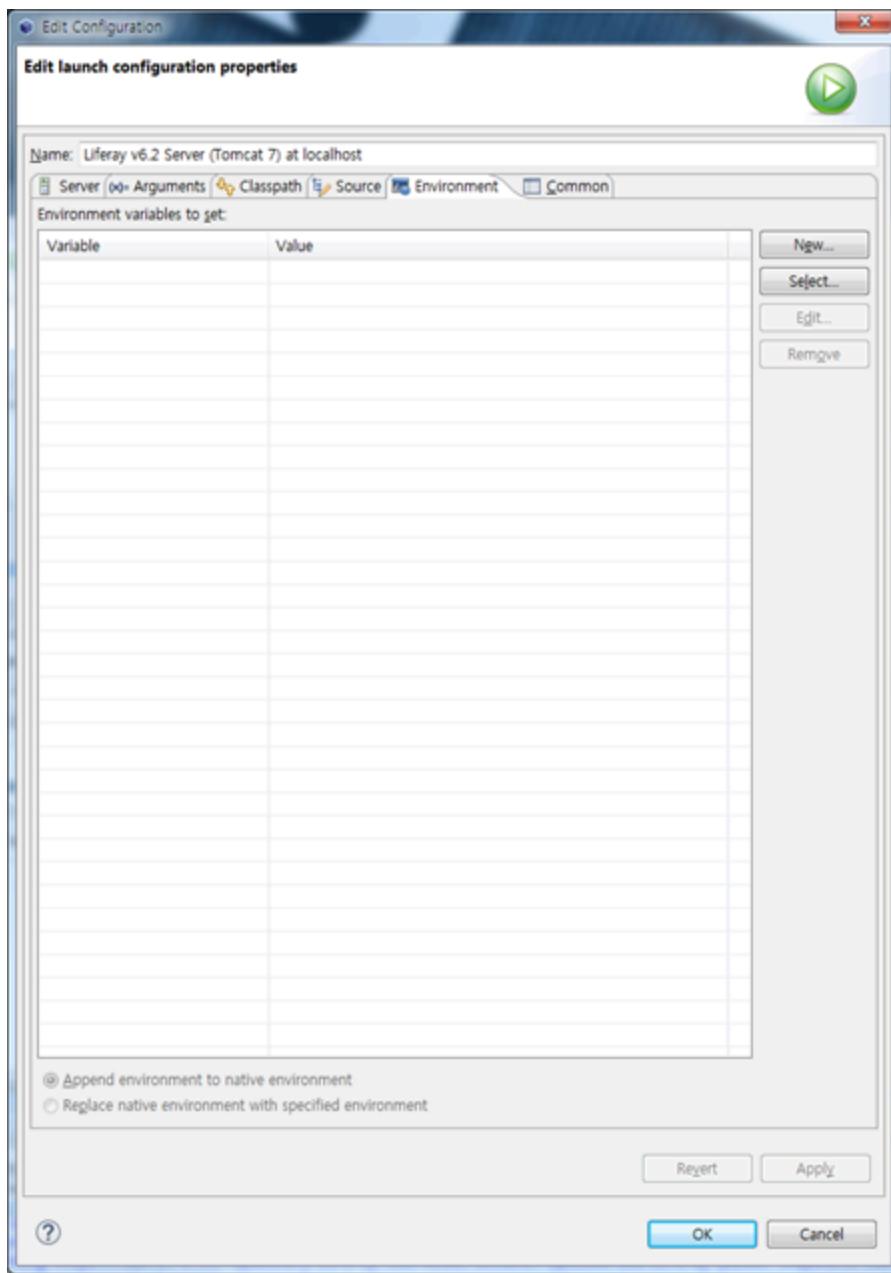


## 서버 실행 설정

서버의 실행 환경 설정을 위해 서버의 Overview 파일에서 **Open Luanch**

**Configuration** 링크를 선택합니다. 해당 링크를 선택하면 아래 그림과 같은 화면이 팝업으로 뜨게 됩니다.



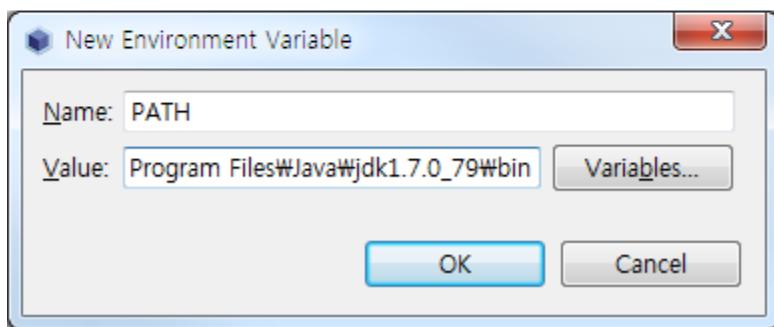
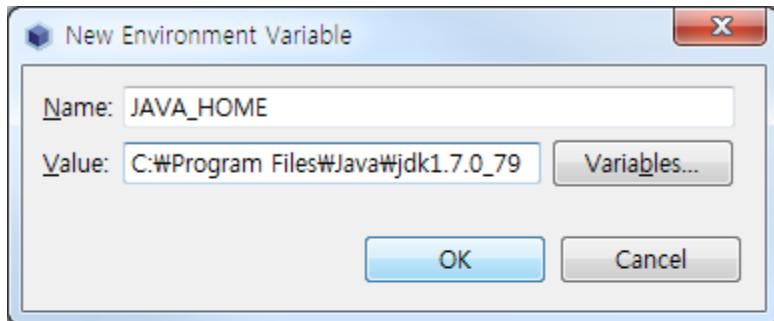


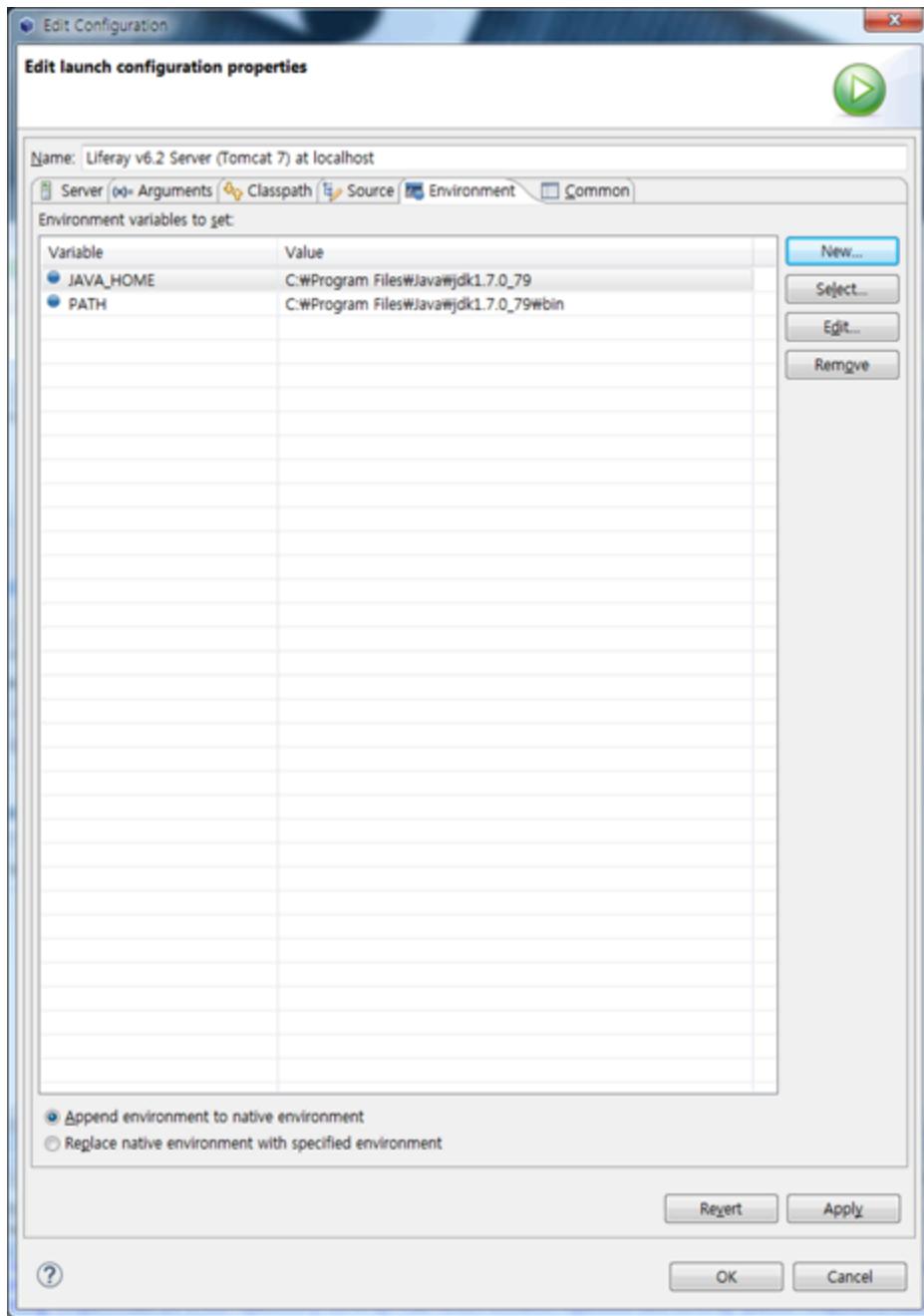
해당 팝업은 실행 환경에서 변수를 지정할 수 있습니다. 해당 변수를 지정하기 위해 다음 그림과 같이 변수를 설정하고 저장합니다.

변수 명	내용	의미
JAVA_HOME	C:\Program Files\Java\jdk1.7.0.79	설치된 JDK 1.7 폴더 경로

변수 명	내용	의미
PATH	C:\Program Files\Java\jdk1.7.0_79\bin	설치된 JDK 1.7 폴더 내 bin 폴더 경로

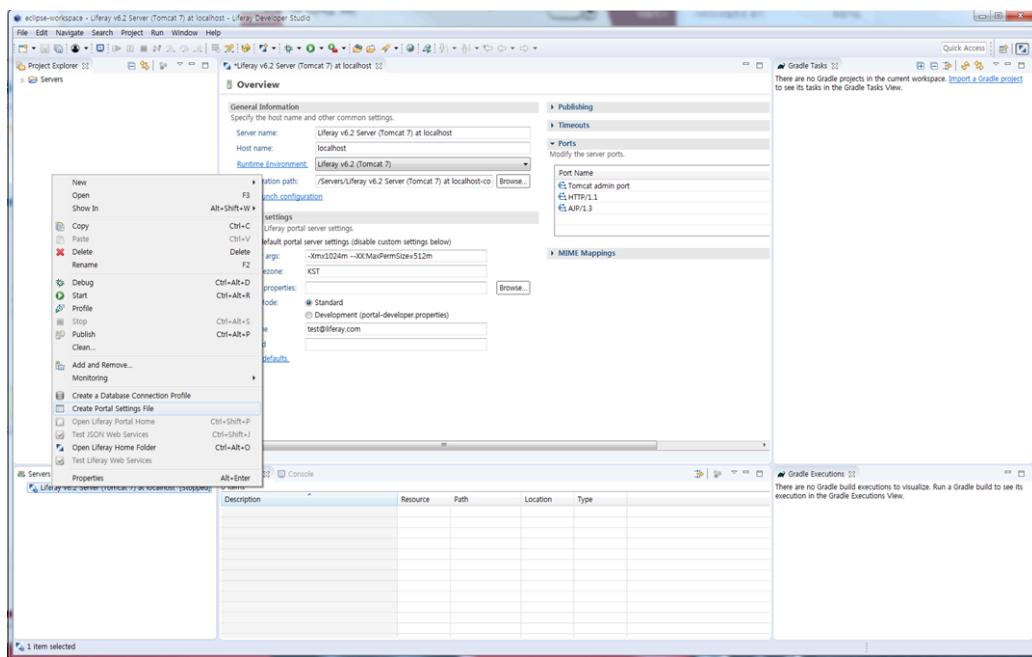
변수를 지정하는 그림은 아래와 같습니다.





## Portal 서버 세부 설정

서버에 추가적으로 설정해야 하는 내용이 있습니다. 따라서 해당 서버의 오른쪽 클릭을 하여 뜨는 메뉴 중에서 **Create Portal Settings File** 을 선택하면 자동으로 설정 파일이 생성되게 됩니다. 오른쪽 클릭을 했을 때 뜨는 화면은 아래 그림과 같습니다.



## 데이터 베이스 연결

서버와 연결해야하는 데이터베이스가 존재한다면, 해당 `portal-ext.properties` 에서 설정이 가능합니다. Mysql 데이터베이스 시스템을 기준으로 연결해야하는 데이터 베이스인 `myDataBase` 의 명으로 존재한다면 다음 예시와 같이 설정 할 수 있습니다.

```
#  
# MySQL  
#  
# Local  
jdbc.default.driverClassName=com.mysql.jdbc.Driver  
jdbc.default.url=jdbc:mysql://localhost/myDataBase?useUnicode=true&characterEncoding=UTF-8&useFastDateParsing=false  
jdbc.default.username=root  
jdbc.default.password=******(해당 패스워드)
```

## 파일 익스플로러 지원을 위한 디렉토리 설정

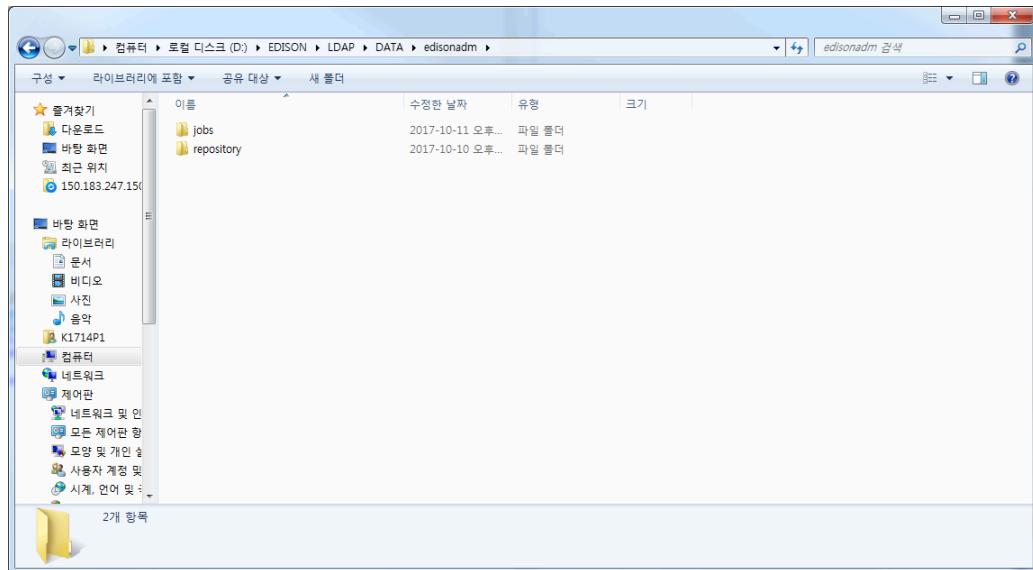
현재 EDISON 포털의 워크벤치를 기반으로 전후처리기를 개발한다면, 해당 전후처리기 개발을 위한 파일 익스플로러의 지원이 필요합니다. 따라서 파일 익스플로러 연결을 위한 몇 가지 설정이 필요합니다. 현재 EDISON 서버 내의 파일 연결을 위한 설정입니다.

```
portlet.add.default.resource.check.enabled=false  
  
osp.root.dir.path=/EDISON/LDAP  
osp.user.root.dir.path=/EDISON/LDAP/DATA
```

위와 같이 해당 내용을 설정한 후, 파일 익스플로러와 연결되어 전후처리기가 동작하는 설정을 하기 위해서는 현재 서버가 위치하는 드라이브 하위에 다음과 같은 폴더 및 디렉토리 구조를 생성해야 합니다. 편집기 분석기에서 서버 파일을 클릭하면 아래의 경로의 파일이 보입니다.

```
/EDISON/LDAP/DATA/[screen name]/repository
```

즉 만약 현재 사용자의 Screen Name 이 **edisonadm** 이고, 톰캣 서버가 존재하는 드라이브가 D 드라이브에 존재한다면 아래 그림과 같은 폴더 구조를 미리 생성해 놓아야 합니다.



# ANT 1.9 설정

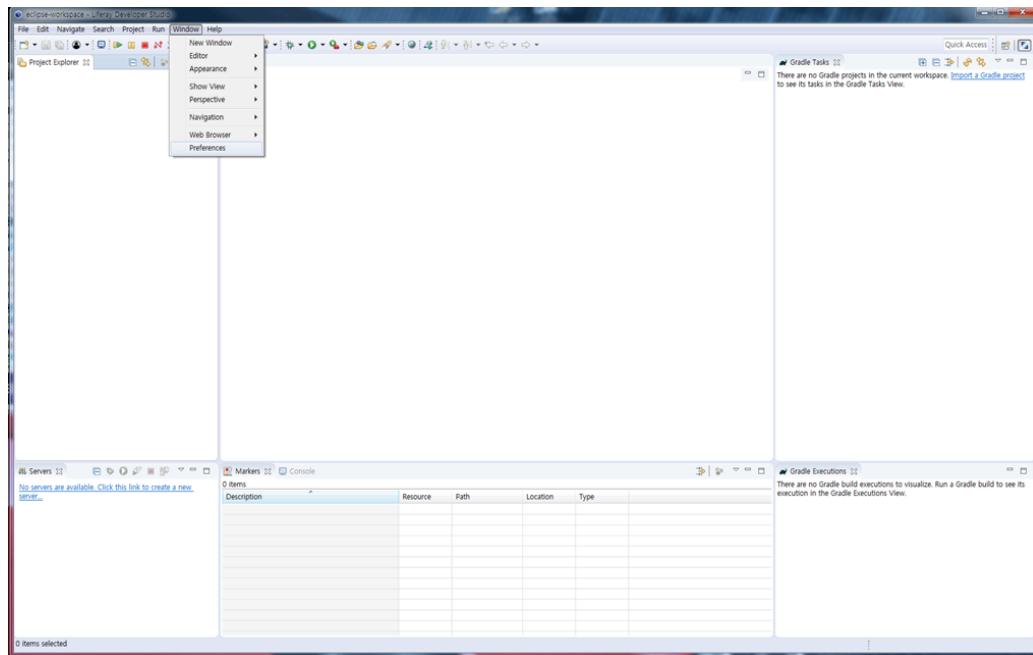
**Summary:** 본 문서는 워크벤치 기반 전후처리기 개발을 위한 매뉴얼입니다.

## ANT 1.9 설정

기본적으로 DeveloperStudio에서 제공되는 Ant는 1.10 버전이며, Liferay 6 버전의 개발을 진행할 수 없습니다. 따라서 기본적인 Ant 설정을 삭제하고 추가적으로 필요한 Ant 1.9 버전을 설치합니다.

