

XMLPortletFactory User Guide

Preface

www.XMLPortletFactory.org
Jack A. Rider
XMLPortletFactory Getting Started Guide
By Jack A. Rider
Copyright © 2009 - 2011 by Jack A. Rider

This work is offered under the Creative Commons Attribution-Share Alike Unported license. You are free:

- to share—to copy, distribute, and transmit the work
- to remix—to adapt the work

Under the following conditions:



Attribution. You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).



Share Alike. If you alter, transform, or build upon this work, you may distribute the resulting work only under the same, similar or a compatible license.

You may view the full version of this license online here:

<http://creativecommons.org/licenses/by-sa/3.0>

Contributors:

David Segui, Fran García, Moisés Belda.

Table of Contents

1. Preface
2. Introduction
 - a. What is XMLPortletFactory?
 - b. In which versions of Liferay runs?
3. Initial Setup
 - a. Pre-requisites
 - b. Download
 - c. Install XMLPortletFactory in Liferay SDK
 - d. First run.
4. XMLPortletFactory in detail
 - a. Must know before
 - b. Node by node explanation
5. How-to
 - a. Eclipse edit, your generated portlet
 - b. I18n, change or add new languages
 - c. Adding new regexp validations to my portlets.

Introduction

What is XMLPortletFactory?

XMLPortletFactory is a project whose goal is the fast and easy generation of portlets, which give support and maintenance to tables in a database, always under Liferay portal infrastructure.

Experience in the programming world, shows that a big percentage of projects; portlets are fundamentally database table maintenance, just adding, editing and deleting rows in tables. Under that circumstances, we spend a lot of time, coding same routines with different table name and fields, is like Groundhog Day in the office again.

Now XMLPortletFactory can generate full, professional-looking portlets within minutes of starting, just have to define one xml file, which contains the applications definition. With it you can generate portlets for the different Liferay portal versions (6.0.1, 6.0.2.....6.0.6...and so on). If you have de xml definition, your portlet for your version is seconds away. Can you imagine your time saving, in portal version upgrades?

In which versions of Liferay runs?

Actually, is tested and found to be generating working portlets in 6.0.x. Whenever 6.1.x comes out, I will try to modify the actual process to select appropriate major Liferay version 6.0 or 6.1 with the same xml files.

Initial Setup

Pre-requisites

We first need Liferay Plugins SDK (6.0.x) installed and working (java, ant, etc.), the Liferay people made it as easy as unzipping a file and editing a file. See instructions here:

<http://www.liferay.com/documentation/liferay-portal/6.0/development/-/ai/the-plugins-sdk>

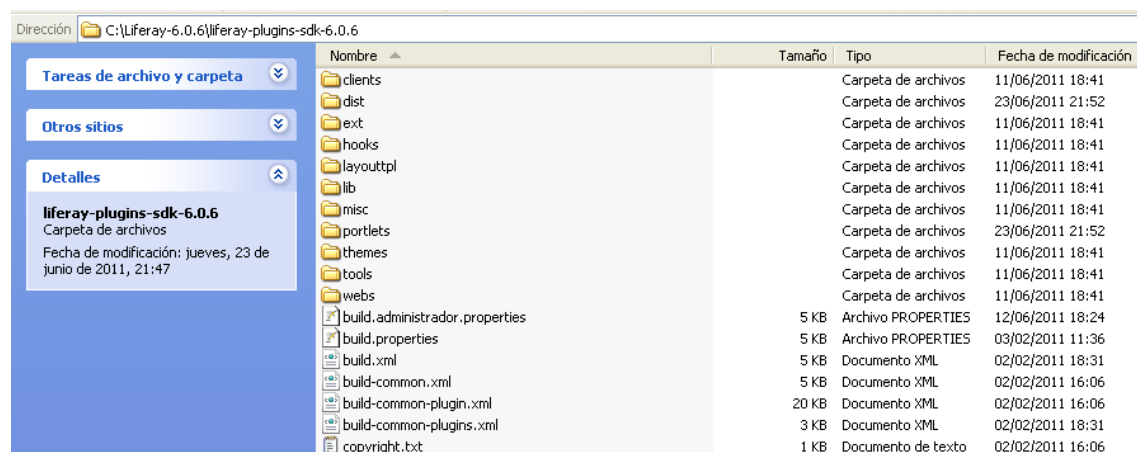
Download

You can download the XMLPortletFactory.zip SDK from here:

<http://sourceforge.net/projects/xmlportlet/files>

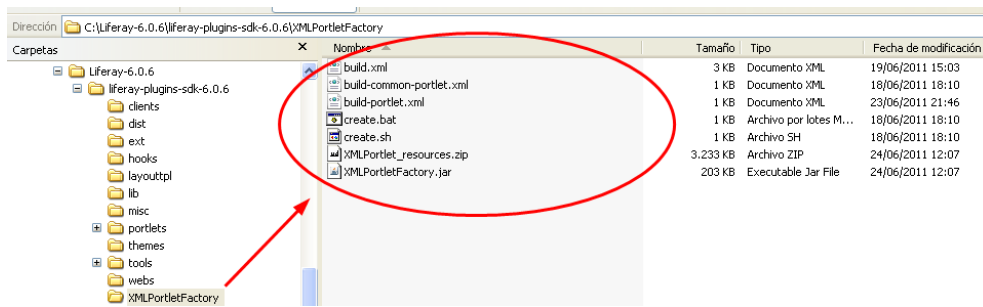
Install XMLPortletFactory in Liferay SDK

