How to build a plugin for Servoy – A step by step tutorial brought to you by Servoy Stuff

A. Introduction

In this step by step tutorial, we will build a very simple but operational Servoy plugin. Our requirements will be to build a component which will query a whois server and retrieve whois information about a domain. Simple enough, but it will allow us to see in details all the basic principles of building a plugin for Servoy.

This will give us an overview of all the steps required to build, write and deploy a plugin in Servoy. In a future tutorial I will probably get into the problem of jar dependencies, and how to solve it but our plugin today will be self-contained so a JNLP file will not be required.

We will see in details how to use Eclipse to build a plugin, and even if this tutorial resource is not about how to use Eclipse (there are plenty of resources about that on the internet, you could start by watching the Eclipse tutorial videos there: http://eclipsetutorial.sourceforge.net/workbench.html , or just Google "Eclipse tutorials" and you will find some interesting resources – and finally you can use the build-in Help system). Anyway, I will try to give as much information as reasonable when it is relevant.

Note: This tutorial was written using the latest (of today) Servoy Developer 4.1.3, java 1.6.0_13-b03 on Windows XP. The screen captures you will see thus reflect this environment, it might be slightly different depending on yours, but in any case you should be able to achieve the same tasks.

B. Using Eclipse for Java projects

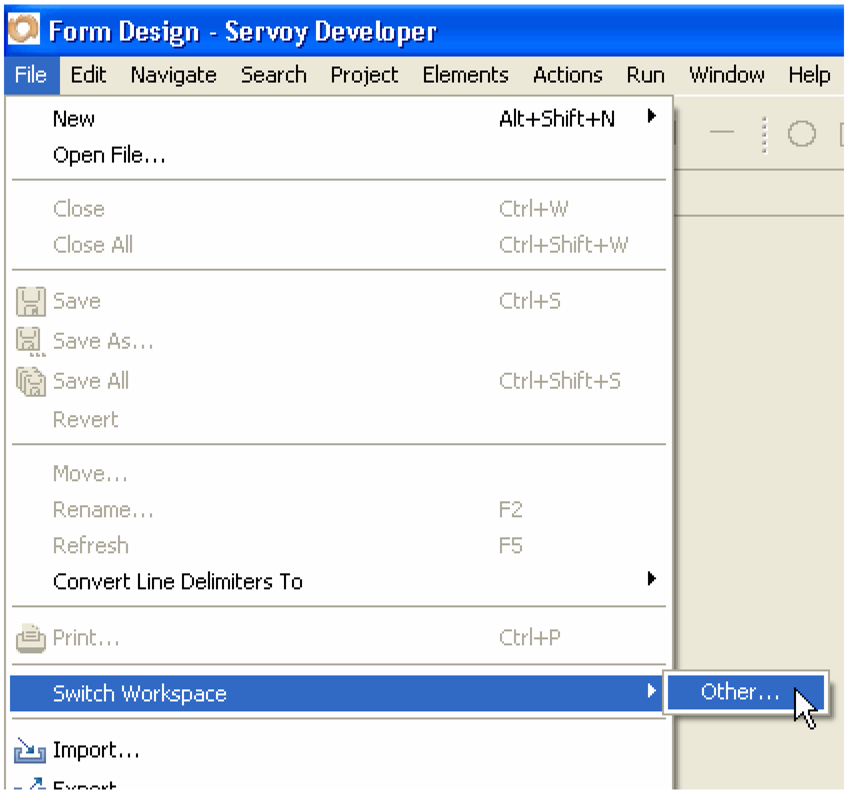
So, to build this plugin we will use Eclipse and since I don't know what distribution of Eclipse you have installed on your computer (there are plenty), we will very naturally use Servoy itself as our Eclipse Java development environment.

Yes! Of course you can use Servoy to build Java projects; in fact you can use Servoy for a lot more than building Servoy solutions! That's what an IDE like Eclipse is for: allowing you to work with the same set of tools for a lot of different projects. And Servoy, being build on Eclipse (version 3.3 precisely - nicknamed "Europa" - for those who like to know all the gritty details), is no exception.