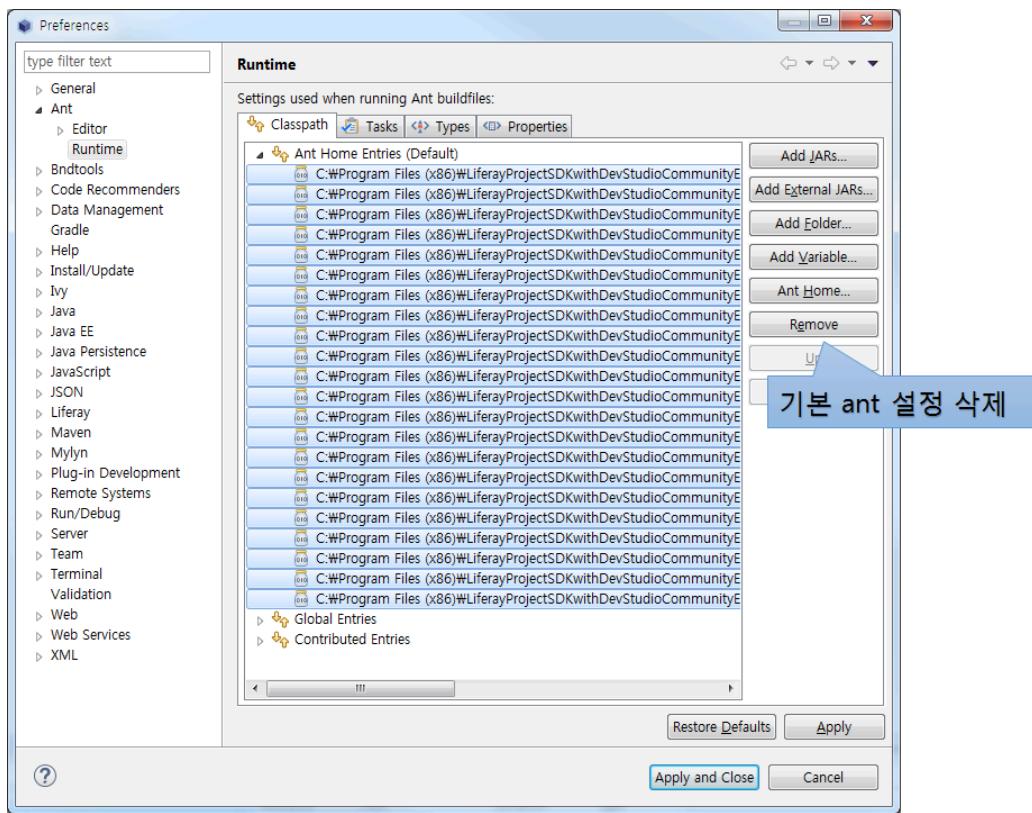
### 1. Preferences 설정

처음으로 DeveloperStudio에서 상위 메뉴에서 Window 하위 메뉴 중 Preferences 를 선택합니다.



### 2. 기본 Ant 설정 삭제

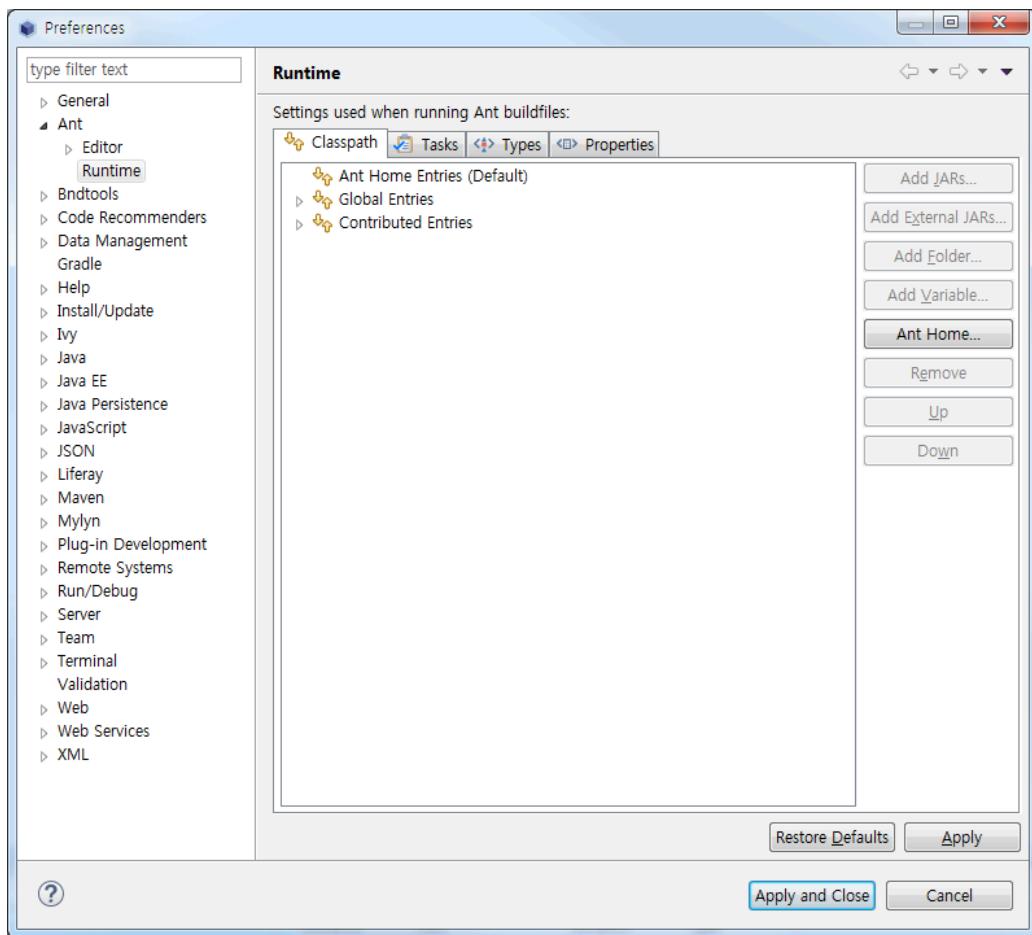
Preferences 창에서 Ant 하위 메뉴 중 Runtime 메뉴를 선택하면 다음 그림과 같은 화면이 나오게 됩니다.



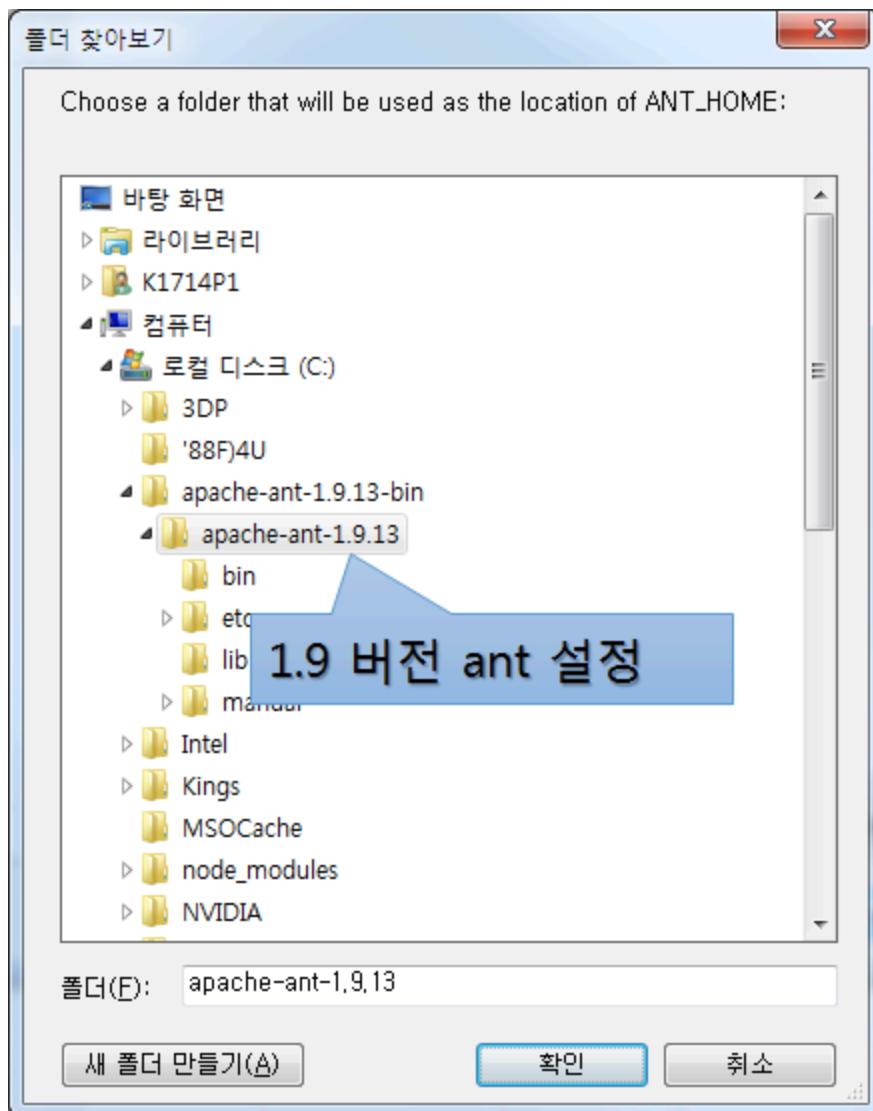
위의 표시된 내용과 같이 Ant의 Home Entries의 내용을 전체 선택 한 후, 기본 Ant 설정을 삭제합니다.

### 3. Ant Home 설정

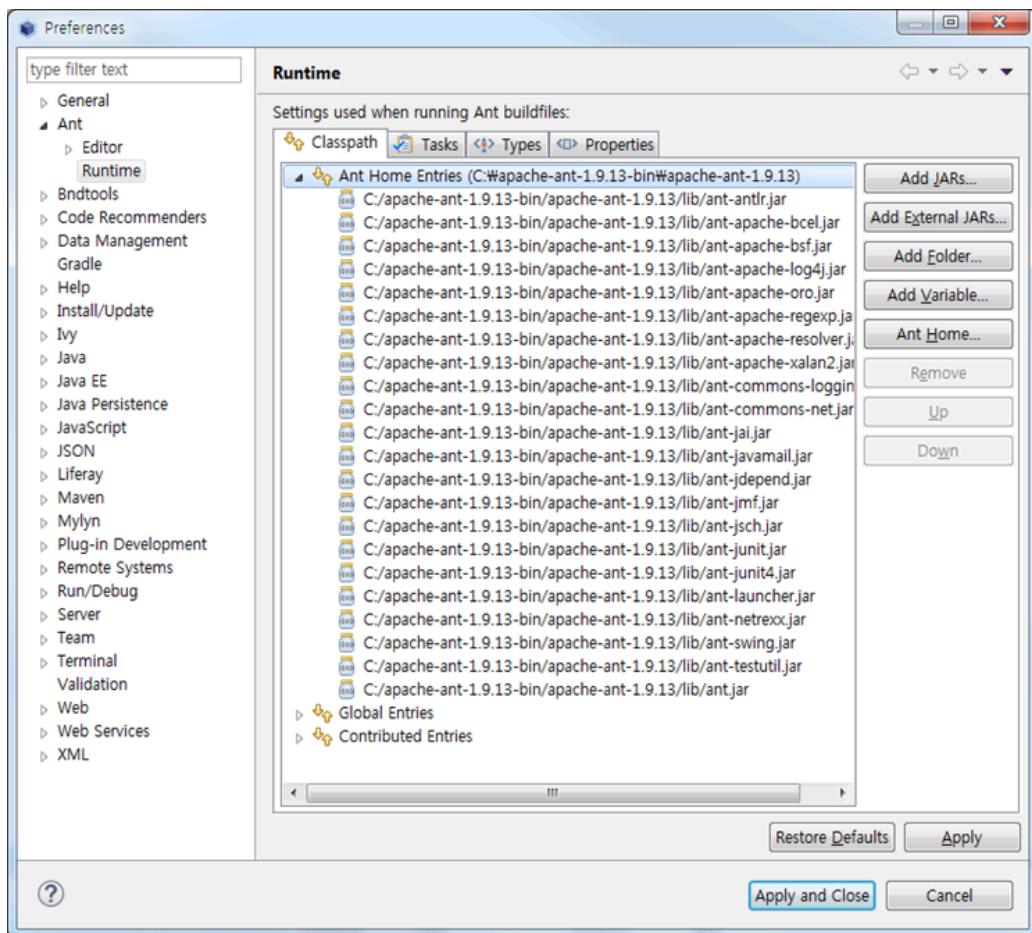
모든 Default로 설정된 Ant 설정이 사라지면 아래와 같은 화면이 됩니다. 그림에서와 같이 이 **Ant Home...** 버튼을 선택합니다.



Ant Home 버튼을 선택하면 Ant가 설치된 디렉토리를 설정할 수 있습니다. Ant 1.9가 설치된 디렉토리를 아래와 같이 선택합니다.



디렉토리를 선택하고 확인 버튼을 누르면 자동으로 Ant Home Entries에 적용된 화면을 아래 그림과 같이 볼 수 있습니다.



# SDK 설정

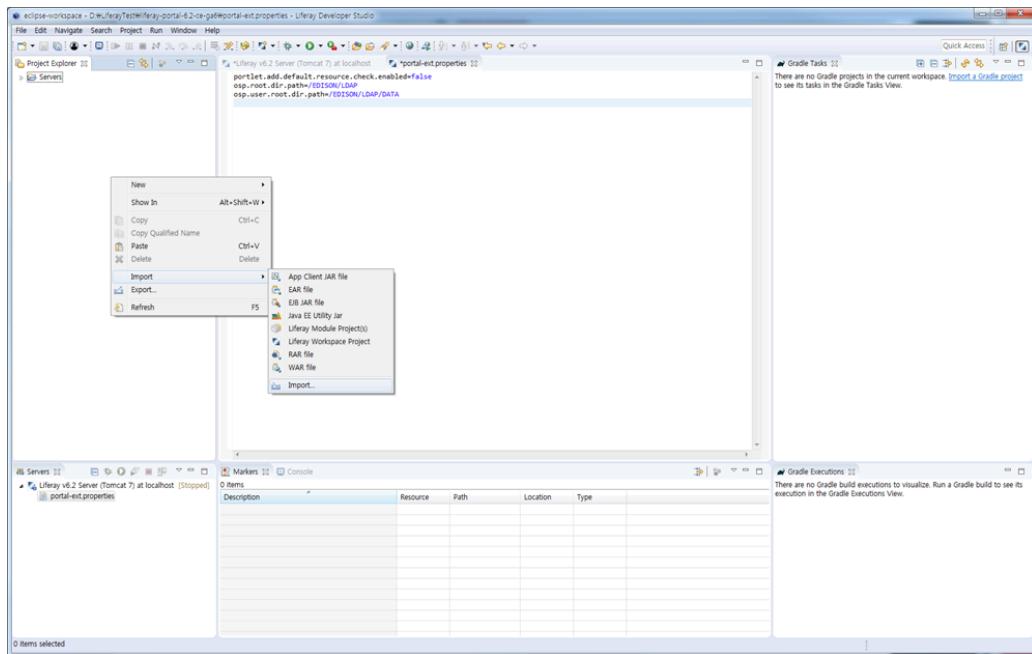
**Summary:** 본 문서는 워크벤치 기반 전후처리기 개발을 위한 매뉴얼입니다.

## SDK 설정

Liferay 6.2에서 개발되어 추가되는 개발 모듈은 포틀릿(portlet) 단위로 구성되어 플러그인처럼 추가 될 수 있습니다. 따라서 개발 모듈에 대한 전체 디렉토리인 Liferay Plugins SDK Directory를 설정할 수 있습니다.

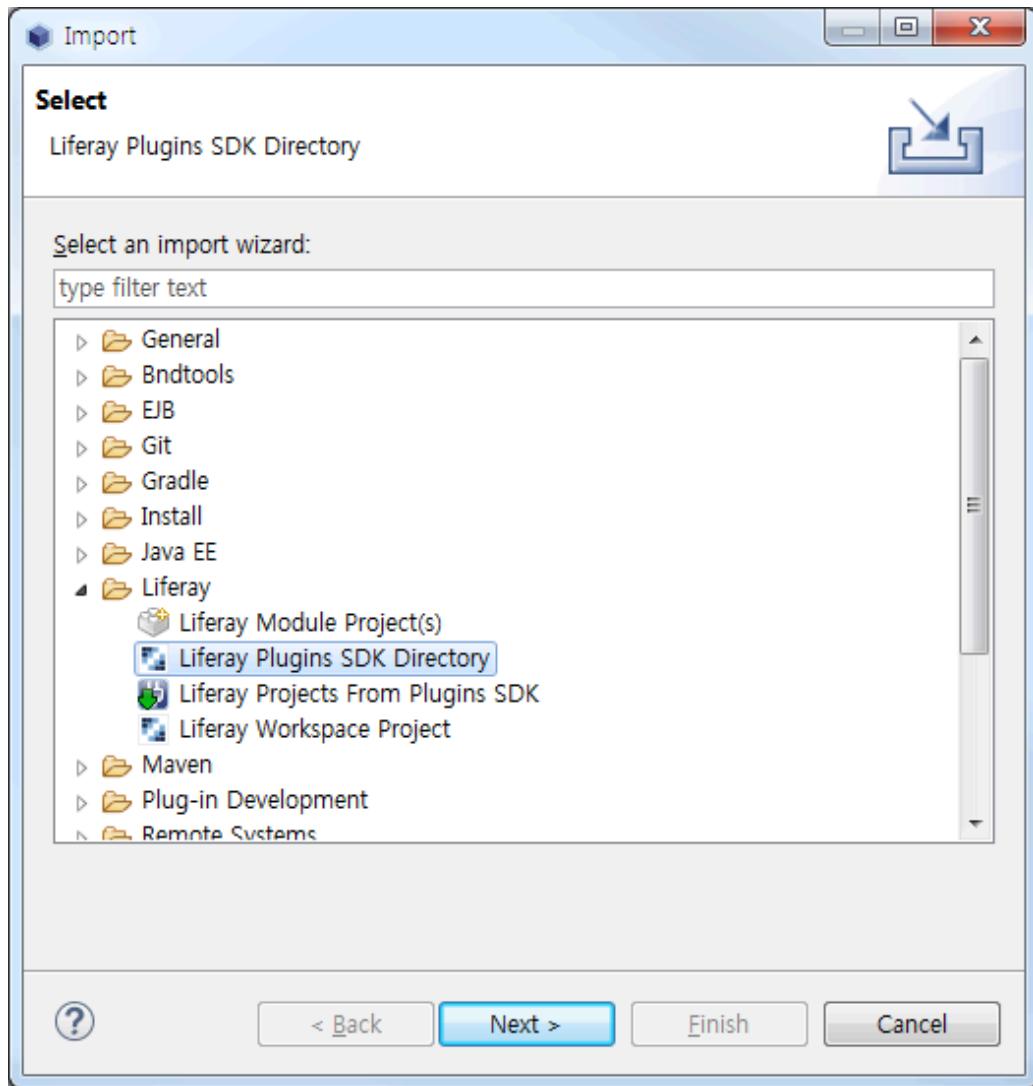
### 1. Import SDK 풀더

처음으로 DeveloperStudio에서 Project Explorer 부분에서 사용자 마우스 우클릭을 합니다. 그 후 Import 메뉴의 하위 메뉴에서 Import를 선택합니다. 아래 그림과 같이 선택합니다.