If you have a Liferay plugin SDK, installed and working, you should have somewhere a directory that looks like this:



Nombre	Tamaño	Tipo	Fecha de modificación
clients		Carpeta de archivos	11/06/2011 18:41
dist		Carpeta de archivos	23/06/2011 21:52
ext		Carpeta de archivos	11/06/2011 18:41
hooks		Carpeta de archivos	11/06/2011 18:41
layouttpl		Carpeta de archivos	11/06/2011 18:41
lib		Carpeta de archivos	11/06/2011 18:41
misc		Carpeta de archivos	11/06/2011 18:41
portlets		Carpeta de archivos	23/06/2011 21:52
themes		Carpeta de archivos	11/06/2011 18:41
tools		Carpeta de archivos	11/06/2011 18:41
webs		Carpeta de archivos	11/06/2011 18:41
build.administrador.properties	5 KB	Archivo PROPERTIES	12/06/2011 18:24
build.properties	5 KB	Archivo PROPERTIES	03/02/2011 11:36
build.xml	5 KB	Documento XML	02/02/2011 18:31
build-common.xml	5 KB	Documento XML	02/02/2011 16:06
build-common-plugin.xml	20 KB	Documento XML	02/02/2011 16:06
build-common-plugins.xml	3 KB	Documento XML	02/02/2011 18:31
copyright.txt	1 KB	Documento de texto	02/02/2011 16:06

Installing XMLPortletFactory is just unzipping the file “XMLPortletFactory.zip” under XMLPortletFactory directory resulting in this:



That's it!!!, XMLPortletFactory is installed in your Liferay plugins SDK.

First run

We are going to create our first portlet, and for that, we can use the same easy and convenient way, Liferay SDK uses for its own plugins. That is, from inside XMLPortletFactory folder, execute the “create” line command passing a name parameter and a description enclosed with quotes. Like this.

On Windows:

create.bat example “Here is an example”

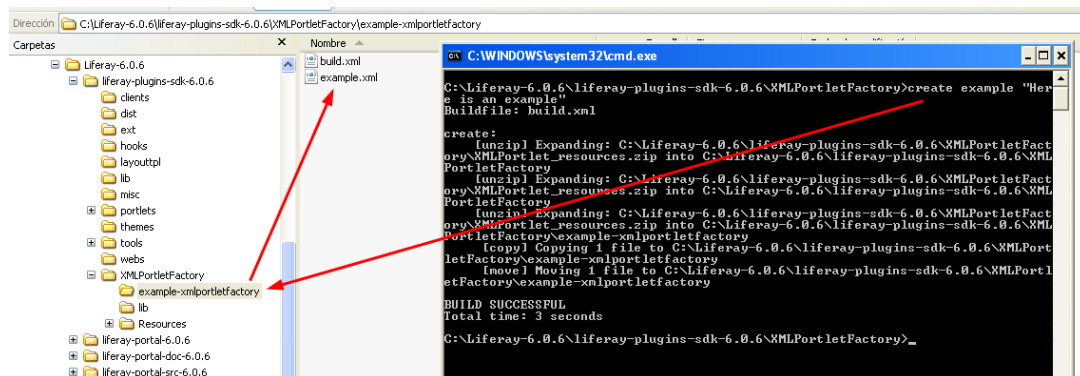
On Linux:

./create.sh example “Here is an example”

Note 1: On Linux, remember to give execute permission to “create.sh”.

Note 2: Portlet name is restricted to [a-z]{1}[a-z][A-Z], meaning that name must start with lower case and no numbers or special characters are allowed in portlet names. (I know, I know, I will work this one)

This results in the creation of a folder, named “example-xmlportletfactory” containing a build file and a small, starting point XMLPortletFactory file definition, named “example.xml”:



This example.xml definition just contains the data for a portlet called “example” that maintains a file with the fields “exampleId” and “exampleDescription”, and is generated here, just to be a starting point. You can edit it to suit your needs following specifications in chapter “***XMLPortletFactory in detail***”.