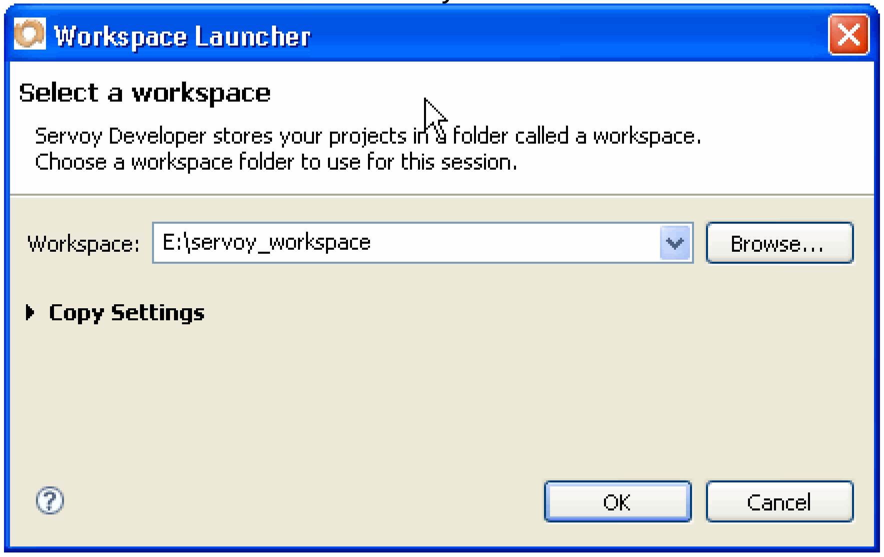
But I know that some of you don't like to "mess with Servoy" for fear of corrupting your solutions, workspace and/or repository. Fear not!

1. You can run another instance of Servoy developer to do your java work (refer to http://www.servoy.com/generic.jsp?mt=396&taxonomy_id=626#qd2 and the "Servoy Server Administrator's Guide" section about "Running multiple instances of Servoy Developer"

2. You can use the same Servoy Eclipse you use for Servoy development, but switch your workspace to another directory for java work, go to "File > Switch Workspace > Other..."



Then choose another directory location:



Select a workspace

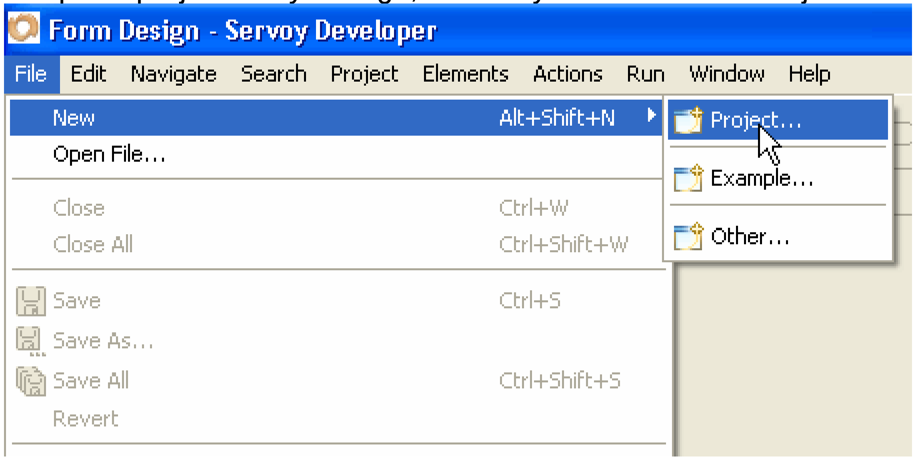
Servoy Developer stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.