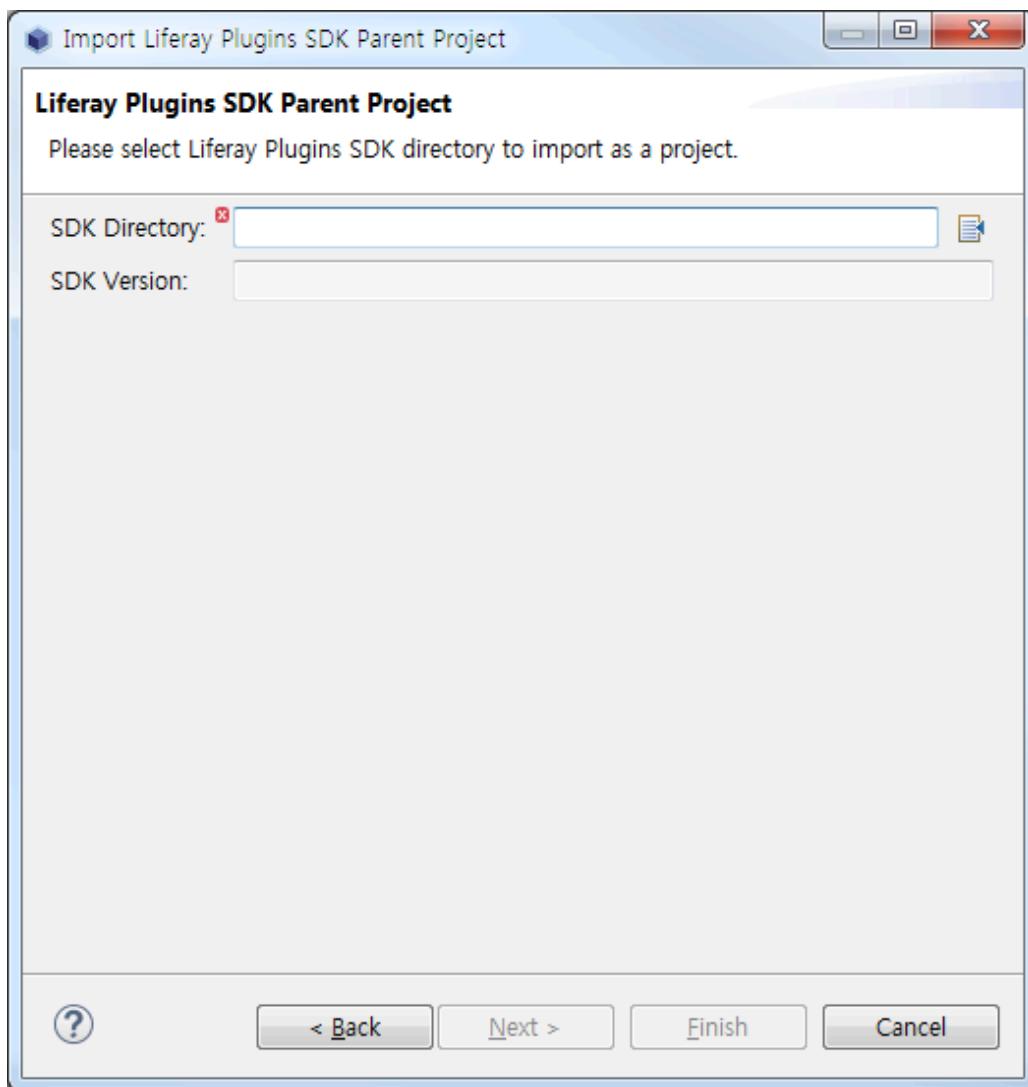
## 2. Liferay Plugins SDK Directory 선택

Import를 선택하면 아래와 같은 팝업창이 나타나게 됩니다. 이 중에서 Liferay 하위 메뉴에서 **Liferay Plugins SDK Directory**를 선택합니다. 그리고 난 후 Next 버튼을 선택합니다.

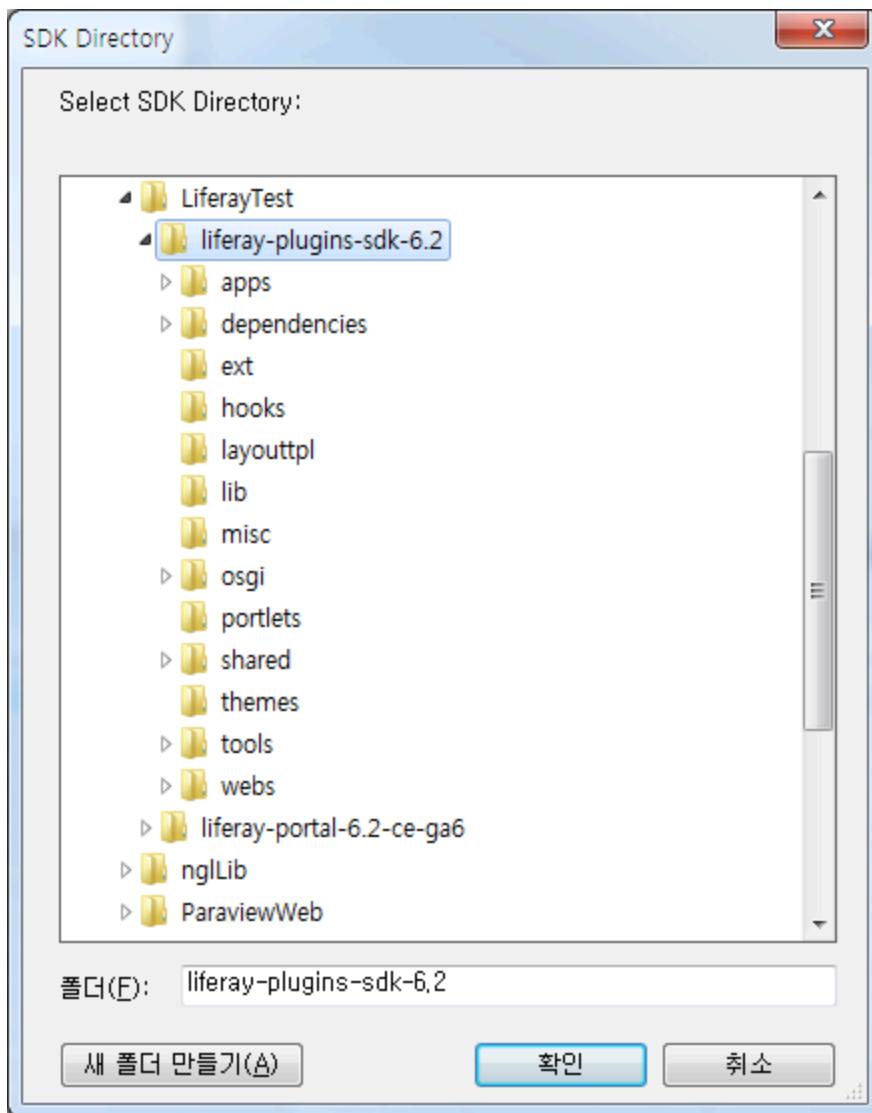


## 3. SDK 디렉토리 설정

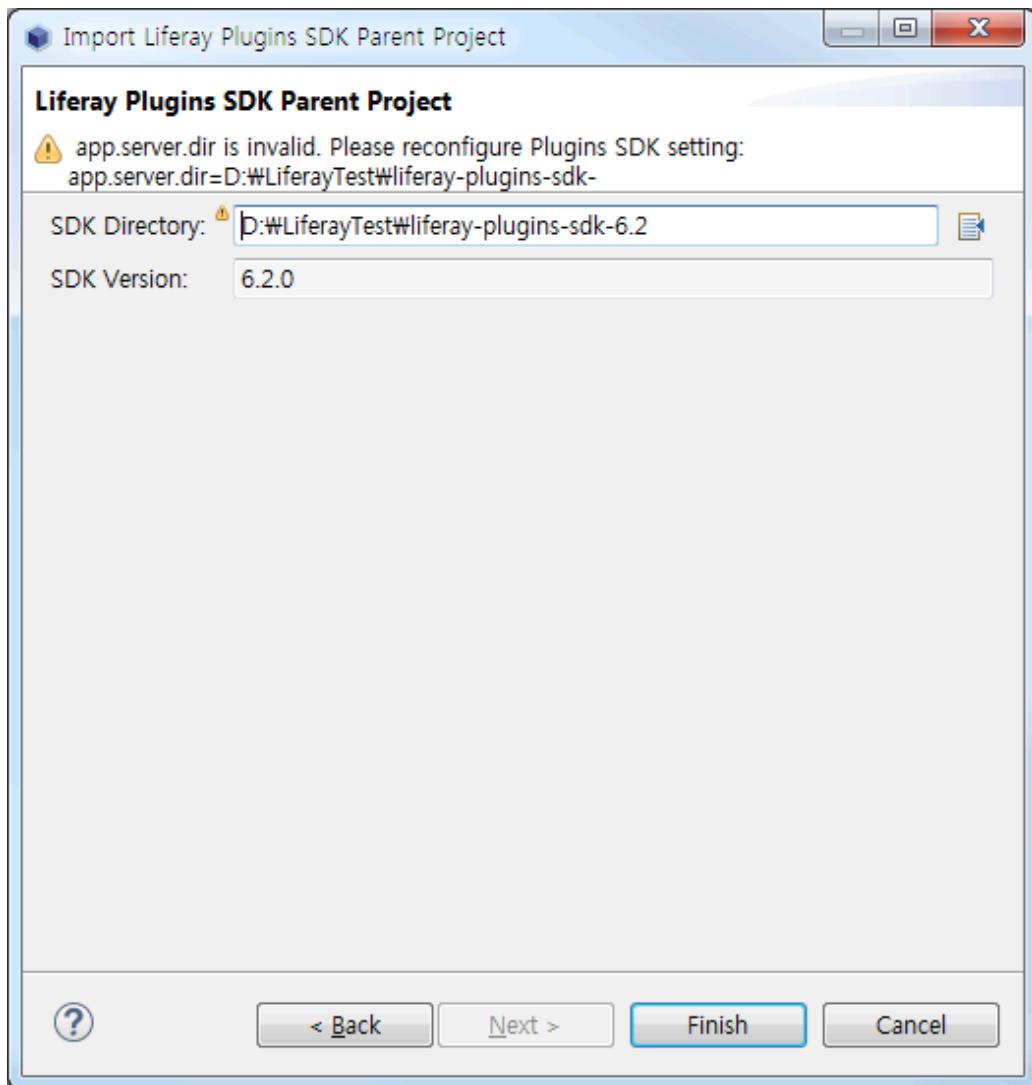
SDK 디렉토리를 자동으로 DeveloperStudio에서는 인식이 가능합니다. 아래 그림과 같이 아무런 SDK 디렉토리가 설정되어 있지 않으면, 다음 단계로 넘어갈 수가 없습니다.



SDK Directory는 초반에 다운받은 Liferay SDK 6.2의 압축을 푼 폴더가 됩니다. 아래 그림과 같이 `liferay-plugins-sdk-6.2`를 선택합니다.

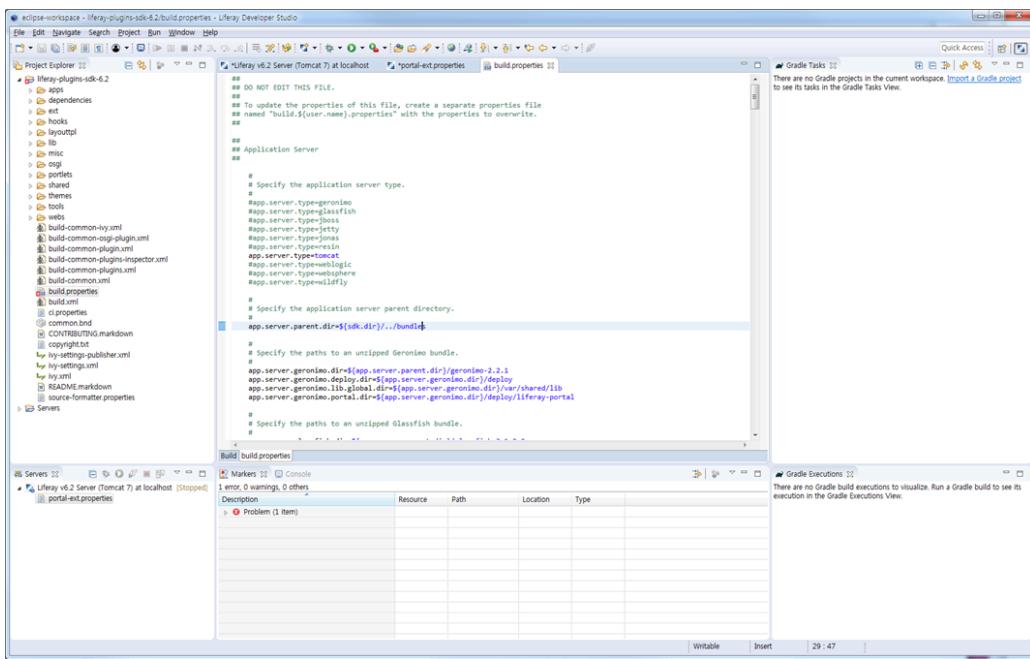


적절한 폴더가 선택되면 아래 그림과 같이 처음과는 다르게 SDK 설정을 마칠 수 있는 **Finish** 버튼이 활성화 됩니다. Finish버튼을 선택하고 설정을 종료합니다.

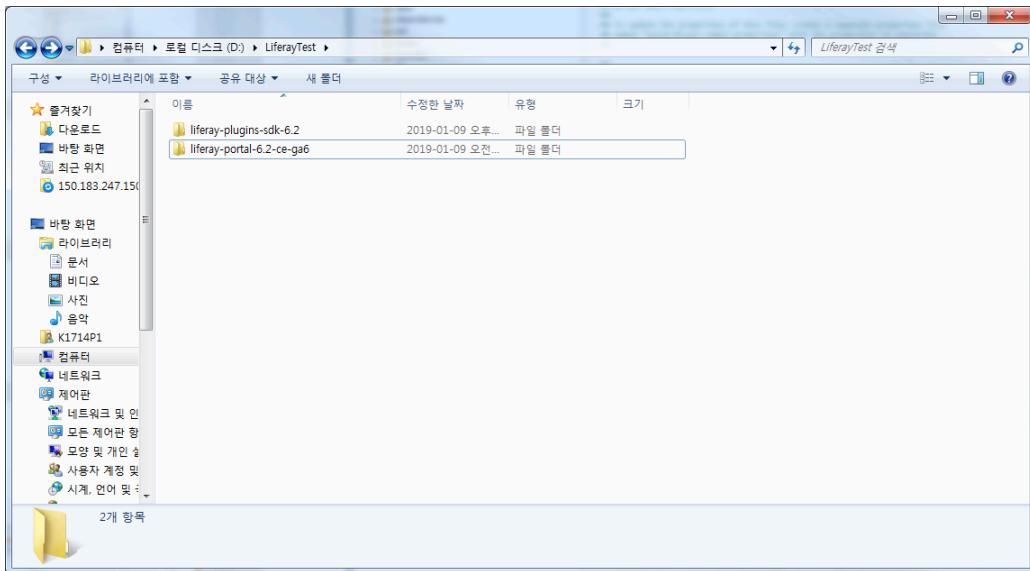


#### 4. 서버 연결

SDK의 import 과정이 모두 끝나게 되면, 서버와 연결을 설정해야 합니다. SDK 폴더 내 `build.properties` 파일 내 tomcat 서버의 연결을 설정하는 부분이 존재하는데 해당 폴더 위치를 다운 받은 Liferay tomcat 서버와 연결시켜야 합니다. 기본으로 설정되어 있는 서버의 위치는 `app.server.parent.dir=${sdk.dir}/../bundles`로 설정되어 있습니다. 해당 내용은 아래 그림에서 확인할 수 있습니다.

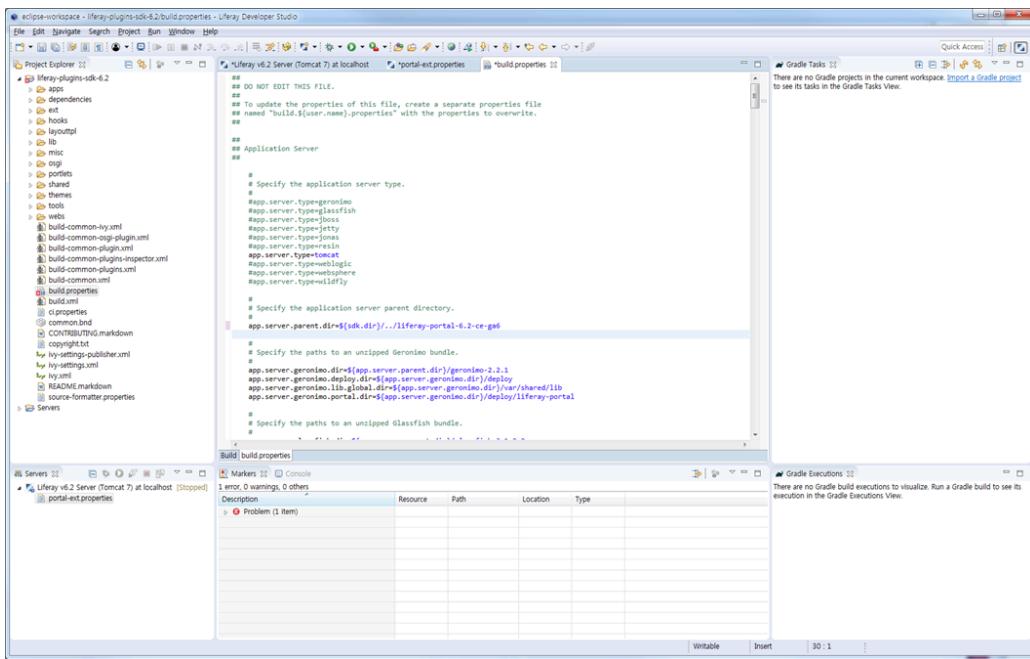


현재 서버와 SDK가 설치된 위치의 관계는 다음 그림과 같습니다.



해당 내용을 바탕으로 서버의 위치를 설정하면

`app.server.parent.dir=${sdk.dir}/../liferay-portal-6.2-ce-ga6` 값으로  
변경해야 합니다. 변경된 내용을 저장하면 다음 그림과 같은 화면을 확인할 수 있습니다.



## 5. ivy url 설정

SDK 폴더 내 ivy 에러 해결을 위해 `plugin-sdk/ivy-settings.xml` 파일 내 ivy url 을 수정해 줘야 합니다. 해당 내용은 아래와 같습니다.

```
<ivysettings>
    <settings defaultResolver="default" />

    <resolvers>
        <!-- remove -->
        <!-- <ibiblio m2compatible="true" name="liferay-public" root="http://cdn.repository.liferay.com/nexus/content/groups/public" /> -->
        <!-- add -->
        <ibiblio m2compatible="true" name="liferay-public" root="https://repository.liferay.com/nexus/content/repositories/public" />
        <ibiblio m2compatible="true" name="local-m2" root="file://${user.home}/.m2/repository" />

        <chain dual="true" name="default">
            <resolver ref="local-m2" />

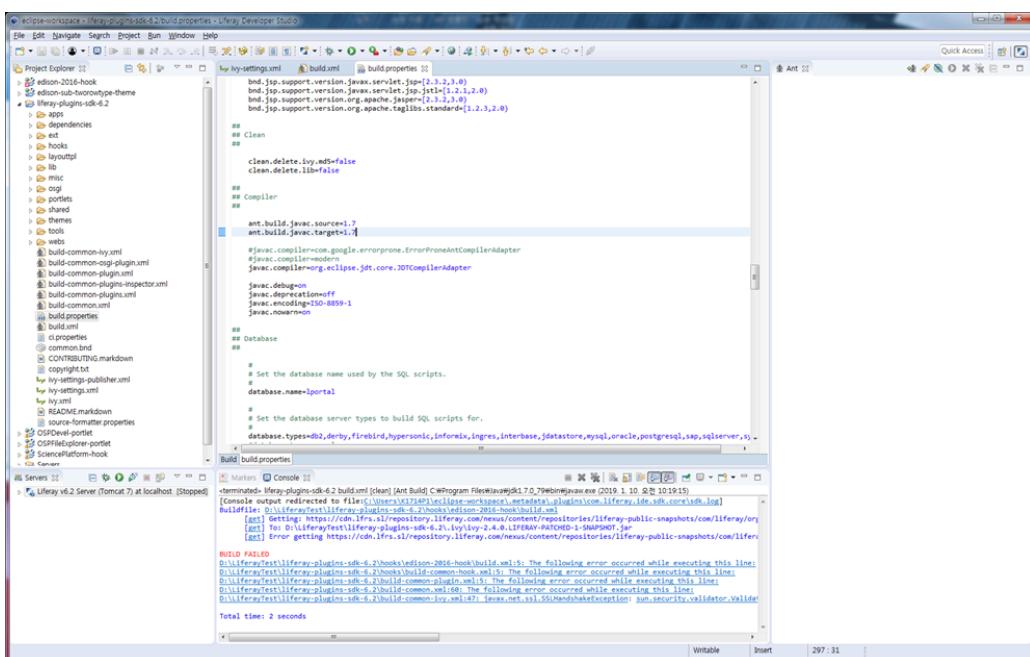
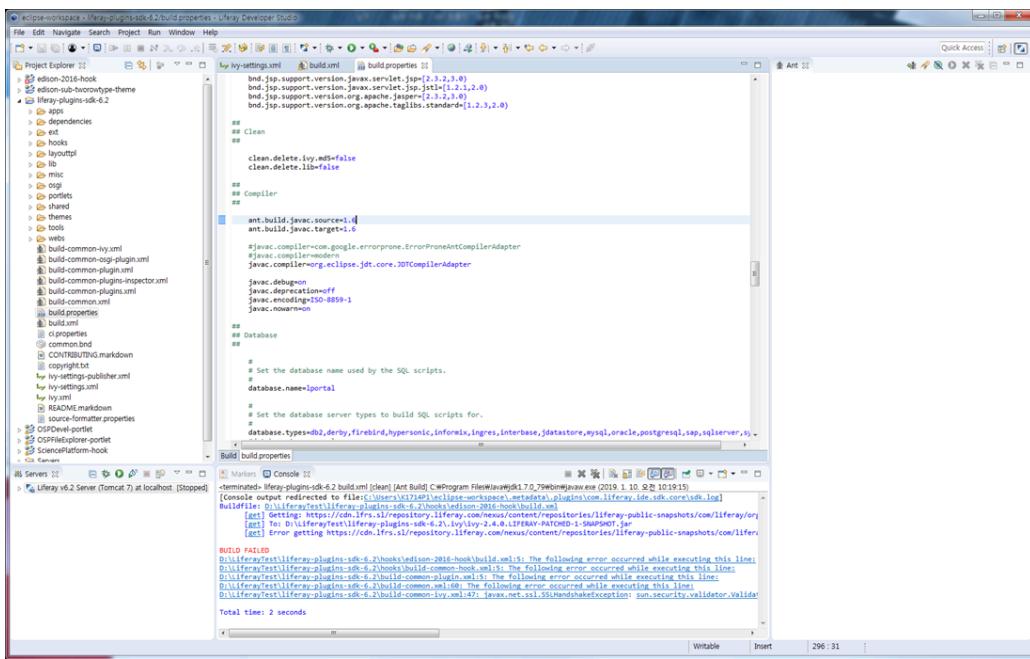
            <resolver ref="liferay-public" />
        </chain>
    </resolvers>
</ivysettings>
```

최근 URL 인식이 안되는 경우가 있습니다. 해당하는 경우 다음 링크에서 파일을 다운받아 압축을 풀어서 SDK 폴더 내에 `.ivy`에 덮어씌워 실행을 하면 에러를 처리할 수 있습니다.