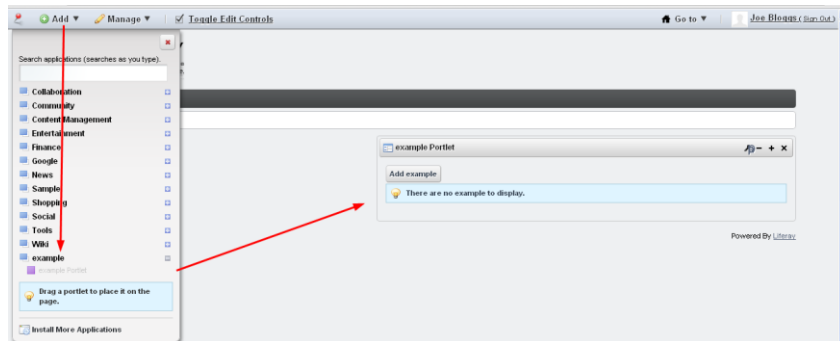
We now have an xml file, that conforms specifications (see ./Resources/xml/xmlportletfactory.xsd), and we want our portlet. Just enter folder example-xmlportletfactory and execute ant command.

cd example-xmlportletfactory
ant

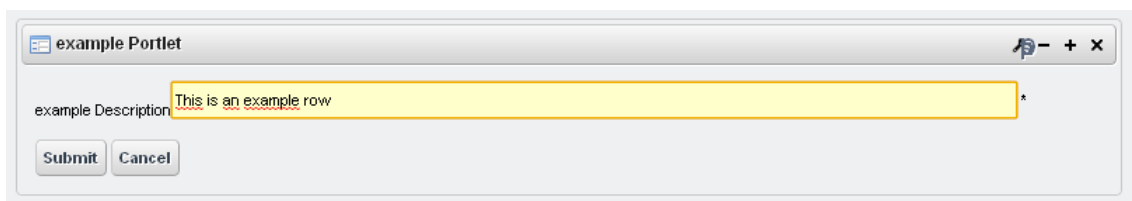
What happened? Many things, let me resume:

- Folder example-portlet was created in SDK's **portlet** folder.
 - Note: be aware that folder name “example-portlet” comes from xml definition node <projectName> and not from xml file name.
- It was populated with all files needed.
- Ant build-service was executed to generate Liferay services, persistence, etc.
- Ant deploy was executed to generate portlet war, that now can be found in SDK's **dist.** folder
- If you had SDK's portal running, now the portlet is deployed, installed and the tables created.

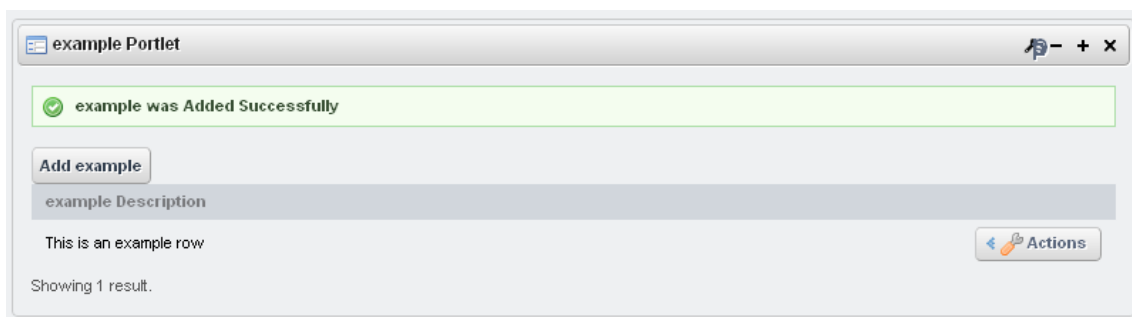
Just go to “Add” → “more” to install applications, and you will find that we have a new portlet in the list:



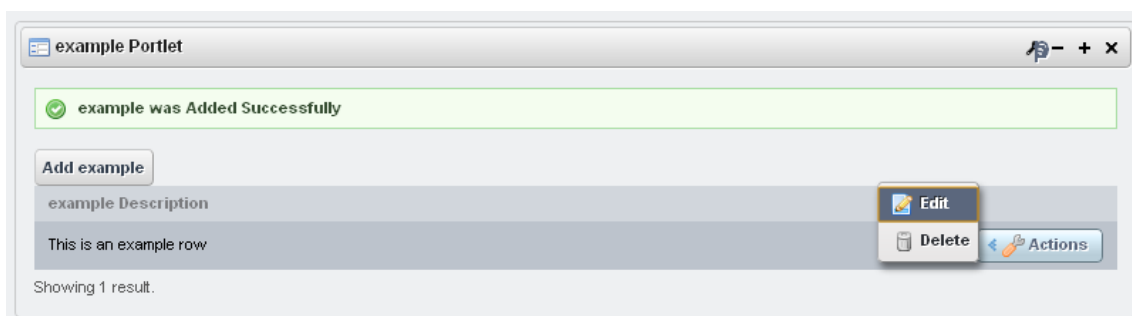
With it you can insert new table rows by pressing “Add example” button:



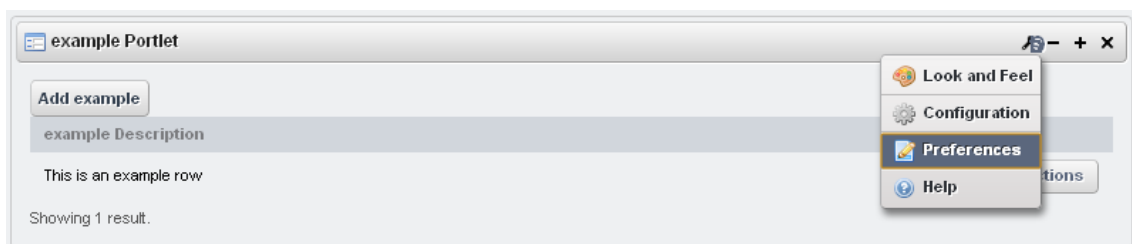
Pressing the “Submit” button:



Pressing the “Actions” button:



Pressing the tools button:



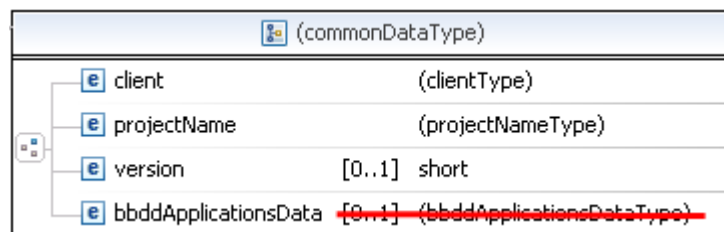
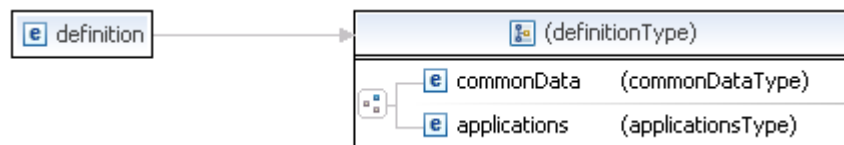
We have now a very simple portlet, but we can do very sophisticated portlets, like header → detail/s, combo validation of other files, regexp validation of fields, filter/search/order rows, change language, etc. Look for this and more in chapter “*XMLPortletFactory in detail*”.

XMLPortletFactory in detail

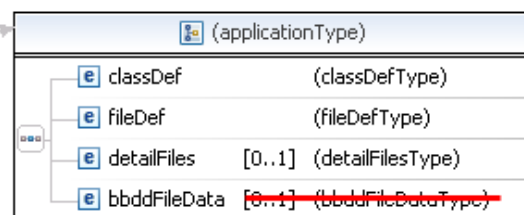
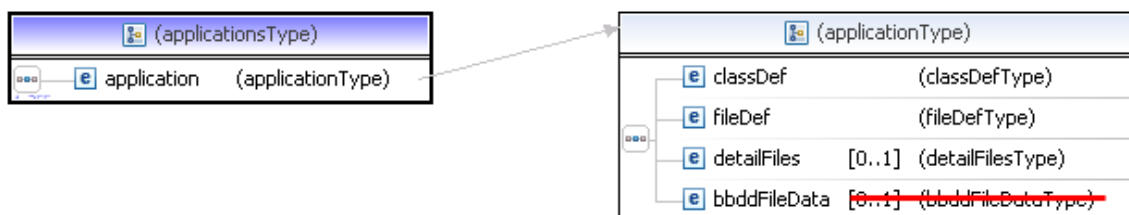
Must know before

Node by node explanation

Xml file definition is supposed to contain all data needed to generate everything in the portlet. The file starts with the tags <definition> that have 2 required children <commonData> and <applications>

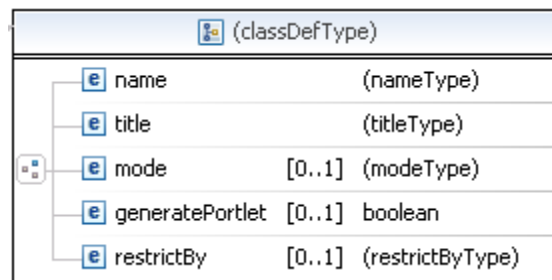


- <commonData>
 - contains 3 more required children
 - groups common data for all applications (portlets to be generated)
 - <client>
 - Is aimed to contain the client name, for this project, as one client can have many projects.
 - [A-Za-z0-9]
 - Required: true.
 - This name will be used (lower case) in generated java package, like:
 - org.xmlportletfactory.portal.<client />
 - <projectName>
 - The project name.
 - [A-Za-z]
 - Required: true.
 - This name will be used for portlet class name and in category for liferay-display.xml
 - In portlet name “<projectName />-portlet
 - <version>
 - Version number of <projectName>
 - Required: false. If omitted defaults to 1
 - Field used in:
 - File “liferay-plugin-package.properties” module-incremental-version variable
 - ~~<bbddApplicationsData>~~ → deprecated / not in use



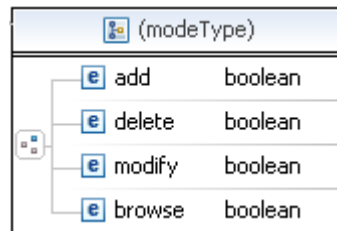
- <applications>
 - Just the container of nodes type <application> 1 to 255 applications in one xml.
 - <application>
 - Each application will correspond to one portlet generated
 - Each application will correspond to one table in the BBDD.