Workspace: E:\servoy_workspace

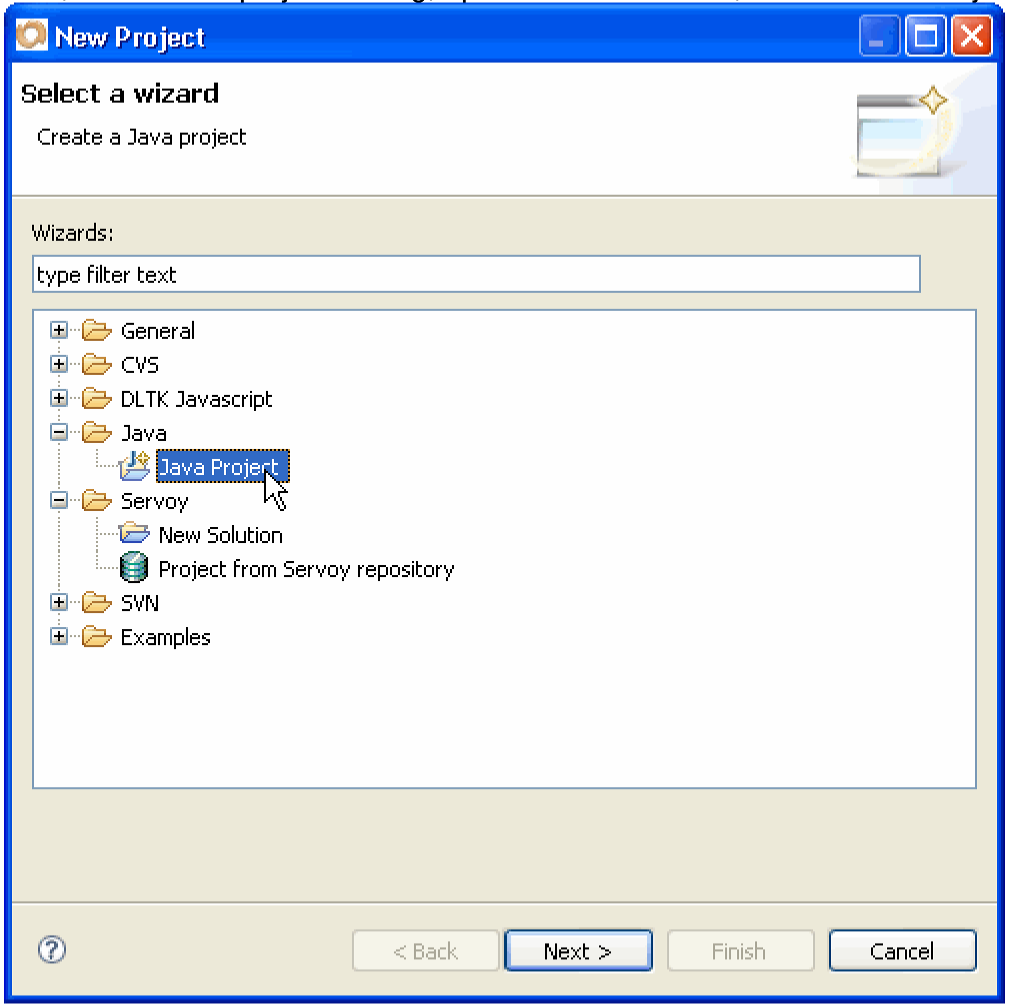
3. Last option: you can do like me and live dangerously☺! (In fact I never ran into any problem – true, I put all my projects under SVN, but I never had to use this “safety net” – not for that anyway)

C. Creating the Java project

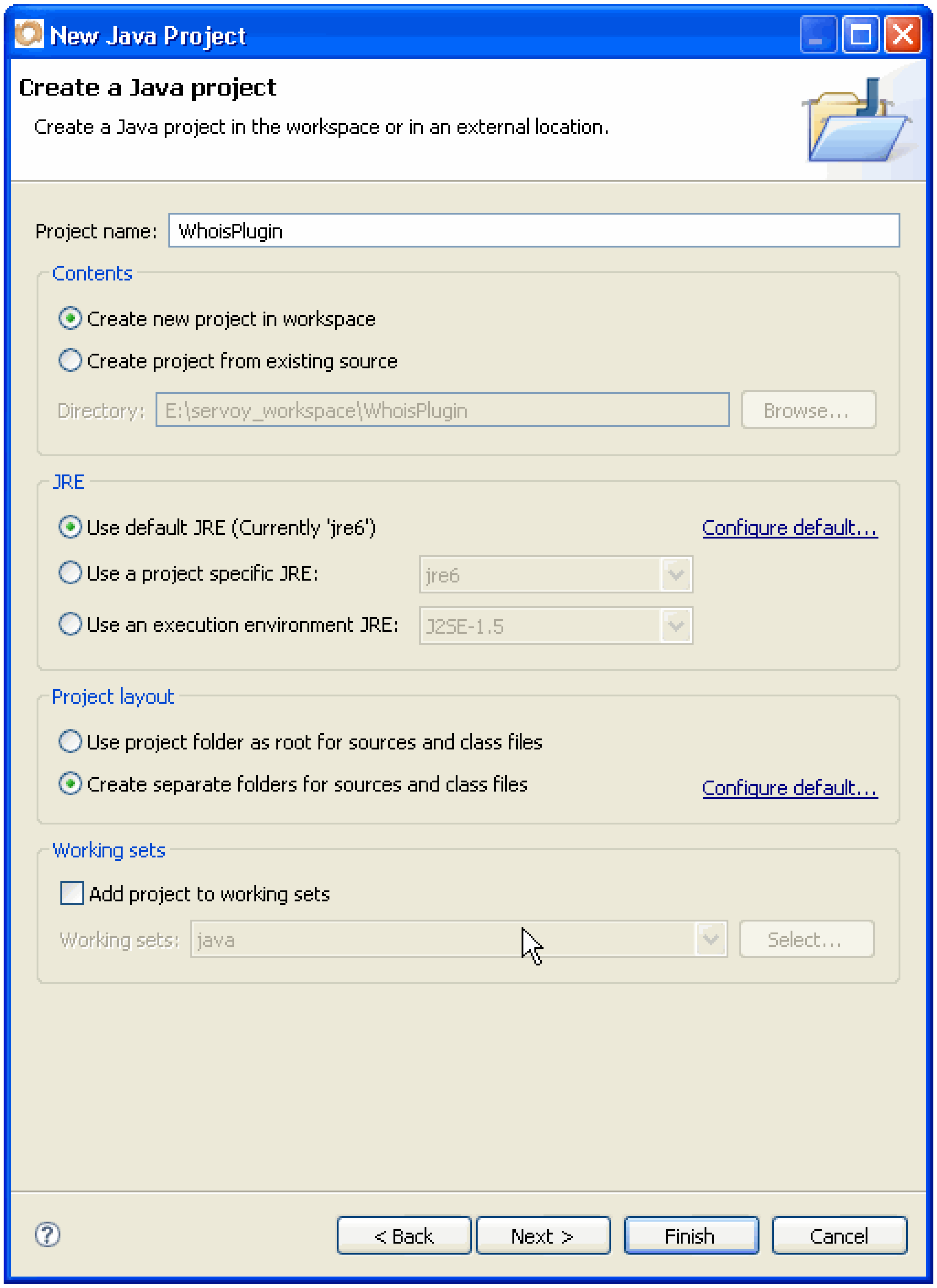
So, now that we have Servoy opened (whichever way you chose to work), the first thing to do is to setup our project. Easy enough, in Servoy: “File > New > Project...”