- [.ivy \(page 0\)](#) [.ivy \(page 0\)](#)

## 6. 그 외 설정

SDK 폴더 내 `build.properties` 내부에 compiler와 관련된 내용을 수정해야 합니다. 해당 내용은 아래 그림과 같습니다.



수정해야 하는 내용은 다음 표와 같습니다.

수정 전	수정 후
ant.build.javac.source=1.6	ant.build.javac.source=1.6

수정 전	수정 후
ant.build.javac.target=1.6	ant.build.javac.target=1.7

## 프로젝트 생성

**Summary:** 에디슨 워크벤치 기반 후처리기(Post-Processor)를 개발하기 위한 프로젝트 생성 및 기본 설정에 대한 매뉴얼

[EDISON](#) 워크벤치 기반 분석기(Post-Processor) 개발 매뉴얼입니다.

Liferay 6.2.5 포틀릿 기반으로 개발 하는 방법에 대해 순차적으로 설명 되어 있습니다.

### Liferay 6.2 기반 프로젝트 구성

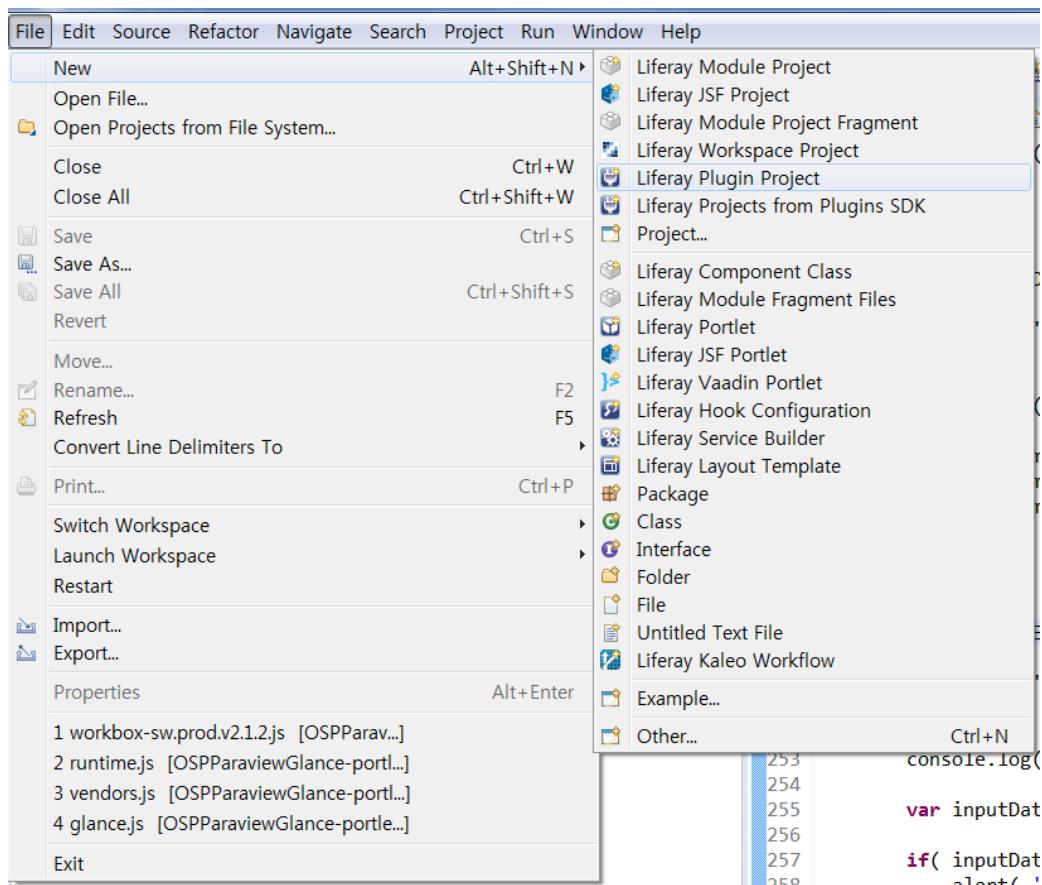
개발 환경은 아래와 같습니다.

- `이클립스 Neon 3 Release (4.6.3)`
- `liferay-portal-6.2-ce-ga6`
- `liferay-plugins-sdk-6.2`

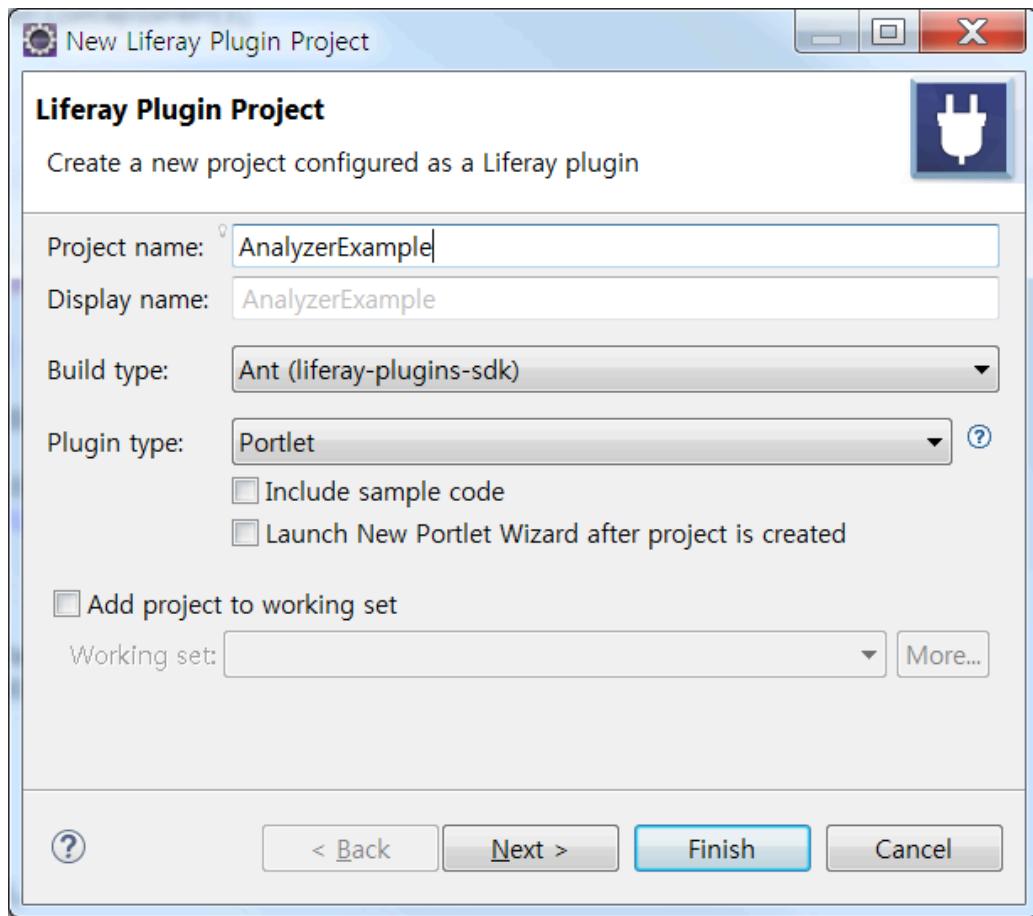
### 1. Liferay plugin Project 생성

이클립스 환경에서 `Liferay Plugins Projects` 를 생성합니다.

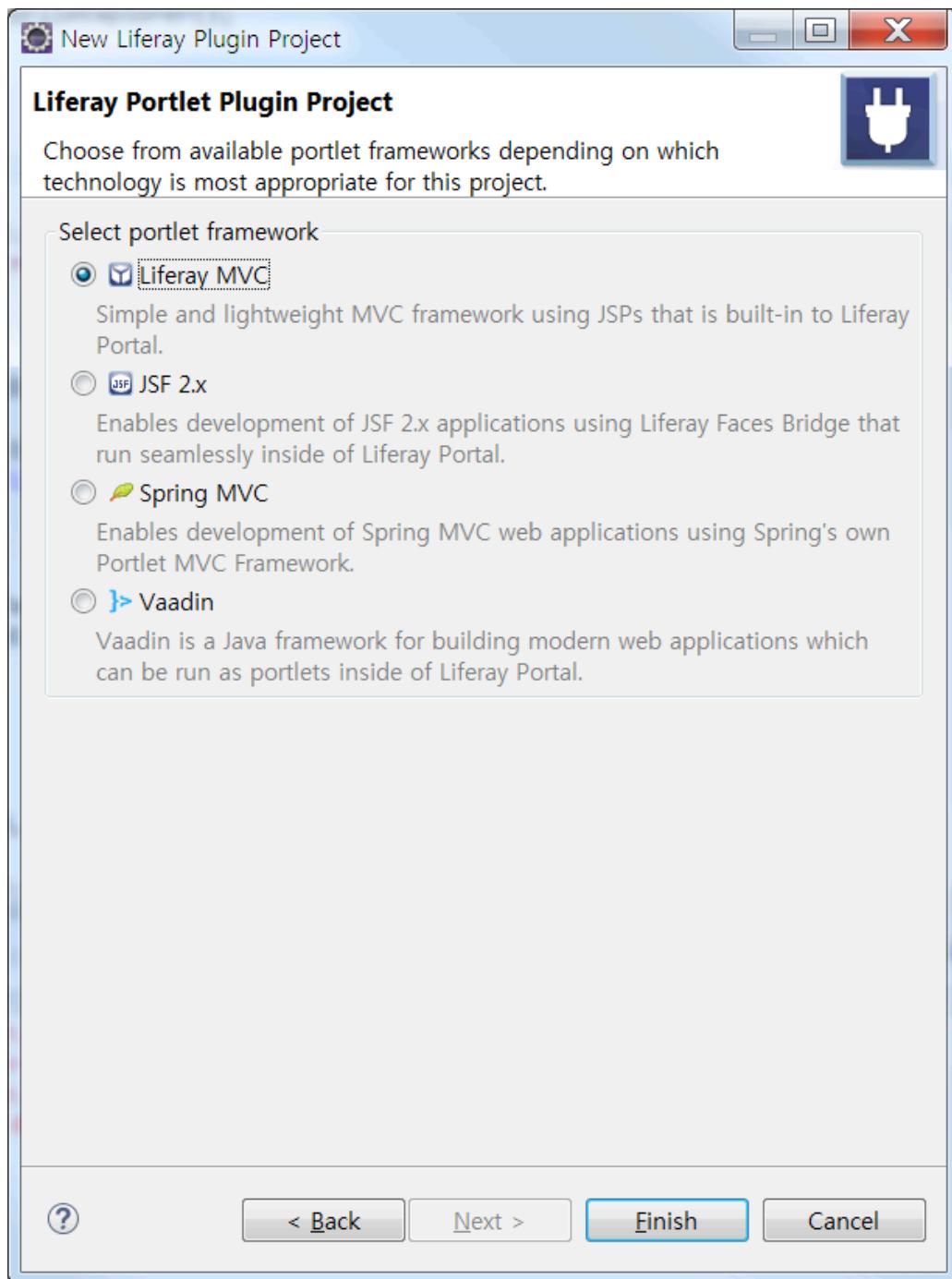
이클립스 환경에서 New -> Liferay Plugin Project를 선택합니다.



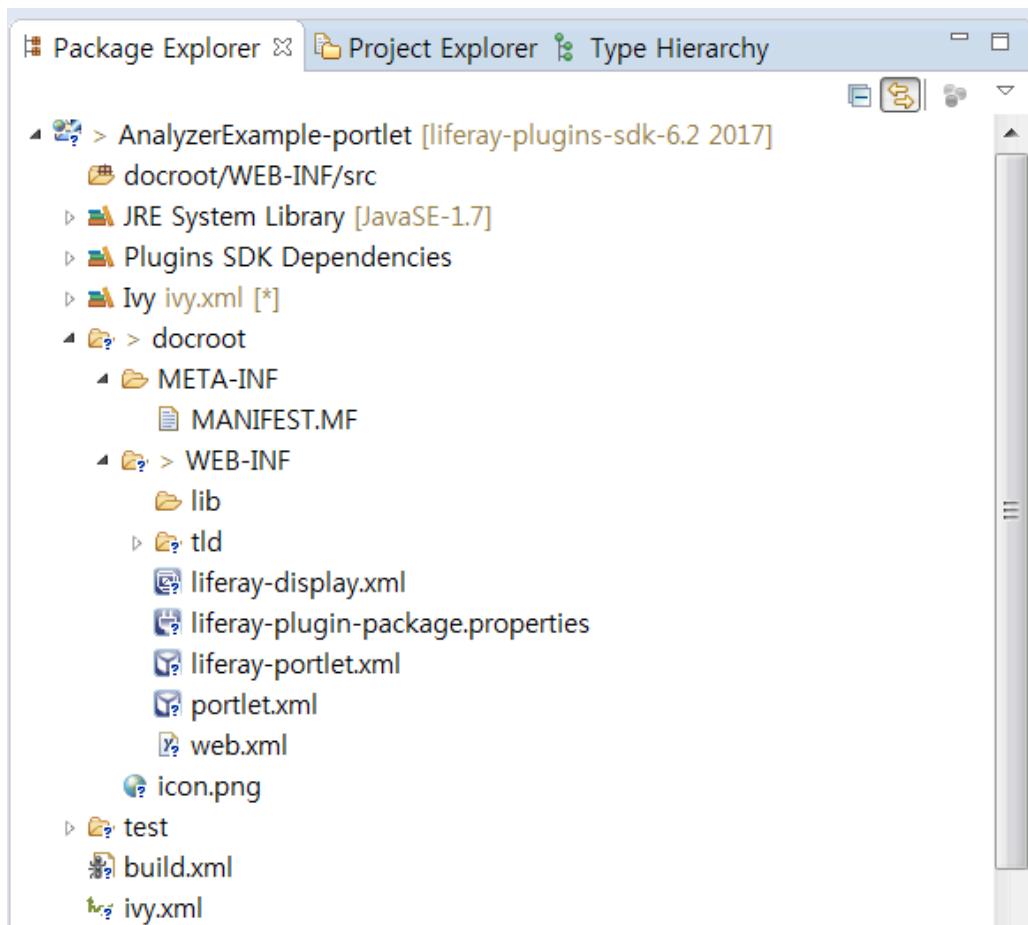
프로젝트 이름을 작성하고 include sample code 체크를 해지합니다.



프로젝트 이름을 작성하고 Liferay MVC를 선택하여 최종적으로 프로젝트 생성을 마무리 합니다.



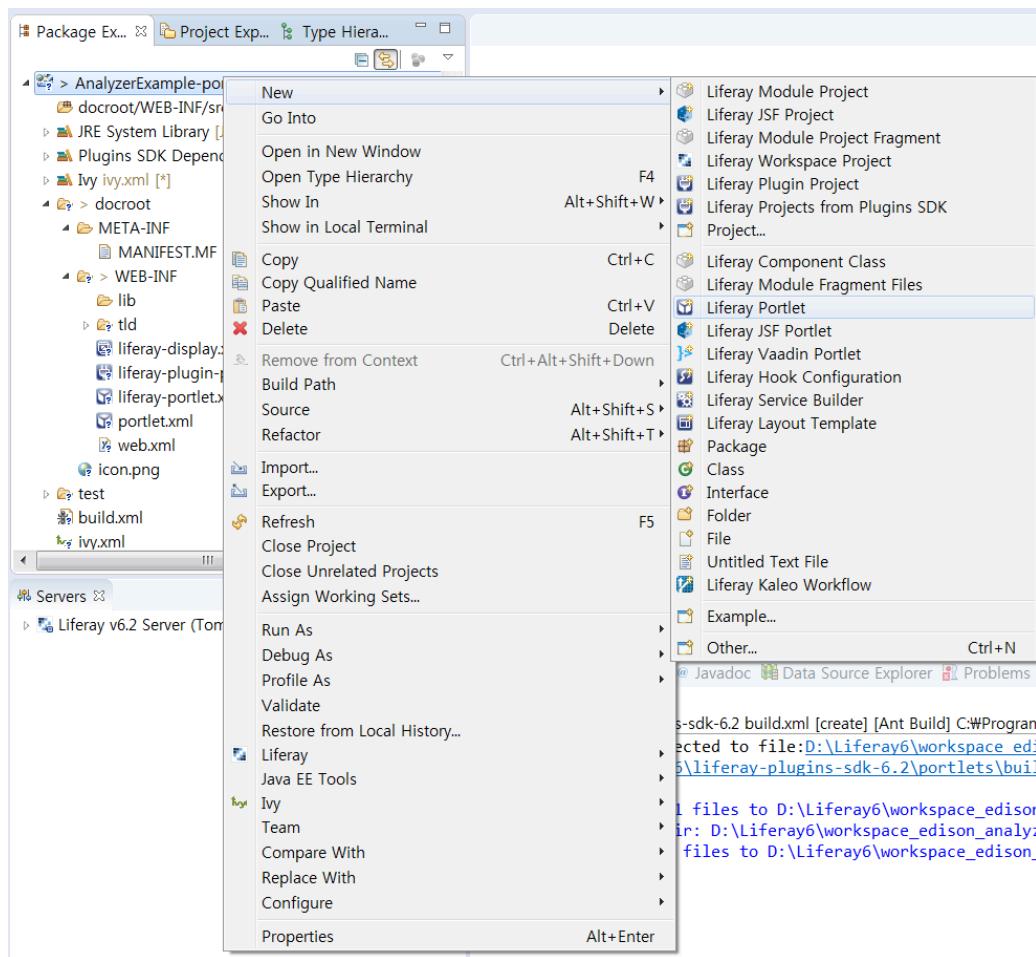
생성이 완료된 플러그인 프로젝트의 구성은 다음과 같습니다.