- Contains 3 children nodes, classDef, fileDef and optional detailFiles.

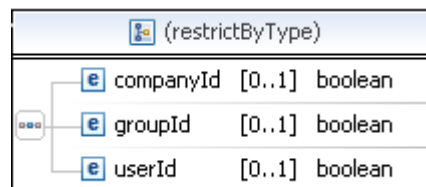


- <classDef>**

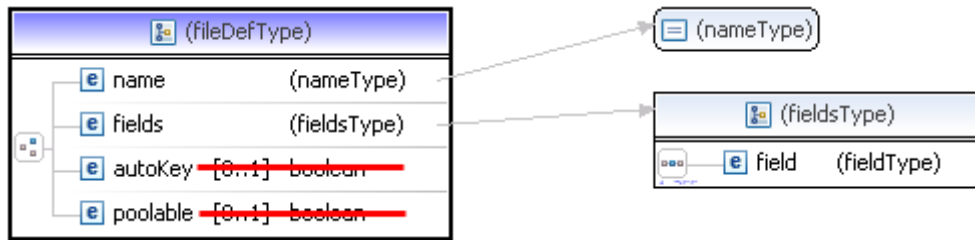
- <name>**
 - Used to reference this portlet everywhere.
 - Also known in velocity files as "file_name" and "class_name"
 - [A-Za-z0-9]
 - Required: true.
- <title>**
 - Actually only used for documentation and readability.
 - Required: true.



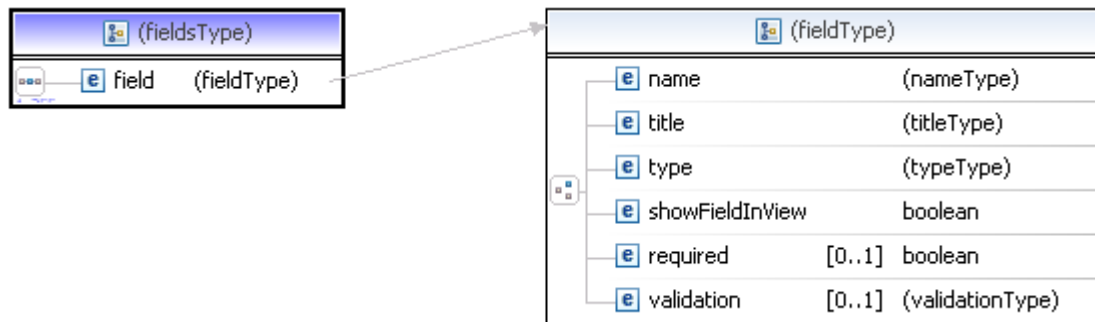
- <mode>**
 - Defines if the portlet can or can't insert, delete, modify, or browse records of de bbdd.
 - All options true by default.
 - Required: false.
- <generatePortlet>**
 - True or false, to define if a portlet is to be generated. If false, it will only generate classes to access BBDD table, and include them in the .jar.



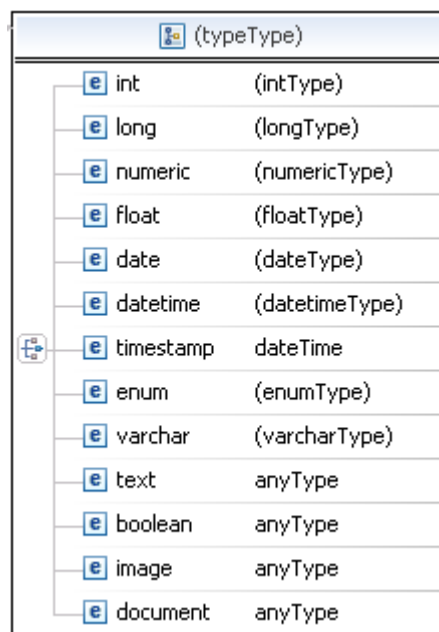
- <restrictedBy>**
 - Very usefull feature that restrict access to BBDD depending or the companyId, groupId or userId of the user that adds a row.
 - If is companyId restricted, all users with the same companyId, will see all rows inserted from that company.
 - If is groupId restricted, only users with the same groupId, will see rows inserted by them.
 - Finally if userId restricted, each user will only see rows inserted by themselves.
 - This feature can be achieved because, XMLPortletFactory always creates DDBB tables including companyId, groupId and userId fields, and silently populates the current user values at insertion time.



- <fileDef>
 - Contains all data referred to each field of the table.
 - <name>
 - The name of the table in the bdd. Be aware that table name will be prefixed with the project name.
 - [a-z]{1}[a-zA-Z0-9]*
 - Required: true.



- <fields>
 - 1 to 255 <field> nodes.
 - IMPORTANT: First <field> is always the primary key of the table.
 - <field>
 - <name>
 - Name of the field in the table.
 - [a-zA-Z0-9]*
 - Required: true.
 - <title>
 - String used in jsp to ask for field value in edition mode or column name in search containers for view mode.
 - Also populated to language_xx.properties files for user modification in I18n.