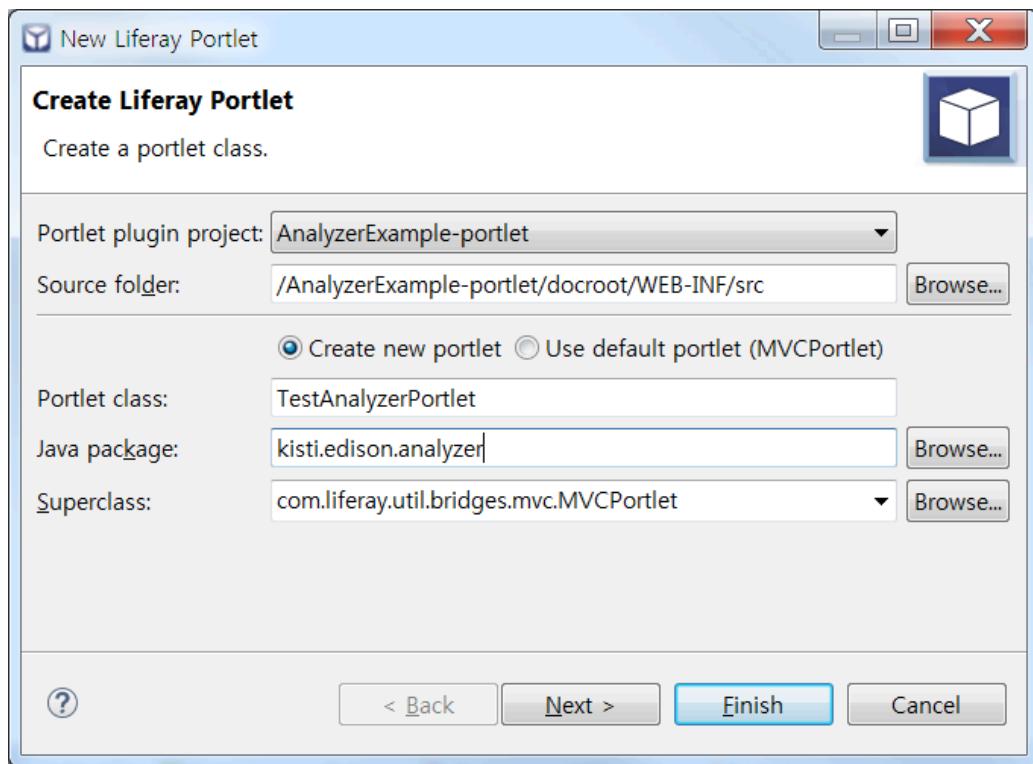
## 2. EDISON 워크벤치 연동을 위한 Analyzer 포틀릿 생성

EDISON 플랫폼의 워크벤치 연동을 위한 포틀릿을 생성합니다.

생성된 Liferay Plugins Projects 내에 Liferay Portlet을 생성합니다.

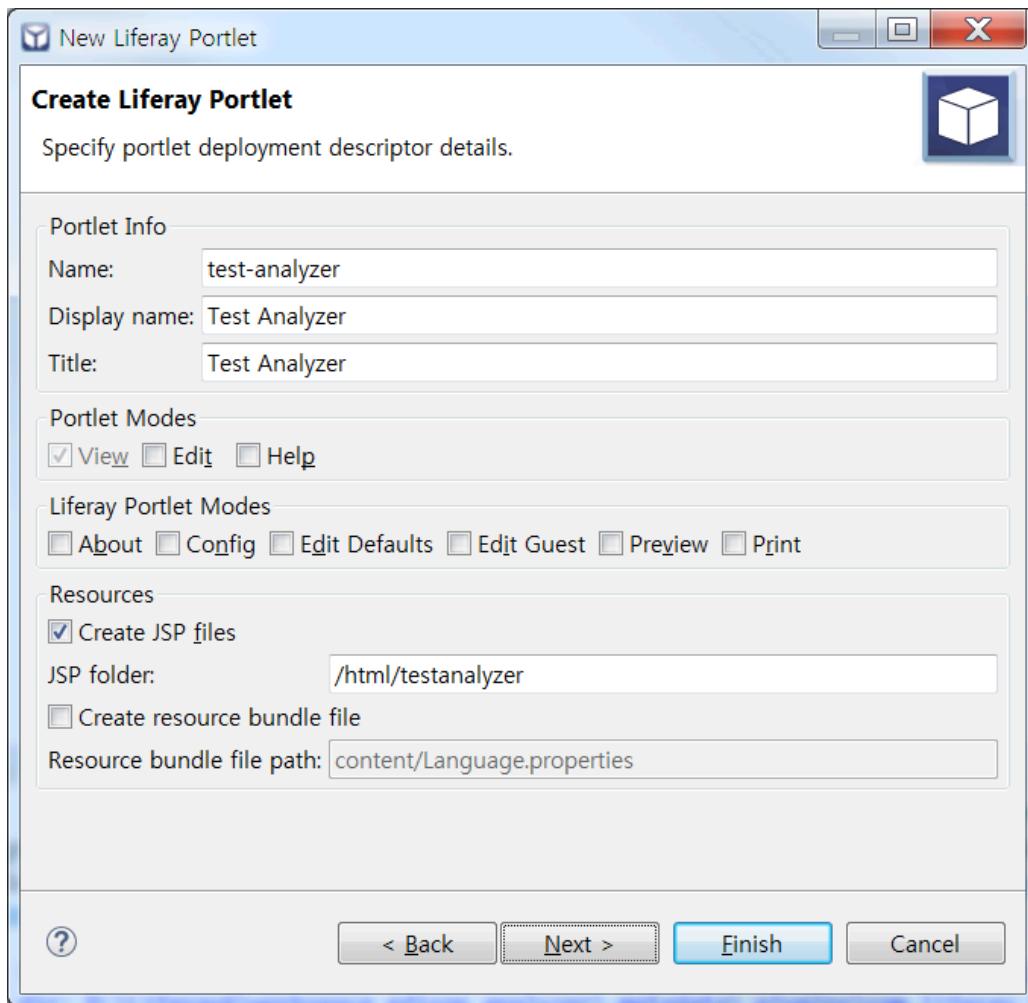


포틀릿 클래스명, 패키지명은 자유롭게 원하는 형태로 생성합니다.



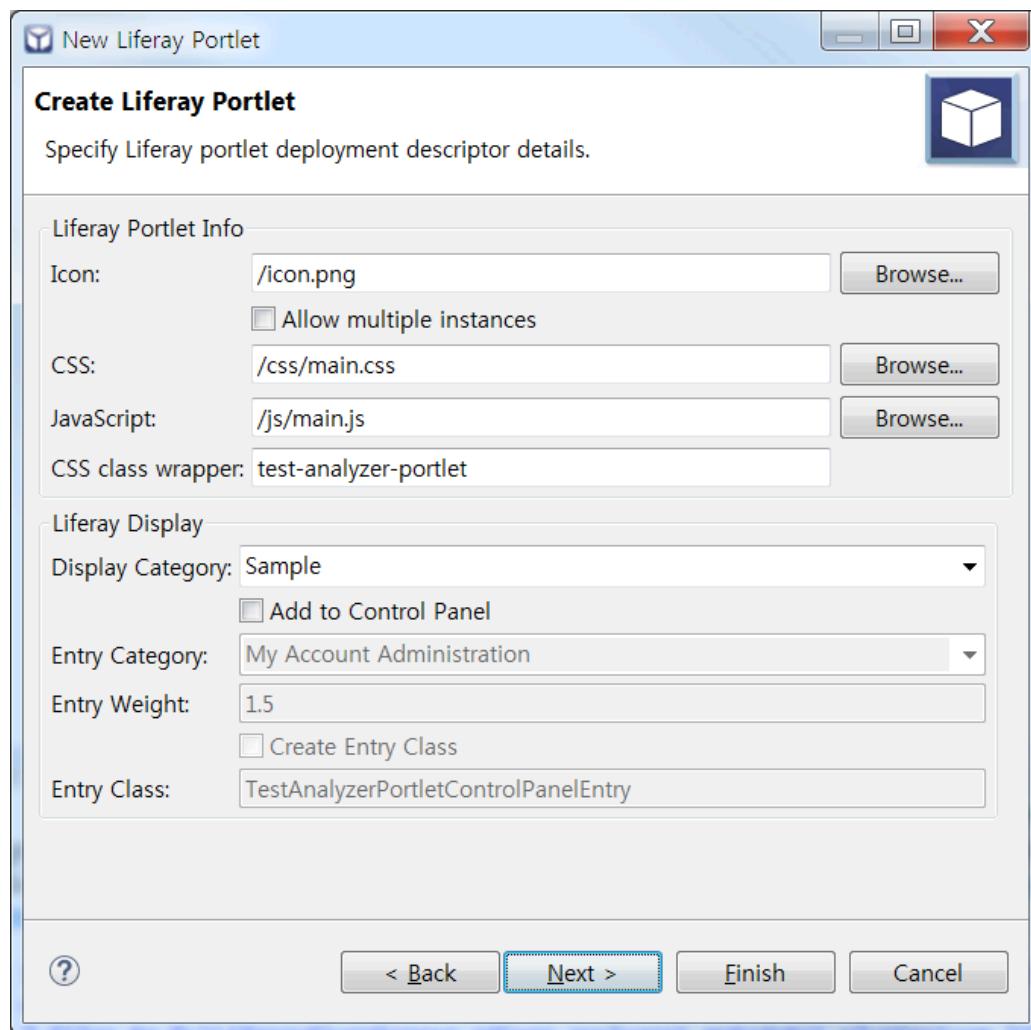
포틀릿 클래스 명에 따라 자동으로 설정되는 값입니다.

나중에 수정할 수 있으니 현재는 Next를 누르고 넘어갑니다.

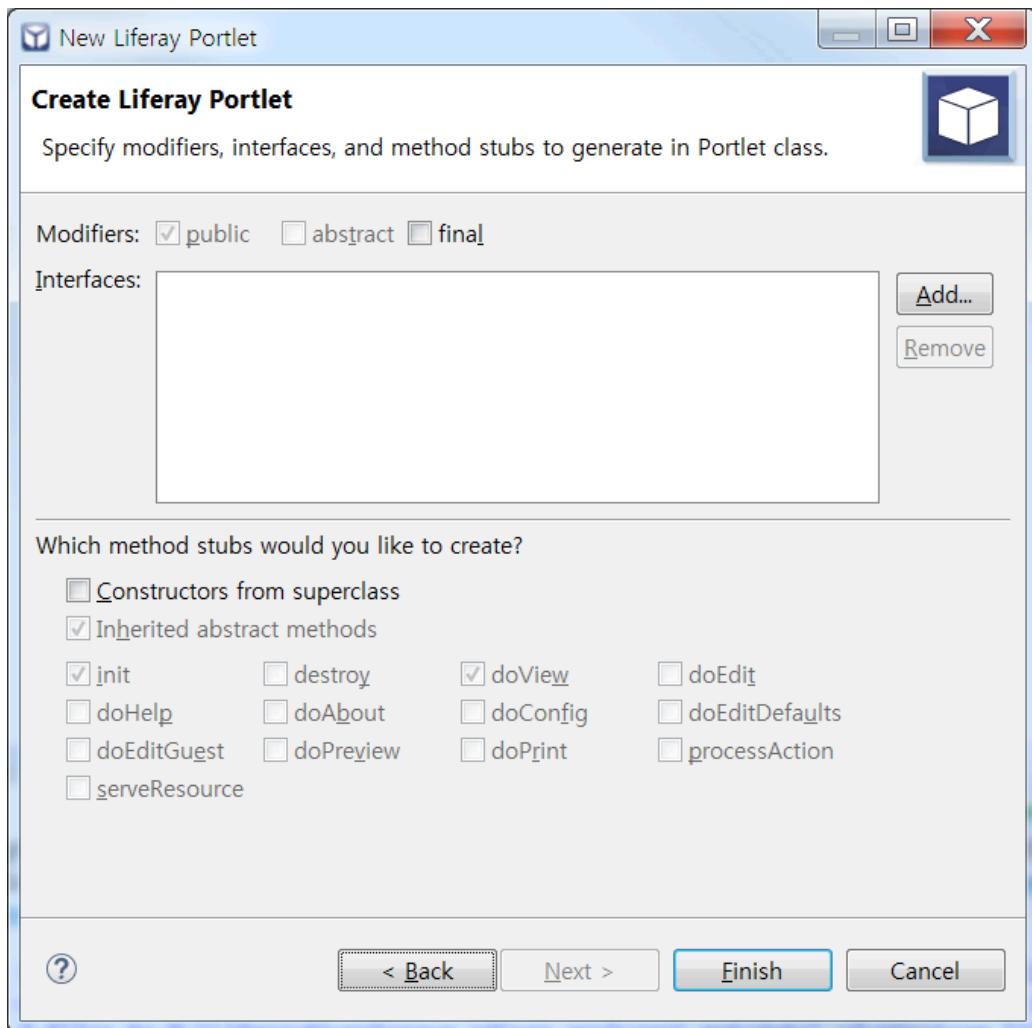


마찬가지로 자동으로 설정되는 값입니다.

나중에 수정할 수 있으니 현재는 Next를 누르고 넘어갑니다.



특별히 추가할 인터페이스가 없다면 Finish를 선택하고 포틀릿 생성을 완료합니다.



### 3. 포틀릿 리소스 접근 설정

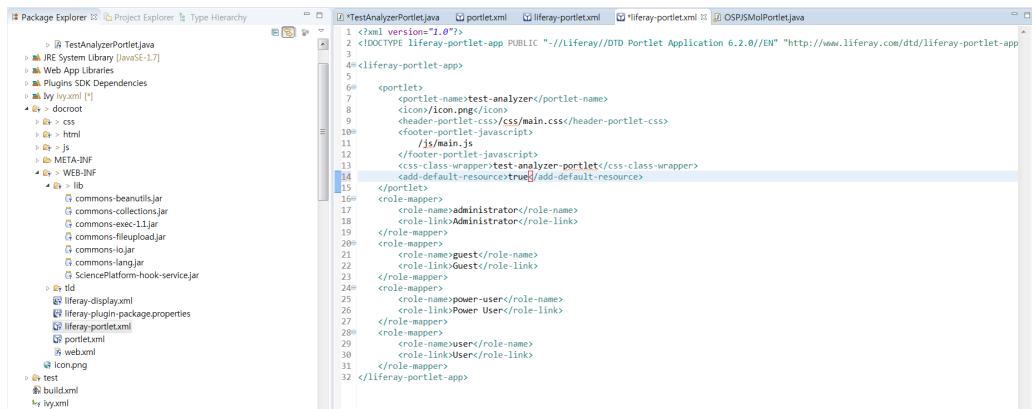
EDISON 플랫폼에서 워크벤치와 연동을 위해서는 각 포틀릿간의 외부 접근이 가능하도록 권한 설정을 해 주어야 합니다.

따라서 외부(다른 포틀릿)에서 개발하려는 포틀릿으로 접근하기 위해서는 포틀릿의 외부 권한을 설정하여야 합니다. 설정하는 방법은 아래와 같습니다.

`liferay-portlet.xml` 파일 내부에 아래 코드를 추가합니다.

```
<add-default-resource>true</add-default-resource>
```

아래의 그림과 같이 xml 파일 내부에 생성한 포틀릿 내 해당 코드를 추가합니다.

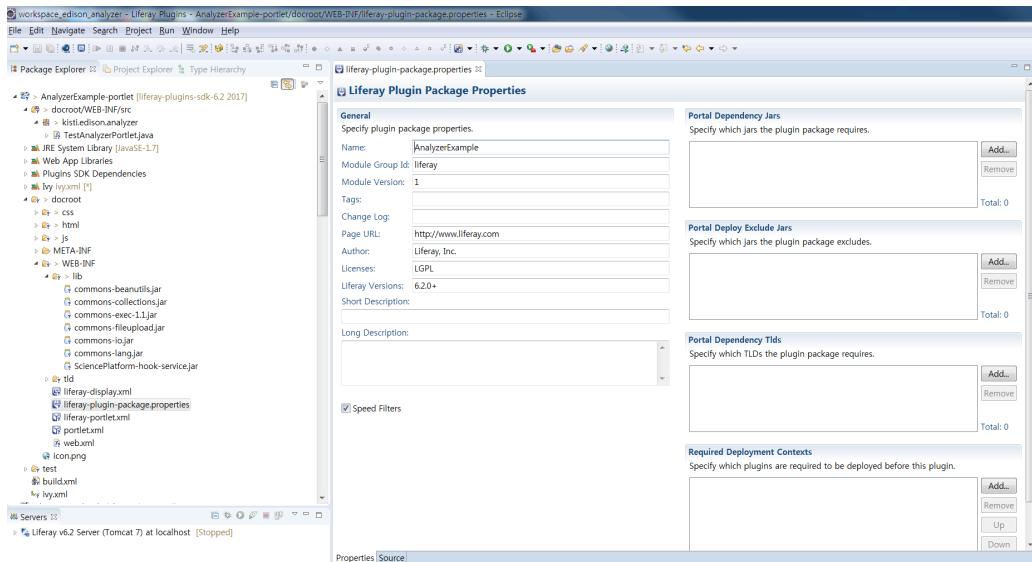


## 4. 기본 플러그인 추가 설정

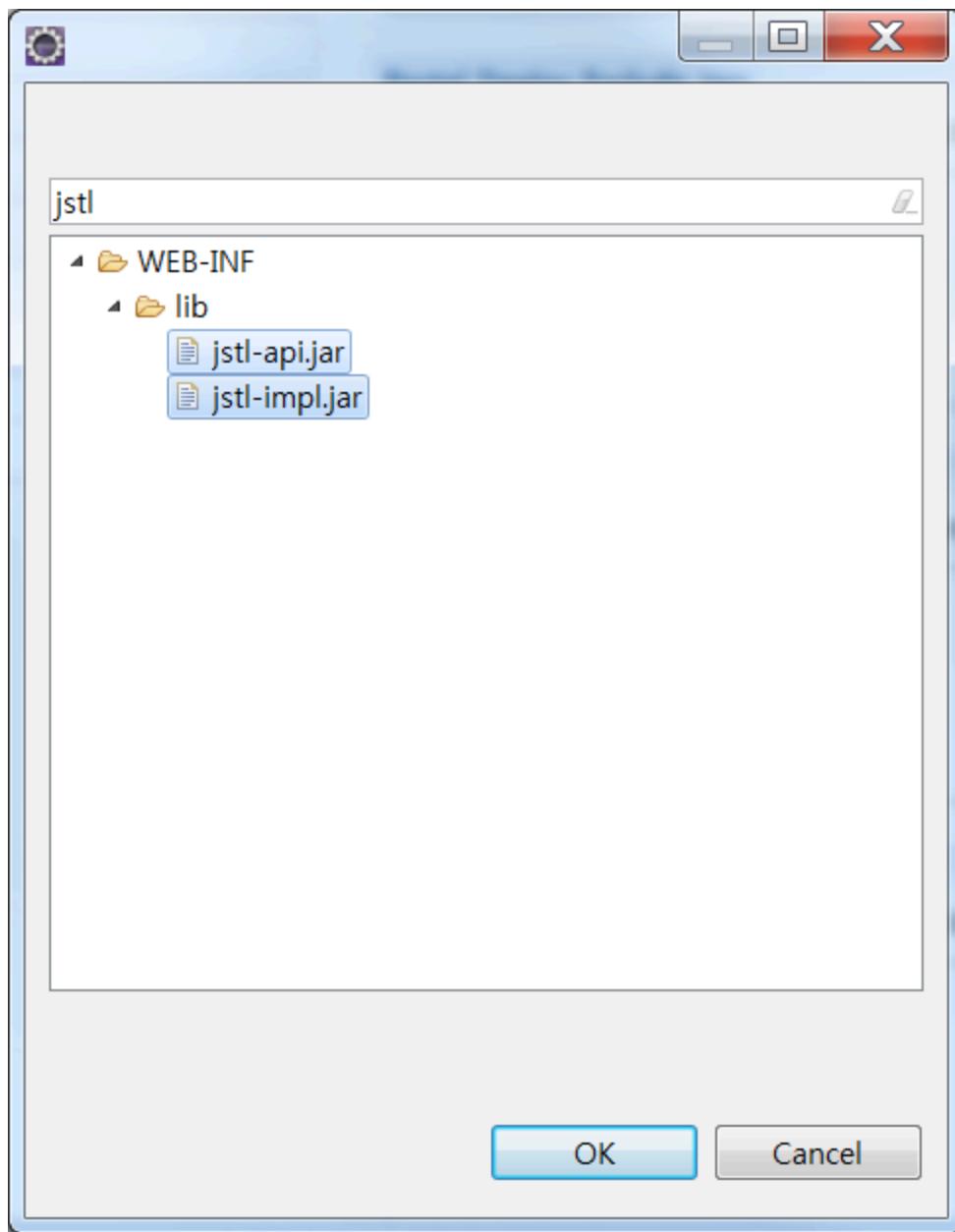
현재 Liferay Plugin Project에서 기본적으로 필요한 플러그인들을 추가합니다.

3번과 마찬가지로 xml 파일을 변경하는 것으로, liferay-plugin-package.properties 파일을 수정합니다.

기본적으로 이클립스에서 제공해주는 UI 환경에서 플러그인을 추가합니다. 해당 화면은 아래와 같습니다.

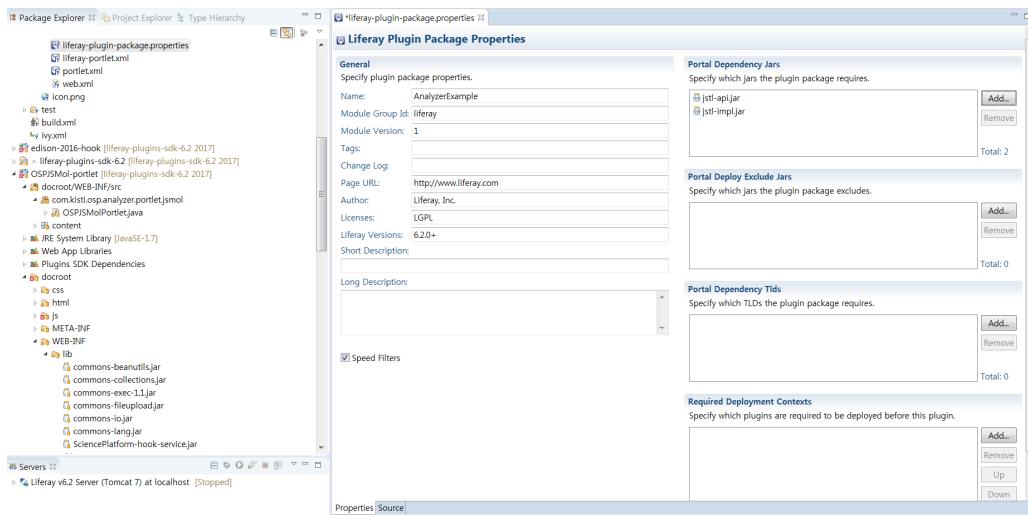


Portal Dependency Jars 섹션에서 Add... 버튼을 클릭한 후, jstl 을 검색해서 나오는 jstl-api.jar, jstl-impl.jar 파일 두개를 추가합니다.



## 5. 워크벤치 연동을 위한 플러그인 추가 설정

워크벤치 연동을 위해 필요한 라이브러리들을 현재 생성한 프로젝트 내 WEB-INF/lib 내 추가합니다.



**❶ Note:** 필요 라이브러리들을 추가합니다.

[commons-beanutils.jar \(page 0\)](#)

[commons-collections.jar \(page 0\)](#)

[commons-exec-1.1.jar \(page 0\)](#)

[commons-fileupload.jar \(page 0\)](#)

[commons-io.jar \(page 0\)](#)

[commons-lang.jar \(page 0\)](#)

[SciencePlatform-hook-service.jar \(page 0\)](#)

## 프로젝트 기본 설정

EDISON 워크벤치 기반 분석기(Post-Processor) 개발 매뉴얼 입니다.

Liferay 6.2.5 포틀릿 기반으로 개발 하는 방법에 대해 순차적으로 설명 되어 있습니다.

### Java 코드 추가 1 (파일 관리)

생성된 프로젝트에서 포틀릿 클래스에 파일 전송 및 파일 다운로드 등을 위한 자바 코드를 추가 합니다. `MVCPortlet` 을 `extends` 하게 되면 그 중 미리 선언되어 있는 `serveResource` 함수를 오버라이딩 하면 됩니다. 해당 코드는 수정할 필요없이 그대로 개발중인 프로젝트에 넣기만 하면됩니다. 이전 프로젝트 생성 부분에서 [SciencePlatform-hook-service.jar \(page 0\)](#)라이브러리를 추가했다면 다음 자바 코드를 문제없이 호출할 수 있습니다.

```
@Override
public void serveResource(ResourceRequest resourceRequest, ResourceResponse resourceResponse)
    throws IOException, PortletException{
    String fileName = ParamUtil.getString(resourceRequest, "fileName");
    String parentPath = ParamUtil.getString(resourceRequest, "parentPath");
    Path filePath = Paths.get(parentPath).resolve(fileName);
    String command = ParamUtil.getString(resourceRequest, "command");
    String repositoryType = ParamUtil.getString(resourceRequest, "repositoryType", OSPRepositoryTypes.USER_JOBS.toString());

    if(command.equalsIgnoreCase("GET_FILE")){
        try{
            OSPFileUtil.getFile(resourceRequest, resourceResponse, filePath.toString(), repositoryType);
        }catch (PortalException | SystemException e){
            _log.error("[JSMOL] readFileContent(): " + filePath.toString());
            throw new PortletException();
        }
    }else if(command.equalsIgnoreCase("READ_IMAGE")){
        try{
            OSPFileUtil.getFile(resourceRequest, resourceResponse, filePath.toString(), repositoryType);
        }catch (PortalException | SystemException e){
            _log.error("[JSMOL] readFileContent(): " + filePath.toString());
            throw new PortletException();
        }
    }else if(command.equalsIgnoreCase("READ_FIRST_FILE")){
        try{
            OSPFileUtil.readFirstFileContent(resourceRequest, resourceResponse, parentPath, fileName, repositoryType);
        }catch (PortalException | SystemException e){
            _log.error("[JSMOL] readFileContent(): " + filePath.toString());
            throw new PortletException();
        }
    }else if(command.equalsIgnoreCase("GET_FIRST_FILE_NAME")){
        try{
            System.out.println("[JSMOL] test get firstFileName");
        }
```

```
        OSPFileUtil.getFirstFileName(resourceRequest, r
esourceResponse, parentPath, fileName, repositoryType);
    }catch (PortalException | SystemException e){
        _log.error("[JSMOL] getFirstFileName(): " + filePat
h.toString());
        throw new PortletException();
    }
}else if(command.equalsIgnoreCase("DOWNLOAD_FILE")){
    try{
        OSPFileUtil.downloadFile(resourceRequest, resou
rceResponse, filePath.toString(), repositoryType);
    }catch (PortalException | SystemException e){
        _log.error("[JSMOL] checkValidFile(): " + filePat
h.toString());
        throw new PortletException();
    }
}else{
    _log.error("Un-known command: " + command);
    throw new PortletException();
}
}
```

## Java 코드 추가 2 (워크벤치 공유 변수)

워크벤치 실행을 위해 필요한 기본 값을 설정하기 위한 자바 코드를 추가합니다. 워크 벤치에서 개발하고 있는 분석기를 이용하여 시뮬레이션 환경을 구성할 때 필수적으로 확인하는 변수들의 값을 기본값으로 설정하는 부분입니다. 마찬가지로 `MVCPortlet` 을 `extends` 하게 되면 그 중 미리 선언되어 있는 `doView` 함수를 오버라이딩 하면 됩니다.

```
@Override  
public void doView(RenderRequest renderRequest, RenderResponse renderResponse) throws IOException, PortletException{  
    boolean eventEnable = ParamUtil.getBoolean(renderRequest, "eventEnable", true);  
    String inputData = ParamUtil.getString(renderRequest, "inputData");  
    String connector = ParamUtil.getString(renderRequest, "connector", "connector");  
    String mode = ParamUtil.getString(renderRequest, "mode", "VIEW");  
  
    renderRequest.setAttribute("eventEnable", eventEnable);  
    renderRequest.setAttribute("inputData", inputData);  
    renderRequest.setAttribute("connector", connector);  
    renderRequest.setAttribute("mode", mode);  
  
    super.doView(renderRequest, renderResponse);  
}
```

지금까지 기본적인 java 소스 코드를 추가하였습니다. 해당 소스코드는 워크벤치와 연동될 때, 파일 관리와 기본 변수 값을 체크하기 위해 필요한 부분입니다. 기본적으로 필요한 자바 소스코드 외에는 추가할 필요가 없는 부분이니 그냥 소스코드를 생성된 자바 파일에 복사해서 붙여넣기만 하면 됩니다.

### 로그 변수 선언

생성된 자바 파일 안에서 시스템에 필요한 로그를 선언하기 위해 변수를 선언할 필요가 있습니다.

```
'private static Log _log =  
LogFactoryUtil.getLog(TestAnalyzerPortlet.class);'
```

해당 변수는 위와 같이 선언하며, 생성된 자바 포틀릿 클래스 명에 따라 클래스명('TestAnalyzerPortlet.class')이 달라집니다.

### Java 코드 라이브러리 확인

위의 절차에 따라 자바 소스 코드들을 추가하였다면, 라이브러리들을 해당 클래스에 import 해야 합니다.

사용되는 클래스 라이브러리들은 아래와 같으니 동일한 이름의 다른 라이브러리를 호출하지 않도록 주의해야 합니다.

```
import java.io.IOException;
import java.nio.file.Path;
import java.nio.file.Paths;

import javax.portlet.PortletException;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import javax.portlet.ResourceRequest;
import javax.portlet.ResourceResponse;

import com.kisti.osp.constants.OSPRestoreTypes;
import com.kisti.osp.service.FileManagementLocalServiceUtil;
import com.kisti.osp.util.OSPFileUtil;
import com.liferay.portal.kernel.exception.PortalException;
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.util.ParamUtil;
import com.liferay.util.bridges.mvc.MVCPortlet;
```

## 스크립트 단위의 뷰 설정

이전에 설정되었던 부분은 서버 사이드에서 실행되는 컨트롤 코드 부분이었습니다. 워크 벤치 기반 분석기 개발을 위해서는 스크립트 단위의 이벤트 프로세싱을 통해 이루어 집니다.

따라서 뷰 부분의 스크립트 단위로 코드를 개발하며, 해당 뷰 부분에서 필요로 하는 기본 코드들이 있습니다. 따라서 그에 따른 스크립트 기본 소스 코드들을 추가하는 방법을 서술하도록 하겠습니다.

### init.jsp 파일 생성 및 코드 입력

생성된 프로젝트 내 `/html` 폴더 내 `init.jsp` 파일을 생성합니다.

생성된 `init.jsp` 파일 내 뷰 개발을 위한 필요 라이브러리 코드를 추가하고 라이프레 이 기본 라이브러리들을 추가합니다. 추가해야 하는 코드는 아래와 같습니다.

```
<%@ taglib uri="http://java.sun.com/portlet_2_0" prefix="portlet" %>
<%@ taglib uri="http://alloy.liferay.com/tld/aui" prefix="aui" %>
<%@ taglib uri="http://liferay.com/tld/ui" prefix="liferay-ui" %>
<%@ taglib uri="http://liferay.com/tld/theme" prefix="theme" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>
<%@ taglib uri="http://liferay.com/tld/security" prefix="liferay-security" %>
<%@ taglib uri="http://liferay.com/tld/portlet" prefix="liferay-portlet" %>

<%@ taglib uri="http://java.sun.com/jsp/jstl/fmt" prefix="fmt" %>
<%@ taglib uri="http://java.sun.com/jsp/jstl/functions" prefix="fn" %>
<!-- JQuery -->
<script src="<%=request.getContextPath()%>/js/jquery/jquery-ui.min.js" ></script>
<script src="<%=request.getContextPath()%>/js/jquery/jquery.blockUI.js" ></script>
<link type="text/css" href="<%=request.getContextPath()%>/js/jquery/jquery-ui.css" rel="stylesheet" />

<link rel="stylesheet" type="text/css" href="<%=request.getContextPath()%>/css/main.css">
<link href="<%=request.getContextPath()%>/js/jquery/bootstrap-toggle.min.css" rel="stylesheet">
<script src="<%=request.getContextPath()%>/js/jquery/bootstrap-toggle.min.js"></script>

<!-- bootstrap -->
<link href="<%=request.getContextPath()%>/js/jquery/bootstrap.min.css" rel="stylesheet">
<script src="<%=request.getContextPath()%>/js/jquery/bootstrap.min.js"></script>

<portlet:defineObjects />
<theme:defineObjects />
```

**스크립트 라이브러리:**

jqueryxxx.min.js 사용 (워크벤치 시스템과 충돌) 스크립트 라이브러리 충돌이 나서 워크벤치 시뮬레이션이 동작하지 않을 수 있습니다. 따라서 jquery 스크립트를 비롯한 대부분의 스크립트 파일은 `init.jsp` 파일에 추가하지 마시기 바랍니다. 하지만 JQuery 스크립트는 플랫폼에 설치되어 있으니 라이브러리를 사용할 수 있습니다.

## 워크벤치 연동을 위한 CSS 설정

`main.css` 파일에서 `osp-analyzer.css` (page 0) 파일을 호출합니다.

```
@import url("./osp-analyzer.css");
```

`osp-analyzer.css` 파일을 `main.css` 파일과 같은 폴더( `/css` )안에 추가합니다.

`osp-analyzer.css` 스타일 코드는 아래와 같습니다.

```
.osp-analyzer {  
    border:none;  
    height: 100%;  
    min-height:800px;  
    margin: 0;  
}  
.osp-analyzer .header{  
    height: 40px;  
    margin:10px 5px 10px 5px;  
}  
.osp-analyzer .header [class*="col-"]{  
    padding-left: 0;  
    padding-right: 0;  
}  
.osp-analyzer .no-header-frame{  
    vertical-align:middle;  
    border:none;  
    height: 100%;  
}  
.osp-analyzer .frame{  
    vertical-align:middle;  
    border:none;  
    height: 90%;  
}  
.osp-analyzer .canvas {  
    border:none;  
    height:100%;  
    margin:0;  
    padding:0;  
  
    overflow:auto;  
}  
.osp-analyzer .iframe-canvas {  
    border:none;  
    height:100%;  
    margin:0;  
    padding:0;  
  
    overflow:hidden;  
}  
.osp-analyzer .icon-menu {  
    font-size: 15px;  
    color: #5cb2d7;  
    border: 2px solid;
```

```
border-radius: 3px;  
padding-left: 2px;  
padding-right: 3px;  
vertical-align: middle;  
}  
  
.osp-analyzer .hidden {  
    display: none;  
}
```

### jsp 파일 뷰와 분석기 호출부 처리

기본 워크벤치 연동을 위한 이벤트 처리를 하는 jsp 파일(testview.jsp)과 실제 분석하는 부분을 사용자에게 서비스하고 데이터를 가시화하여 제공하는 부분의 jsp 파일(loadAnalyzer.jsp)를 프로젝트 내 생성하여 데이터를 처리한다. 파일명은 임의로 설정 할 수 있지만 `portlet.xml` 파일에서 설정을 잘 잡아주어야 합니다. 이벤트 처리를 하는 jsp 파일을 포틀릿과 연결된 초기 jsp 파라미터로 설정해 주어야 합니다.

이벤트 처리를 하는 jsp 파일의 파일명을 testview.jsp로 했다면, `portlet.xml` 파일의 설정은 다음과 같습니다.

```
<init-param>  
    <name>view-template</name>  
    <value>/html/testanalyzer/testview.jsp</value>  
</init-param>
```

지금까지 설정에 따른 전체 프로젝트의 구조는 다음과 같습니다.

```
|AnalyzerExample-portlet/  
|---docroot/  
|   |---css/  
|   |   |---main.css  
|   |   |---osp-analyzer.css  
|   |---html/  
|   |   |---testanalyzer/  
|   |   |   |---testview.jsp  
|   |   |   |---loadAnalyzer.jsp  
|   |---js/  
|   |   |---main.js  
|   |---META-INF/  
|   |   |---MANIFEST.MF  
|   |---WEB-INF/  
|   |   |---lib/  
|   |   |   |---commons-beanutils.jar  
|   |   |   |---commons-collections.jar  
|   |   |   |---commons-exec-1.1.jar  
|   |   |   |---commons-fileupload.jar  
|   |   |   |---commons-io.jar  
|   |   |   |---commons-lang.jar  
|   |   |   |---SciencePlatform-hook-service.jar  
|   |---tlb/  
|   |   |---liferay-display.xml  
|   |   |---liferay-plugin-packkage.properties  
|   |   |---liferay-portlet.xml  
|   |---portlet.xml  
|   |---web.xml
```

이외에 추가로 파일이 더 있을 수도 있지만 필수적으로 필요한 파일들은 위의 구조와 같으니 비교하여 체크해 보시기 바랍니다.

# 분석기 구조 설명

DISON 플랫폼에서 워크벤치와 연동되는 분석기는 모듈 단위로 개발 될 수 있으며, 각 모듈은 Liferay 포틀릿으로 구분된다. 각 포틀릿에서 생성된 가시화 및 서비스 등은 EDISON 플랫폼에서 제공되는 서비스와 연동되어 개발될 수 있습니다.

## 분석기와 EDISON 연동 시스템 구조

EDSION 워크벤치 시뮬레이션 시스템과 연동되는 시스템 구조는 다음 그림과 같이 표현 될 수 있습니다.

imagetestYejin

분석기는 전체 이벤트를 처리하는 이벤트 처리부분과 내부적으로 데이터를 가시화하고 분석 할 수 있는 Analyzer 구현 부분으로 나눠질 수 있습니다.

EDISON 플랫폼에서 파생되는 이벤트를 받아들이고 적절한 시기에 사용자에게 데이터 분석을 위한 툴을 제공하는 부분을 포함하고 있습니다.

따라서 EDISON 워크벤치와 연동되는 분석기 모듈을 개발하기 위해서는 이벤트를 처리 하는 부분과 실제 데이터 가시화로 위한 부분을 나눠서 개발을 해야 합니다.

## 분석기 이벤트 처리 부분

워크벤치의 분석기를 개발하는데 있어 이벤트를 처리하는 부분은 자바 스크립트로 이루 어져 있으며, Liferay 포틀릿 기반으로 이벤트를 받아들입니다.

다음 예제 코드는 워크벤치가 실행되면서 분석기를 호출하고 이에 대한 응답을 하는 코드입니다. Liferay 에서 제공하는 `Liferay.fire()` 함수와 `Liferay.on()` 기반으로 이벤트 프로세싱을 제공합니다.

- 샘플 예제

```

Liferay.on(
    OSP.Event.OSP_HANDSHAKE,
    function(e){
        var myId = '<%=portletDisplay.getI
d()%>';
        if( e.targetPortlet !== myId ){
            return
        }
        console.log(' [NGLViewer]OSP_HANDSHAKE:
['+e.portletId+', '+new Date()+']');
        <portlet:namespace/>connector = e.portle
tId;

        if( e.mode )
            <portlet:namespace/>action = e.m
ode;
        else
            <portlet:namespace/>action = 'VI
EW';

        var events = [
            OSP.Event.OSP_EVENTS_REGISTERED,
            OSP.Event.OSP_LOAD_DATA
        ];
        var eventData = {
            portletId: myId,
            targetPortlet: <portlet:namespac
e/>connector,
            data: events
        };
        Liferay.fire( OSP.Event.OSP_REGISTER_EVENTS, event
Data );
    }
);

```

- `Liferay.fire()` : 이벤트를 발생시키는 Liferay 자바스크립트 라이브러리 함수입니다.
- `Liferay.on()` : 이벤트를 리스닝 하는 Liferay 자바스크립트 라이브러리 함
수입니다.

## 데이터 처리 부분

실제 서버에서 계산과학공학 시뮬레이션이 실행되고 결과 값을 분석하기 위해서는 시뮬레이션 결과를 호출하고 적절한 데이터를 받아 가시화 하는 데이터 처리 구현 부분입니다. 계산 결과 데이터를 수신하고 가시화 하기 위해서는 데이터 호출 및 응답에 대한 코드 구현이 필수적입니다.

이미 이전 단계에서 추가한 OSP 라이브러리를 기반으로 데이터를 호출하고 적절한 데이터를 받는 코드 부분이 구현 되어야 합니다.

이러한 데이터를 구현하기 위해서는 다양한 스크립트 함수들이 필요하며, 해당 함수들을 순차대로 정리하면 다음과 같습니다.