- <type>
 - Here you select which type of field is going to be from the ones showed in the image above.
 - Int, long, numeric, float and varchar, admit <regexp> function. That means that you can control the data values admitted.
 - <regexp> values are defined in file "regexp.properties" and actually includes, email, ip, url, zipspain, nifspain, phonespain, etc.
 - Int, long, numeric, float, date, datetime and varchar types admit <order> function. That means that the columns with order value true, in search container of the portlet view mode, will be orderable with a click.
 - <order> defaults to false.
 - Int, long, numeric, float and varchar types admit the <lenght> clause. That means that you control de length of the field.
 - Varchar type, admits a special clause <filter>, that when is true, automatically generates a search/filter field in the portlet view mode, that permits filtering of rows field values, with this clause.
 - <type> is probably the most complex node of all, see examples provided to get a more exact point of view.
 - <text> creates a text box.
 - <boolean> creates a radio button
 - <image> creates a complete infrastructure to upload images, which internally are stored in Liferay's Image Gallery.
 - <document> creates a complete infrastructure to upload documents that internally are stored in Liferay's Document Library.
 - Incredible but true.
- <showFieldInView>
 - True or false, depending if we want the field value to be shown in portlet view mode, as a search container column.
 - If we have many fields in the table, we usually don't want to show all of them as columns, in view mode, because of width overflow. So we just put as true the most important ones.
 - At least one.
- <required>
 - Used to make sure, a field value is obtained when inserting a row.
 - Fields required, are marked with an asterisk in portlet edit mode.
 - It defaults to false

(validationType)		
e	className	Name
e	fieldName	Name
e	orderByField	Name

- <validation>
 - Used to validate input against other BBDD table row.
 - It generates a combo, with values from other table.
 - <className>
 - Existing <className> from other <application> of this xml.
 - <fieldName>
 - Existing <fieldName> of <application> referenced by previous <className>.
 - This field is the one used to match, for validation, so it should be of the same type. For example, if we are asking for the telephoneType of a client, we

would use the `<fieldName>` "telephoneTypeeld" from the telephoneTypes class.

- `<orderByField>`
 - Is the descriptive field of the validation application, to show in the combo box.
 - Following the previous example, would be the "telephoneTypeDescription" field.
- `<autoKey>` → deprecated / not in use
- `<poolable>` → deprecated / not in use



- `<detailFiles>`
 - You can define a list up to 255 detailFiles, for each application.
 - `<detailFile>`
 - Is an application reference whose records/rows depend on the ones defined in the current application, which can be zero to many. For example, telephone numbers of a company, or lines in an invoice, etc.
 - You can have many detail files to one header, and even, detail files of detail files. But for all this to work, portlets have to be installed in the same page because, their view is refreshed via portlet intercommunication processes. Be aware of this.
 - `<detailFileName>`
 - Just the class name of the application that should exist within this xml.
 - `<connectionFieldName>`
 - The field name used to connect the header to the detail application.
 - IMPORTANT: actually, this should be field #2 of the detail file.
 - In an invoice header → invoice lines example, would be
 - Invoice Header fields:
 - InvoiceID
 - InvoiceDate
 - Invoice..etc.
 - Invoice Lines fields:
 - InvoiceLineId
 - InvoiceId ← field #2 of Invoice Lines.
 - InvoiceLineArticleId
 - InvoiceLine....Etc
 - `<connectionTitle>`
 - Just to describe the detailFile relationship, could be something like "Invoice Lines"
 - Required: false.
 - `<bbddFileData>` → deprecated / not in use.

How-to

Here will relate different possibilities of how to make XMLPortletFactory your favourite tool to generate portlets. Actually most of the generation is based in velocity templates that are very easy to edit.

How to Eclipse edit, your generated portlet

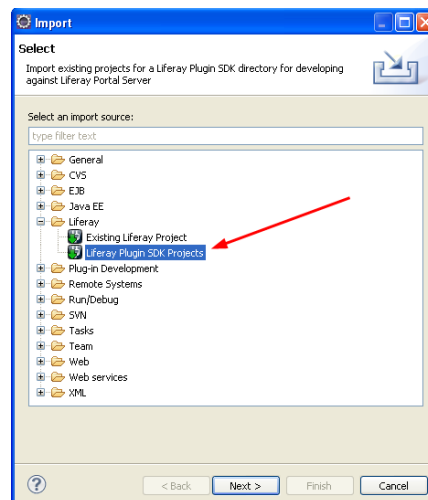
So you are a portlet developer and want to twist the generated portlet to meet your needs, well that is going to be ease, first remember to have prerequisites controlled:

Prerequisites

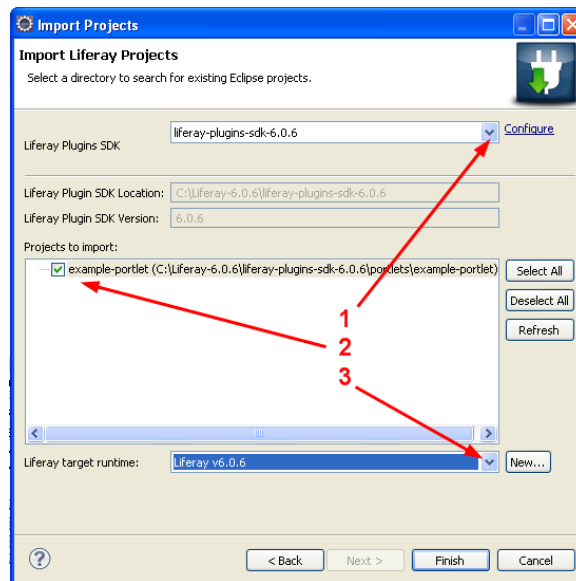
- Liferay plugins SDK installed and working
 - <http://www.liferay.com/documentation/liferay-portal/6.0/development/-/ai/the-plugins-sdk>
- XMLPortletFactory installed and working in Liferay plugins SDK
 - <http://sourceforge.net/projects/xmlportlet/files/>
 - A portlet already generated, (see “First Run” chapter)
 - We are going to use same example here (the “example-portlet”)
- Eclipse Java EE IDE for web developers, installed and working.
 - <http://www.eclipse.org/>
 - Tested on version Helios Service Release 2.
 - Liferay IDE tools 1.2.3v2011 installed.
 - <http://www.liferay.com/documentation/liferay-portal/6.0/development/-/ai/liferay-i-2>

Steps

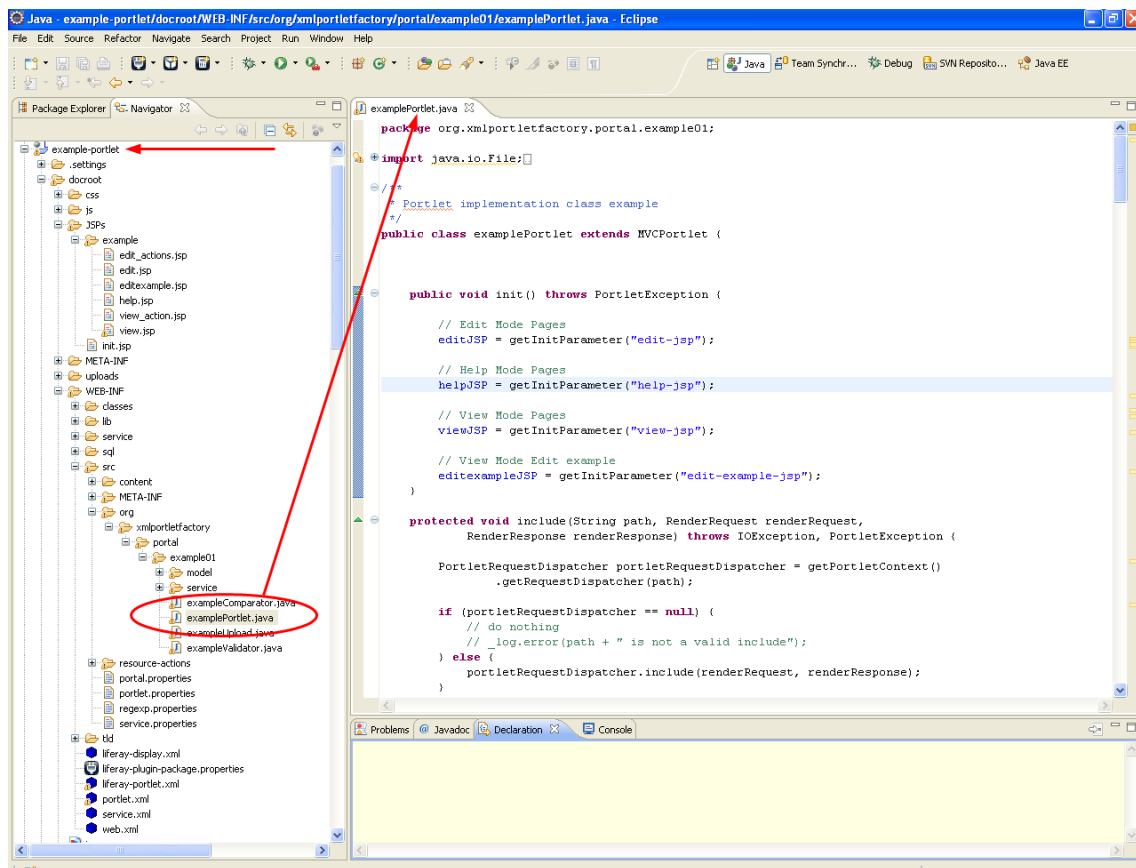
1. Start eclipse
2. On the Navigator or Package Explorer right button → import → Liferay Plugin SDK Projects



3. Next Button, select your Plugins SDK, the example-portlet, and your target runtime



4. That's it, you are ready to go, you have the full portlet with sources at your disposal in your favourite IDE. It should look something like this:



Have fun.