### 데이터 처리를 위해 정의가 필요한 함수

함수명	정의
loadXXXXFile(OSPInputData)	실제 서버 사이드의 파일을 읽어오는 방법에 대한 정의 Output 포트의 path 태입에 따라 케이스를 나눠 처리 방법에 대한 정의가 되어 있음 폴더/확장자/파일
getFirstFileName(OSPInputData)	결과 포트의 path 태입이 폴더 또는 확장자인 경우, 해당 폴더 내 첫번째 파일을 읽어오는 역할이 정의

#### 데이터 처리 방법

서버에서 시뮬레이션 결과로 생성된 결과 데이터는 OSP 라이브러리를 이용하여 전송받을 수 있으며, 해당 데이터를 받은 다음 iframe 으로 연결된 실제 데이터 가시화 부분에 데이터를 전공하는 방법으로 분석기가 동작하게 됩니다.

## 초기값 설정

EDISON 워크벤치와 연동을 하며 워크벤치에서 요구하는 이벤트를 처리하기 위해서는 기본적으로 셋팅이 완료되어야 하는 몇 가지 초기 값이 있습니다.

### 데이터 전송 및 라이브러리 사용

다음은 데이터 전송 및 기타 라이브러리들을 사용하기 위한 기본 코드입니다.

`testview.jsp` 파일 내부에 제일 위쪽 부분에 해당 라이브러리 호출 부분을 포함합니다.

```
<%@page import="com.kisti.osp.constants.OSPRepositoryTypes"%>
<%@page import="com.liferay.portal.kernel.util.GetterUtil"%>
<%@page import="com.liferay.portal.util.PortalUtil"%>
<%@page import="com.liferay.portal.kernel.portlet.LiferayWindowState"%>
<%@page import="javax.portlet.PortletPreferences"%>
<%@include file="../init.jsp"%>
```

### 라이프레이 URL 추가

라이프레이 데이터 전송 방식을 사용하기 위한 URL을 추가합니다.

데이터 전송을 위한 Liferay 포틀릿의 URL은 resourceURL입니다.

```
<portlet:resourceURL var="serveResourceURL"></portlet:resourceU
RL>
<portlet:renderURL var="renderURL">
    <portlet:param name="jspPage" value="/html/testanalyzer/tes
tview.jsp"/>
</portlet:renderURL>
```

### 초기 데이터 셋팅을 위한 코드 추가

워크벤치와 연동되어 동작하기 위해 필요한 기본값들의 초기 셋팅 값입니다. 이벤트 처리를 하는 워크벤치와 연동을 할 것인지 아니면 독립적으로 사용자에게 제공되어 동작할 것인지 호출하고 세팅하는 값이 포함되어 있습니다.

```
<%
PortletPreferences preferences = portletDisplay.getPortletSetup();
preferences.setValue("portletSetupShowBorders", String.valueOf(Boolean.FALSE));
preferences.store();

String inputData = GetterUtil.getString(renderRequest.getAttribute("inputData"), "{}");
String connector = (String)renderRequest.getAttribute("connector");
String mode = GetterUtil.getString(renderRequest.getAttribute("mode"), "VIEW");
boolean eventEnable = GetterUtil.getBoolean(renderRequest.getAttribute("eventEnable"), true);
%>
```

## 가시화 태그

워크벤치의 분석기를 개발하는데 있어 이벤트를 처리하는 부분은 자바 스크립트로 이루어져 있으며, Liferay 포틀릿 기반으로 이벤트를 받아들입니다.

다음 예제 코드는 워크벤치가 실행되면서 분석기를 호출하고 이에 대한 응답을 하는 코드입니다. Liferay에서 제공하는 `Liferay.fire()` 함수와 `Liferay.on()` 기반으로 이벤트 프로세싱을 제공합니다.

- 샘플 예제

```

Liferay.on(
    OSP.Event.OSP_HANDSHAKE,
    function(e){
        var myId = '<%=portletDisplay.getI
d()%>';
        if( e.targetPortlet !== myId ){
            return
        }
        console.log(' [NGLViewer]OSP_HANDSHAKE:
['+e.portletId+', '+new Date()+']');
        <portlet:namespace/>connector = e.portle
tId;

        if( e.mode )
            <portlet:namespace/>action = e.m
ode;
        else
            <portlet:namespace/>action = 'VI
EW';

        var events = [
            OSP.Event.OSP_EVENTS_REGISTERED,
            OSP.Event.OSP_LOAD_DATA
        ];
        var eventData = {
            portletId: myId,
            targetPortlet: <portlet:namespac
e/>connector,
            data: events
        };
        Liferay.fire( OSP.Event.OSP_REGISTER_EVENTS, event
Data );
    }
);

```

- `Liferay.fire()` : 이벤트를 발생시키는 Liferay 자바스크립트 라이브러리 함수입니다.
- `Liferay.on()` : 이벤트를 리스닝 하는 Liferay 자바스크립트 라이브러리 함
수입니다.

# 파일 탐색기 설정

## 파일 메뉴 설정

기본적으로 분석기를 EDISON 워크벤치와 연동을 하면서 필요한 파일 메뉴는 3가지입니다. 현재 대부분의 워크벤치와 연동된 파일 메뉴는 로컬 파일 불러오기, 서버 파일 불러오기, 현재 파일 다운로드하기 가 있습니다. 이러한 파일 메뉴와 더불어 실제 파일을 불러와 가시화하는 가시화 부분을 호출하는 `iframe` 도 같이 포함하여 정의합니다. 이러한 파일 메뉴는 이전 포스트에서 설정되어 있는 [osp-analyzer.css \(page 0\)](#)에서 정의되어 있는 스타일을 사용합니다.

다음은 파일 메뉴를 설정하기 위한 html 태그 코드입니다.

```
<div class="container-fluid osp-analyzer">
    <div class="row-fluid header">
        <div class="col-sm-10" id="
```

## 파일 다이얼로그 추가

서버와 연동되는 파일 처리를 위한 서버 파일 리스트를 볼 수 있는 다이얼로그를 띄울 수 있는 태그를 추가합니다.

또한 서버의 파일 리스트를 볼 수 있는 서버 파일 탐색기를 호출하게 됩니다. 이 서버 사이드 파일 탐색기는 EDISON 플랫폼에 탑재되어 있으며, 분석기를 개발하려면 호출하기만 하면 됩니다. 호출하는 함수는 `OSPFileExplorerportlet`로서 자바스크립트로 정의되어 있습니다.

```
<div id="hiddenSection" class="osp-analyze
r hidden">
    <div id="fileExplorer" class="pane
l panel-primary ui-draggable" style="padding:0px;margin-botto
m:0px;">
        <!-- title -->
        <div class="panel-heading">
            <button type="button" class="close" dat
a-dismiss="modal" aria-hidden="true" id='<portlet:namespace/>cl
oseDialog'>&times;</button>
            <h4>Select a File</h4>
        </div>

        <!-- content -->
        <div class="panel-body" id="file-explorer-content" style="height: 81%"></div>

        <!-- bottom -->
        <div class="panel-footer">
            <div class="ui-dialog-buttonset">
                <input class="btn btn-primary"
id="file-explorer-ok" type="button" valu
e="OK">
                <input class="btn" id="file-explorer-cancel" type="button" value="Cance
l">
            </div>
        </div>

        </div>
        <input type="file" id="selectFil
e"/>
    </div>
```

## 파일 메뉴 이벤트 처리

- 로컬 파일 처리 함수

```
$('&#lt;portlet:namespace>selectFile').bind(  
    'change',  
    function(event){  
        var input = document.getElementById('<portlet:n  
amespace/>selectFile');  
        var reader = new FileReader();  
  
        reader.onload = function (e) {  
            $('#<portlet:namespace>canvas').each(f  
unction(){  
                $(this).prop('contentWindow').l  
oadJSMolFile(e.target.result);  
            });  
            console.log("[JSMOL] local file test : ", inpu  
t);  
            console.log("[JSMOL] local file test2  
: ", input.files[0]);  
  
            <portlet:namespace>setTitle(input.fil  
es[0].name);  
        };  
  
        reader.readAsDataURL(input.files[0]);  
    }  
);
```

- 서버 파일 처리 함수 `initializeFileExplorer` 함수는 서버 파일 탐색기를 호출하기 위해 초기값을 설정하는 코드입니다.

```
function <portlet:namespace>initializeFileExplorer(){
    if( $.isEmptyObject(<portlet:namespace>initData) ||(
        <portlet:namespace>initData.type() !== OSP.Enumeration.PathType.FILE &&
        <portlet:namespace>initData.type() !== OSP.Enumeration.PathType.FOLDER &&
        <portlet:namespace>initData.type() !== OSP.Enumeration.PathType.EXT ) )      return;

    var eventData = {
        portletId: '<%=portletDisplay.getId()%>',
        targetPortlet: <portlet:namespace>fileExplorerId,
        data: OSP.Util.toJSON(<portlet:namespace>initData)
    };

    Liferay.fire( 'OSP_LOAD_DATA', eventData );
}
```

`openFileExplorer` 함수는 사용자가 서버 파일 오픈 메뉴를 선택하였을 때, 다이얼로그 형태로 사용자에게 서버쪽 파일 리스트를 탐색한 내용을 업데이트 하여 보여주는 함수 부분입니다.

```

function <portlet:namespace>openFileExplorer(){
    AUI().use('liferay-portlet-url', function(A){
        if($("#<portlet:namespace>file-explorer-content").children().length > 0){
            $<portlet:namespace>fileExplorerDialog
Section.dialog("open");
        }else{
            var inputData;
            if(      !$.isEmptyObject(<portlet:na
mespace/>initData) && (
                <portlet:namespace>initData.ty
pe() === OSP.Enumeration.PathType.FILE ||
                <portlet:namespace>initData.ty
pe() === OSP.Enumeration.PathType.FOLDER ||
                <portlet:namespace>initData.ty
pe() === OSP.Enumeration.PathType.EXT )) {
                inputData = <portlet:namespac
e>initData;
            }
            else{
                inputData = new OSP.InputDat
a();
                inputData.repositoryType( '<%=0
SPRepositoryTypes.USER_HOME.toString()%>');
                inputData.type( OSP.Enumeratio
n.PathType.FOLDER );
                inputData.parent('');
                inputData.name('');
            }
        }

        var dialogURL = Liferay.PortletURL.crea
teRenderURL();
        dialogURL.setPortletId(<portlet:namespa
ce/>fileExplorerId);
        dialogURL.setParameter('inputData', JSO
N.stringify(inputData));
        dialogURL.setParameter('mode', 'VIEW');
        dialogURL.setParameter('eventEnable', f
alse);
        dialogURL.setParameter('connector',
'<portletDisplay.getId()%>');
        dialogURL.setWindowState('<%=LiferayWin
dowState.EXCLUSIVE%>');

        $("#<portlet:namespace>file-explorer-c

```

```
ontent").load( dialogURL.toString());
                $<portlet:namespace/>fileExplorerDialog
Section.dialog("open");
}
});
}
```

- 현재 파일 다운로드 현재 선택되어 있는 파일을 다운로드 하는 함수입니다.

```
function <portlet:namespace/>downloadCurrentFile(){
console.log("[testView] Download current data", <portlet:namespace/>currentData);
if( $.isEmptyObject(<portlet:namespace/>currentData) ||
    <portlet:namespace/>currentData.type() !== OSP.Enumeration.PathType.FILE )
    return;

var filePath = <portlet:namespace/>currentData;
var data = {
    <portlet:namespace/>command: "DOWNLOAD_FILE",
    <portlet:namespace/>pathType: filePath.type_,
    <portlet:namespace/>repositoryType: filePath.repositoryType_,
    <portlet:namespace/>parentPath: filePath.parent_,
    <portlet:namespace/>fileName: filePath.name_
};

var base = '<%=serveResourceURL.toString()%>';
var sep = (base.indexOf('?') > -1) ? '&' : '?';
var url = base + sep + $.param(data);

location.href = url;
<portlet:namespace/>loadXXXXFile( <portlet:namespace/>currentData );
}
```

# 이벤트 처리 함수

## 이벤트 처리

개발하려는 분석기는 Liferay에서 제공하는 이벤트 전송 및 수신 함수(Liferay.on(), Liferay.fire())를 사용하여 워크벤치와 연동하여 동작합니다. 자바 스크립트 단위로 동작을 하기 때문에 개발하려는 분석기 내에 필요한 이벤트를 처리하고 발생시키는 루틴을 넣어야 합니다.

분석기에서 사용되는 이벤트의 종류와 설명은 아래와 같습니다.

이벤트명	정의
OSP.Event.OSP_HANDSHAKE	초기 워크벤치에 해당 모듈을 등록하고 초기 설정값을 워크벤치로 부터 받을 때 쓰는 이벤트
OSP.Event.OSP_Event_REGISTERED	초기 OSP_HANDSHAKE에 대한 워크벤치의 결과 응답 이벤트
OSP.Event.OSP_LOAD_DATA	특정 위치와 파일에 대한 데이터 로드 이벤트
OSP.Event.OSP_RESPONSE_DATA	사용자가 파일 익스플로러에서 선택한 파일에 대한 응답 이벤트
OSP.Event.OSP_REFRESH_OUTPUT_VIEW	포틀릿 아이디 공유를 위한 이벤트
OSP.Event.OSP_INITIALIZE	워크벤치에서 보내는 초기화 값 설정을 위한 이벤트

## 이벤트처리함수

일단 개발하려는 분석기가 동작하기 위해서는 위의 이벤트를 워크벤치로부터 받아서 처리하는 루틴이 필수적으로 필요합니다. 따라서 기본으로 해당 이벤트를 처리하는 코드를 작성하고 나머지 가시화 및 분석을 위한 코드를 추가하는 것이 좋습니다.

## OSP.Event.OSP\_HANDSHAKE

처음 워크벤치가 실행되면서 각 포틀릿에게 OSP.Event.OSP\_HANDSHAKE이벤트를 보내게 됩니다. 따라서 워크벤치와 연동되는 분석기를 개발하기 위해서는 OSP.Event.OSP\_HANDSHAKE 이벤트를 받아서 다시 워크벤치에게 전달하는 루틴이 필요합니다. 아래 코드에서 볼 수 있듯이, OSP.Event.OSP\_HANDSHAKE이벤트를 받 아들이고 현재 동작중인 분석기를 워크벤치에 등록시키는 과정을 포함합니다.

```
Liferay.on(
    OSP.Event.OSP_HANDSHAKE,
    function(e){
        var myId = '<%=portletDisplay.getId()%>';
        if( e.targetPortlet !== myId ){
            return
        }
        <portlet:namespace>connector = e.portletId;

        if( e.mode )
            <portlet:namespace>action = e.mode;
        else
            <portlet:namespace>action = 'VIEW';
        var events = [
            OSP.Event.OSP_EVENTS_REGISTERED,
            OSP.Event.OSP_LOAD_DATA
        ];
        var eventData = {
            portletId: myId,
            targetPortlet: <portlet:namespace>co
nnector,
            data: events
        };

        Liferay.fire( OSP.Event.OSP_REGISTER_EVENTS, ev
entData );
    }
);
```

### OSP.Event.OSP\_Event\_REGISTERED

OSP\_HANDSHAKE 이벤트의 결과에 대한 이벤트입니다. 간단하게 이벤트를 받아들여 체크하는 정도의 코드만 추가하면 됩니다.

```
Liferay.on(
    OSP.Event.OSP_EVENTS_REGISTERED,
    function(e){
        var myId = '<%=portletDisplay.getId()%>';
        if(e.targetPortlet === myId){
            console.log('[TestView] Registered'+e.portletId+' activated. '+new Date());
        }
    }
);
```

### OSP.Event.OSP\_LOAD\_DATA

데이터를 로드하기 위한 함수입니다. 워크벤치를 기반으로 하여 시뮬레이션을 수행하는데 있어, 시뮬레이션이 성공적으로 종료되었거나 이전에 실행한 성공한 시뮬레이션 잡의 결과 데이터를 가져오기 위한 이벤트 처리 부분입니다.

워크벤치에서 데이터를 로드하고 이벤트를 발생시키고, 해당 이벤트를 받아 분석에서는 데이터를 로드하여 분석할 수 있도록 시각화를 하면 됩니다.

- `<portlet:namespace/>initialize(e.data);` 에서는 초기 데이터 셋팅 값을 받은 이벤트 데이터를 기반으로 설정합니다.
- `<portlet:namespace/>loadTestviewFile(<portlet:namespace/>initData.clone());` 코드는 실제 데이터를 가져와 가시화를 하기 위한 코드입니다.
- `<portlet:namespace/>initializeFileExplorer();` 코드는 현재 받은 이벤트 데이터를 기반으로 서버와 연결된 파일 익스프로러의 값을 재설정 하는 부분입니다.

```
Liferay.on(
    OSP.Event.OSP_LOAD_DATA,
    function(e){
        var myId = '<%=portletDisplay.getId()%>';
        if( e.targetPortlet !== myId )
            return;

        <portlet:namespace/>initialize(e.data);
        <portlet:namespace/>loadTestviewFile(<portlet:namespace/>initData.clone());

        <portlet:namespace/>initializeFileExplorer();

    }
);
```

## OSP.Event.OSP\_RESPONSE\_DATA

사용자가 서버와 연동된 파일 익스플로러에서 선택한 파일에 대한 응답 이벤트 코드입니다. 사용자가 파일을 선택하면 해당 파일 데이터를 읽어오기 위한 이벤트 처리 부분입니다.

```
Liferay.on(
    OSP.Event.OSP_RESPONSE_DATA,
    function( e ){
        var myId = '<%=portletDisplay.getId()%>';
        if( myId !== e.targetPortlet ) return;

        var inputData = new OSP.InputData( e.data );

        if( inputData.type() !== OSP.Enumeration.PathType.FILE ){
            alert( 'File should be selected!' );
            return;
        }else{
            <portlet:namespace/>loadTestviewFile( inputData );
            $<portlet:namespace/>fileExplorerDialogSection.dialog('close');
        }
    }
);
```

## OSP.Event.OSP\_REFRESH\_OUTPUT\_VIEW

포틀릿의 아이디를 공유하기 위한 이벤트 처리 부분입니다.

```
Liferay.on(
    OSP.Event.OSP_REFRESH_OUTPUT_VIEW,
    function(e){
        var myId = '<%=portletDisplay.getId()%>';
        if( myId !== e.targetPortlet ) return;

        var eventData = {
            portletId: '<%=portletDisplay.getId()%>',
            targetPortlet: <portlet namespace/>connector
        };

        Liferay.fire(OSP.Event.OSP_REQUEST_OUTPUT_PATH, eventData);
    }
);
```

## OSP.Event.OSP\_INITIALIZE

처음 HANDSHAKE 이벤트를 처리하면서 내부저궁로 서버와 연동된 파일 익스플로러를 초기화 시키는 코드입니다.

```
Liferay.on(
    OSP.Event.OSP_INITIALIZE,
    function(e){
        var myId = '<%=portletDisplay.getId()%>';
        if(myId !== e.targetPortlet) return;

        if( $.isEmptyObject(<portlet:namespace/>initData) )
            return;
        <portlet:namespace/>initializeFileExplorer( <portlet:namespace/>initData.clone() );
    }
);
```

# 공용 함수

## 공용함수

이벤트에 대한 응답이나 데이터를 처리하고 실제 가시화를 하기 위한 공용 함수들이 필요합니다. 예를 들어 워크벤치와 연동되어 시뮬레이션을 수행할 때, 시뮬레이션이 종료되고 결과를 확인하기 위해서는 워크벤치에서 데이터를 로드하라는 OSP\_LOAD\_DATA 이벤트를 분석기에 보내게 됩니다.

해당 이벤트를 해석하고 데이터를 로드하는 함수를 정의하고 로드된 데이터를 통해 시뮬레이션 분석을 위한 가시화를 진행할 수 있습니다. 따라서 데이터를 처리하고 파일을 분석하는 내용에 대한 공용 함수를 정의합니다.

분석기에서 사용되는 함수의 종류와 설명은 아래와 같습니다.