More information about importing existing projects into Liferay IDE here:

<http://www.liferay.com/documentation/liferay-portal/6.0/development/-/ai/importing-existing-projects-into-liferay-ide>

How to I18n, change or add new languages

So you already generated a few portlets and need to add new language templates or change something in the actual ones. As you already notice portlet generation contemplates “en”, and “es” languages, if you develop a template for other language, feel free to send a copy to me, I will include it in the next version.

Prerequisites

- Liferay plugins SDK installed and working
 - <http://www.liferay.com/documentation/liferay-portal/6.0/development/-/ai/the-plugins-sdk>
- XMLPortletFactory installed and working in Liferay plugins SDK
 - <http://sourceforge.net/projects/xmlportlet/files/>
 - A portlet already generated, (see “First Run” chapter)
 - We are going to use same example here (the “example-portlet”)

Steps

In the first run of XMLPortletFactory, two folders were automatically created, that were, lib and Resources. Under resources, you will find many files used in the generation of the portlet. We are going to talk about the ones that control I18n.

1. Go to XMLPortletFactory / Resources / Velocity Templates / PortletFiles folder.
2. If you just want to change something already exists just edit corresponding “Portlet_Language_??_properties.vm” file to suit your needs.
3. In case you want to create a new language template copy “Portlet_Language_properties.vm” to “Portlet_Language_<your lang here>_properties.vm”
4. Change line 4 to meet your lang so corresponding new file will be generated
 - a. #set(\$createName = "/Language_<your lang here>.properties")
 - b. Change rest of lines ALWAYS at the right side of the “=” sign.
 - c. Don’t touch velocity variables enclosed like this: \${xxxxxxx} they are values substituted at generation time.
 - d. Don’t touch lines starting with the “#” sign; they are velocity commands or file comments.
 - e. If you have doubts compare English and Spanish files, it clears a lot to compare them.
5. Now generate again
 - a. Cd to your “portlet”-xmlportletfactory
 - b. Ant

Results

You now are supposed to have a new language_xx.properties file in your newly generated portlet, and in every portlet you generate from this XMLPortletFactory SDK.

Remember to backup a copy of the file.

Have fun.

How to add new regexp validations to my portlets.

As you already noticed, regexp validation can be made to almost all field types with ease, but I see you want more possibilities. You can achieve this in very few steps.

Prerequisites

- Liferay plugins SDK installed and working
 - <http://www.liferay.com/documentation/liferay-portal/6.0/development/-/ai/the-plugins-sdk>
- XMLPortletFactory installed and working in Liferay plugins SDK
 - <http://sourceforge.net/projects/xmlportlet/files/>
 - A portlet already generated, (see “First Run” chapter)
 - We are going to use same example here (the “example-portlet”)
- Regexp knowledge
 - http://en.wikipedia.org/wiki/Regular_expression

Steps

In the first run of XMLPortletFactory, two folders were automatically created, that were, lib and Resources. Under resources, you will find many files used in the generation of the portlet. We are going to talk about the ones that control regexp validations.

1. Go to XMLPortletFactory / Resources / PortletStructureAndFiles / docroot / WEB-INF / src folder.
2. Edit regexp.properties file
 - #
 - # XMLPortletFactory regular expression validation types
 - #
 - float_regexp=[-+]?[0-9]*\.[0-9]+\$
 - integer_regexp=[-+]?[0-9]+[0-9]*\$
 - email=[A-Z0-9._%+-]+@[A-Z0-9.-]+\.[A-Z]{2,4}\$
 - email_error=email_error
 - ip=(25[0-5]|2[0-4][0-9]|01?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|01?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|01?[0-9][0-9]?)\.(25[0-5]|2[0-4][0-9]|01?[0-9][0-9]?)\$
 - url=(http|ftp(s?)):\.\/[0-9a-zA-Z]([-\/w]*[0-9a-zA-Z])*(:(0-9)*(\.\/?)([a-zA-Z0-9-\.\/\|]+\⇕?))\$
 - url_error=url_error
 - zipspain=[0-9]{2}[0-9][1-9][1-9][0-9][0-9]{3}\$
 - zipspain_error=zipspain_error
 - nifspain=[0-9]{8}[A-Za-z]
 - nifspain_error=nifspain_error
 - phonespain=[0-9]{2,3}-?[0-9]{6,7}\$
 - phonespain_error=phonespain_error
3. change or add whatever new regexp expression to the file, just considering the following 2 rules
 - a. Add a “_error” property for each new regexp expression.
 - i. That is if you put a upper case regexp expression named “uppercase=[A-Z]*” you must create “uppercase_error=uppercase_error”.
 - b. Add your “???_error” property to your I18n files
 - i. Something like this: “uppercase_error=Must be upper case letters.”
 - ii. See “How to I18n, change or add new languages” for help.
 - iii. This way you will receive errors messages whenever you introduce wrong data.

Results

You now are supposed to have a new regexp expression that can be used in your xml files, and will be considered in future portlet generations. Remember to backup a copy of the file.

Have fun.