함수명	정의
loadXXXXXXFile(OSPInputData)	실제 서버쪽의 파일을 읽어오는 방법에 대한 정의 Output 포트의 path 타입에 따라 케이스를 나누고 처리 방법에 대한 정의가 되어 있음
getFirstFileName(OSPInputData)	Output 포트의 파일 타입이 폴더이거나 확장자일 경우 해당 결과 result 폴더에서 첫 번째 파일을 읽어오도록 정의
drawXXXXXXView()	서버와 연동되어서 선택된 파일을 실제 뷰 부분의 iframe 내부로 전달하여 가시화 함
initializeFileExplorer()	서버와 연결된 파일 익스플로러 초기값 설정
DownloadCurrentFile()	현재 사용자가 파일 익스플로러에서 선택한 파일을 다운로드

loadXXXXXXFile(OSPInputData)

```
function <portlet:namespace/>loadTestviewFile( inputData ){
    <portlet:namespace/>currentData = inputData;
    console.log("[testView] Load Data : input Data ", inputData);
    switch( inputData.type() ){
        case OSP.Enumeration.PathType.FILE:
            <portlet:namespace/>drawView( inputData );
            break;
        case OSP.Enumeration.PathType.FOLDER:
        case OSP.Enumeration.PathType.EXT:
            <portlet:namespace/>getFirstFileName( inputData );
            // serveResourceUrl.setParameter('command', 'READ_FIRST_FILE');
            break;
        case OSP.Enumeration.PathType.URL:
            alert('Un supported yet.'+inputData.type());
            break;
        default:
            alert('Un supported yet.'+inputData.type());
    }
}
```

drawXXXXXXView()

```
function <portlet:namespace>drawView( inputData ){
    setTimeout(
        function(){
            var iframe = document.getElementById('<portlet:namespace>canvas');
            var iframeDoc = iframe.contentDocument || iframe.contentWindow.document;

            console.log( '[testView]iframeDoc.readyState : ', iframeDoc.readyState);
            if ( iframeDoc.readyState == 'complete' && iframe.contentWindow.drawTestViewer ) {

                console.log("[testView] Load data in draw");
                AUI().use('liferay-portlet-url', function(a) {
                    <portlet:namespace>currentData = inputData.clone();
                    if( !<portlet:namespace>currentData.repositoryType() )
                        <portlet:namespace>currentData.repositoryType('<%=OSPRepositoryTypes.USER_JOBS.toString()%>');

                    var serveResourceUrl = Liferay.PortletURL.createResourceURL();

                    serveResourceUrl.setPortletId('<portletDisplay.getId()%>');
                    serveResourceUrl.setParameter('command', 'GET_FILE');
                    serveResourceUrl.setParameter('repositoryType', <portlet:namespace>currentData.repositoryType());
                    serveResourceUrl.setParameter('pathType', inputData.type());
                    serveResourceUrl.setParameter('parentPath', inputData.parent());
                    serveResourceUrl.setParameter('fileName', inputData.name());
                    serveResourceUrl.setParameter('relative', inputData.relative());

                    console.log( '[testView]Draw Input Data : ', inputData);
                });
            }
        }
    );
}
```

```
        iframe.contentWindow.drawTestVi  
ewer(inputData, serveResourceUrl.toString());  
  
        $( '#<portlet:namespace>titl  
e' ).html( inputData.name() );  
    } );  
}  
else{  
    console.log("[testView] test esle  
: load data in draw");  
  
    <portlet:namespace/>drawView( input  
Data );  
}  
},  
10  
);  
}
```

`getFirstFileName(OSPInputData)`

```
function <portlet:namespace>getFirstFileName( inputData ){
    console.log('[testView]get First File Name : ', inputData );

    var data = {
        <portlet:namespace>command: 'GET_FIRST_FILE_NAME',
        <portlet:namespace>pathType: inputData.type_,
        <portlet:namespace>repositoryType: inputData.repositoryType_,
        <portlet:namespace>parentPath: inputData.parent_,
        <portlet:namespace>fileName: inputData.name_
    };
    console.log("[testView] laod get first file test : ", data);

    $.ajax({
        url: '<%= serveResourceURL.toString()%>',
        type: 'POST',
        data : data,
        dataType : 'json',
        success: function(data) {
            console.log("[testView] get result data ", data);
            inputData.name( data.fileName );
            inputData.type( OSP.Enumeration.PathType.FILE );
            <portlet:namespace>drawView( inputData );
            console.log("[testView] Get First File Data : ", inputData);
        },
        error:function(data,e){
            console.log(' [testView]AJAX ERROR1-->', data);
            console.log(' [testView]AJAX ERROR2-->', e);
        },
        complete: function( jqXHR, textStatus ){
            console.log(' [testView]AJAX complete ', jqXHR);
        }
    });
}
```

```
initializeFileExplorer()
```

```
function <portlet:namespace>initializeFileExplorer(){
    if( $.isEmptyObject(<portlet:namespace>initData) || (
        <portlet:namespace>initData.type() !== OSP.Enumeration.PathType.FILE &&
        <portlet:namespace>initData.type() !== OSP.Enumeration.PathType.FOLDER &&
        <portlet:namespace>initData.type() !== OSP.Enumeration.PathType.EXT ) )      return;

    var eventData = {
        portletId: '<%=portletDisplay.getId()%>',
        targetPortlet: <portlet:namespace>fileExplorerId,
        data: OSP.Util.toJSON(<portlet:namespace>initData)
    };

    Liferay.fire( 'OSP_LOAD_DATA', eventData );
}
```

## DownloadCurrentFile()

```
function <portlet:namespace>downloadCurrentFile(){
    console.log("[testView] Download current data", <portlet:namespace>currentData);
    if( $.isEmptyObject(<portlet:namespace>currentData) || 
        <portlet:namespace>currentData.type() !== OS
P.Enumeration.PathType.FILE )
    return;

    var filePath = <portlet:namespace>currentData;
    var data = {
        <portlet:namespace>command: "DOWNLOAD_FILE",
        <portlet:namespace>pathType: filePath.type_,
        <portlet:namespace>repositoryType: filePath.re
positoryType_,
        <portlet:namespace>parentPath: filePath.paren
t_,
        <portlet:namespace>fileName: filePath.name_
    };

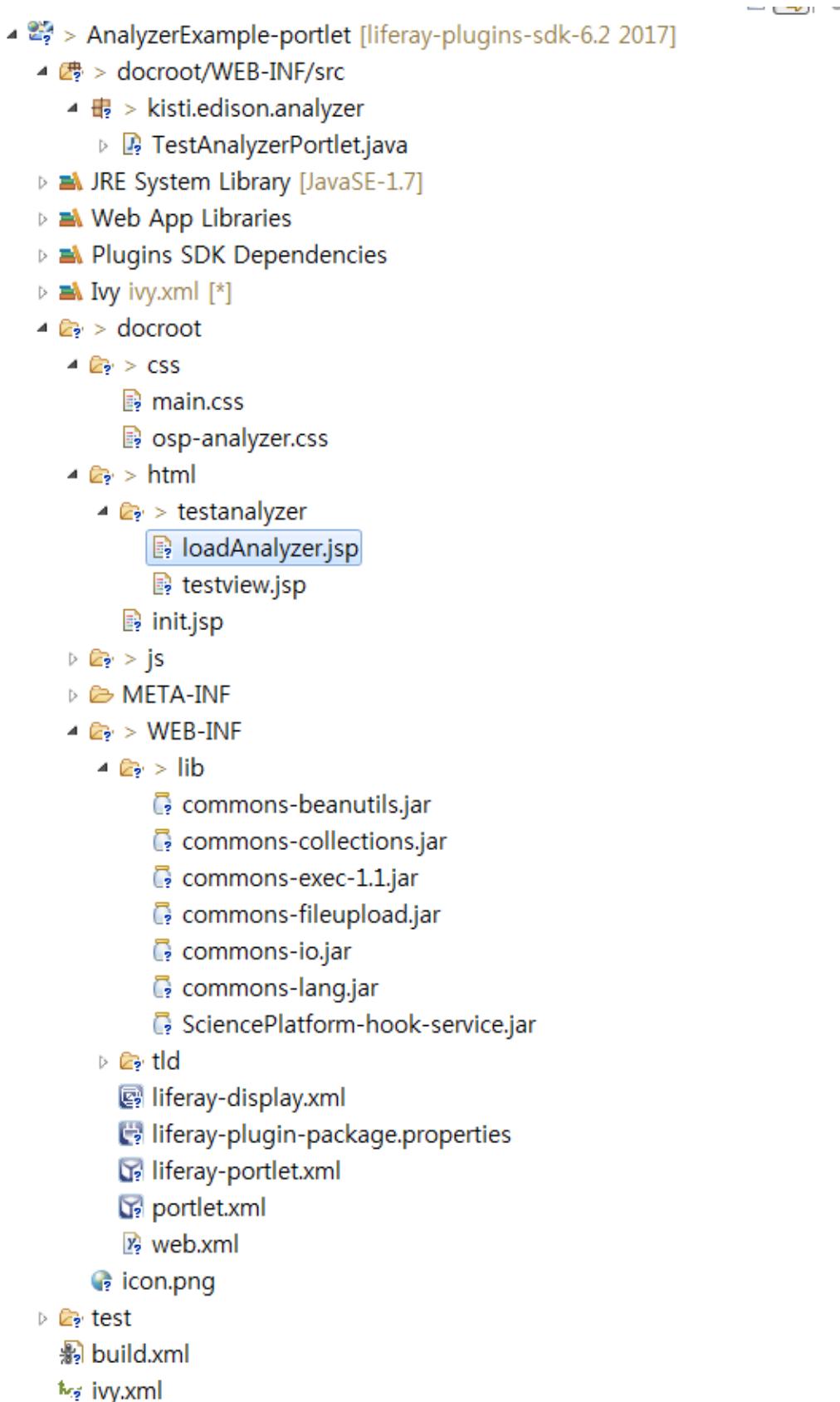
    var base = '<%=serveResourceURL.toString()%>';
    var sep = (base.indexOf('?') > -1) ? '=' : '?';
    var url = base + sep + $.param(data);

    location.href = url;
    <portlet:namespace>loadJSMolFile( <portlet:namespac
e>currentData );

}
```

## 최종 파일 구조

지금까지 기본 설정을 마친 후 최종적인 파일 구조는 다음 그림과 같습니다.





# HANDSHAKE

**Summary:** 초기 워크벤치를 실행하면서 필요한 편집기와 분석기 모듈들을 호출합니다. 해당 편집기와 분석기는 실행되기 전에 HANDSHAKE 이벤트를 워크벤치로부터 받아 현재 상태 및 아이디를 등록을 하게 됩니다.

## OSP HANDSHAKE 이벤트

초기 워크벤치를 실행하면서 필요한 편집기와 분석기 모듈들을 호출합니다. 해당 편집기와 분석기는 실행되기 전에 HANDSHAKE 이벤트를 워크벤치로부터 받아 현재 상태 및 아이디를 등록을 하게 됩니다.

해당 HANDSHAKE 등록 과정에서 편집기 또는 분석기 모듈은 워크벤치의 아이디를 등록하게 되며, 앞으로 발생되는 이벤트에 대해 이벤트를 보내고자 하는 목적지의 주소와 같은 역할을 하게 됩니다.

- 샘플 예제

```
Liferay.on(
    OSP.Event.OSP_HANDSHAKE,
    function(e){
        var myId = '<%=portletDisplay.getI
d()%>';
        if( e.targetPortlet !== myId ){
            return
        }
        console.log(' [NGLViewer]OSP_HANDSHAKE:
['+e.portletId+', '+new Date()+']');
        <portlet:namespace/>connector = e.portle
tId;

        if( e.mode )
            <portlet:namespace/>action = e.m
ode;
        else
            <portlet:namespace/>action = 'VI
EW';
        var events = [
            OSP.Event.OSP_EVENTS_REGISTERED,
            OSP.Event.OSP_LOAD_DATA
        ];
        var eventData = {
            portletId: myId,
            targetPortlet: <portlet:namespac
e/>connector,
            data: events
        };
        Liferay.fire( OSP.Event.OSP_REGISTER_EVENTS, event
Data );
    }
);
```

# REGISTERED EVENT

## OSP REGISTERED 이벤트

- 샘플 예제

```
Liferay.on(
    OSP.Event.OSP_EVENTS_REGISTERED,
    function(e){
        var myId = '<%=portletDisplay.getId()%>';
        if(e.targetPortlet === myId){
            console.log(' [NGLViewer] Registered '+e.portletId+
            ' activated. '+new Date());
        }
    }
);
```

# REQUEST DATA EVENT

## OSP REQUEST DATA 이벤트

사용자가 서버에 저장된 파일을 선택하거나 새로운 데이터를 로드하고자 할 때, 분석기나 편집기에서는 새로운 데이터를 요청해야 합니다. 따라서 분석기나 편집기에서는 새로운 데이터를 요청하는 이벤트를 발생시키고, 해당 이벤트를 받아 새로운 데이터를 받아오게 됩니다.

- 샘플 예제

```
Liferay.on(
    OSP.Event.OSP_REQUEST_DATA,
    function(e){
        var myId = '<%=portletDisplay.getId()%>';
        if( e.targetPortlet === myId ){
            var iframe = document.getElementById('<portlet:namespace/>TBox');
            var eventData = {
                portletId: myId,
                targetPortlet: e.portletId,
                data: {
                    type_: OSP.Enumeration.PathType.FILE_CONTENT,
                    context_: iframe.contentWindow
                },
                params: e.params
            }
            Liferay.fire(OSP.Event.OSP_RESPONSE_DATA,
            eventData);
        }
    }
);
```

# RESPONSE DATA EVENT

OSP RESPONSE DATA 이벤트

- 샘플 예제

```
Liferay.on(
    OSP.Event.OSP_RESPONSE_DATA,
    function( e ){
        var myId = '<%=portletDisplay.getId()%>';
        if( e.targetPortlet !== myId )           return;
        console.log("[ATOM EDITOR] OSP OSP_RESPONSE_DA
TA :", e );
        var iframe = document.getElementById('<portlet
namespace>TBox');
        var filePath = new OSP.InputData( e.data );
        if( filePath.type() !== OSP.Enumeration.PathTy
pe.FILE ){
            alert('File Name is not available. Choose
another one.');
            return;
        }

        if( !<portlet:namespace>saveAction ){
            <portlet:namespace>loadText( filePath, tr
ue );
            $<portlet:namespace>fileExplorerDialogSec
tion.dialog('close');
        }
        else{
            $('#<portlet:namespace>uploadFileName').v
al(filePath.name());
            filePath.type( OSP.Enumeration.PathType.FI
LE_CONTENT );
            filePath.context( iframe.contentWindow.get
Parameters() );

            $.ajax({
                url: '<%=serveResourceURL.toStrin
g()%>',
                type: 'POST',
                dataType: 'json',
                data:{
                    <portlet:namespace>command: "CHEC
K_DUPLICATED",
                    <portlet:namespace>repositoryTyp
e: filePath.repositoryType(),
                    <portlet:namespace>parentPath: fi
lePath.parent(),
                    <portlet:namespace>fileName: file
Path.name()
                }
            });
        }
    }
);
```

```
        },
        success: function(result){
            var duplicated = result.duplicate
d;
            if( duplicated ){
                var confirmDialog = $('#<portlet
et:namespace/>confirmDialog');
                confirmDialog.dialog(
                    {
                        resizable: false,
                        height: "auto",
                        width: 400,
                        modal: true,
                        buttons: {
                            'OK': func
tion() {

                $( this ).dialog( 'destroy' );

                <portlet:namespace>saveContentAs ( filePath );

                $<portlet:namespace/>fileExplorerDialogSection.dialog('close');
                    },
                    Cancel: fu
nction() {

                $( this ).dialog( 'destroy' );
                    }
                }
            );
        }
        else{
            <portlet:namespace/>saveConten
tAs ( filePath );
            $<portlet:namespace/>fileExplo
rerDialogSection.dialog('close');
        }
    },
    error: function( e, d ){
        console.log(' [ATOM EDITOR] file s
ave error', e, d);
    }
});
```

```
    }  
};
```

# LOAD DATA EVENT

## OSP LOAD DATA 이벤트

편집기 또는 분석기에서 데이터를 로드할 때 사용되는 이벤트로, 서버에 저장되어 있는 파일을 로드하거나 시뮬레이션 결과 데이터를 로드할 때 쓰이는 이벤트입니다.

사용자가 서버쪽 파일을 선택하거나 시뮬레이션이 성공적으로 종료되는 상황에서 워크 벤치에서 이벤트를 발생시키며, 이벤트 데이터 내 파일 경로에 따라 파일을 읽어 들이는 프로세스를 거쳐 사용자에게 보여줍니다.

- 샘플 예제

```
Liferay.on(
    'OSP_LOAD_DATA',
    function(e){
        var myId = '<%=portletDisplay.getId()%>';
        console.log("[ATOM EDITOR] OSP LOAD DATA :", myId, e.targetPortlet );
        if( e.targetPortlet !== myId )           return;

        <portlet:namespace/>initData = e.data;
        if( ! <portlet:namespace/>initData.repositoryType_ ){
            <portlet:namespace/>initData.repositoryType_ = 'USER_HOME';
        }
        <portlet:namespace/>loadEPDEditor( OSP.Util.toJSON(
            <portlet:namespace/>initData ) );
        <portlet:namespace/>initializeFileExplorer();
    }
);
```

# INITIAIZE EVENT

## OSP INITIAIZE 이벤트

워크벤치가 실행된 이후, 초기화 값을 설정하는 이벤트입니다. 워크벤치가 실행되고 시뮬레이션 실행 환경이 구성되고 난 뒤, 각 편집기나 분석기에서는 필요한 값들의 초기화를 진행하는데 이 때 사용되는 이벤트입니다.

- 샘플 예제

```
Liferay.on(  
    'OSP_INITIALIZE',  
    function(e){  
        var myId = '<%=portletDisplay.getId()%>';  
        if( e.targetPortlet !== myId )      return;  
        <portlet:namespace>initializeFileExplorer();  
    }  
);
```

# REQUEST OUTPUT VIEW EVENT

OSP REQUEST OUTPUT VIEW 이벤트

- 샘플 예제

```
Liferay.on(
    OSP.Event.OSP_REFRESH_OUTPUT_VIEW,
    function(e){
        var myId = '<%=portletDisplay.getId()%>';
        if( myId !== e.targetPortlet ) return;

        var eventData = {
            portletId: '<%=portletDisplay.getI
d()%>',
            targetPortlet: <portlet:namespace>con
nector
        };

        Liferay.fire(OSP.Event.OSP_REQUEST_OUTPUT_PAT
H, eventData);
    }
);
```

# REQUEST DATA CHANGED EVENT

## OSP DATA CHANGED 이벤트

해당하는 포틀릿이 편집기인 경우, 사용자의 입력을 받게 됩니다. 사용자의 입력을 받아 입력 데이터가 변경되게 되면 해당 데이터의 변경을 워크벤치에 알려야 합니다. 따라서 사용자의 입력에 따라 변경된 데이터를 JSON의 형태로 이벤트를 발생시켜 워크벤치에 전달합니다.

- 샘플 예제

```
function <portlet:namespace>fireTextChangedEvent( data
){
    <portlet:namespace>ViewStructure();

    var inputData = new OSP.InputData();
    inputData.type( OSP.Enumeration.PathType.FILE_CONT
ENT );
    if( $.isEmptyObject(<portlet:namespace>initData)
{
        inputData.repositoryType( '<%=OSPRestoreType
s.USER_HOME.toString()%>' );
    }
    else if( <portlet:namespace>initData.repositoryTy
pe_ ){
        console.log("[ATOM EDITOR] test re[psotpry Typ
e]", <portlet:namespace>initData);
        inputData.repositoryType(<portlet:namespace>i
nitData.repositoryType_);
    }
    else{
        inputData.repositoryType( '<%=OSPRestoreType
s.USER_HOME.toString()%>' );
    }
    inputData.context( data );

    var eventData = {
        portletId: '<%=portle
tDisplay.getId()%>',
        targetPortlet: <portl
et:namespace>connector,
        data: OSP.Util.toJSON
N(inputData)
    };
    Liferay.fire( OSP.Event.OSP_DATA_CHANGED, eventData
);
}
```