Then, in the “New project” dialog, open the “Java” folder, choose “Java Project” and click “Next >”



In the next window, type the name of your project (it can be anything you want, but of course it's best if it reflects what you are going to do, and it should be unique in your workspace), so let's call it "WhoisPlugin":



Create a Java project

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

Project name: WhoisPlugin

Contents

(●) Create new project in workspace
( ) Create project from existing source

Directory: E:\servoy_workspace\WhoisPlugin

JRE

(●) Use default JRE (Currently 'jre6')
( ) Use a project specific JRE: jre6
( ) Use an execution environment JRE: J2SE-1.5

Project layout

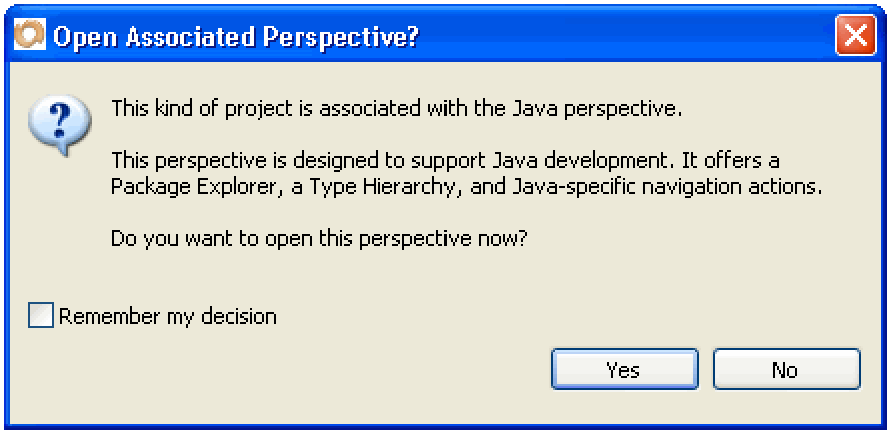
( ) Use project folder as root for sources and class files
(●) Create separate folders for sources and class files

Working sets

[ ] Add project to working sets

Working sets: java

Leave everything to default, - that will do for now – and click "Finish".
Depending on whether you have changed your preferences already, you might have or not the following alert popping up as soon as you hit "Finish":


Open Associated Perspective?

This kind of project is associated with the Java perspective.

This perspective is designed to support Java development. It offers a Package Explorer, a Type Hierarchy, and Java-specific navigation actions.

Do you want to open this perspective now?

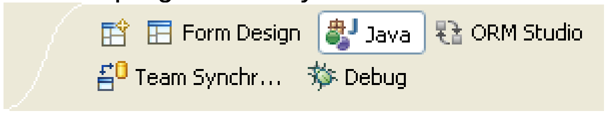
☐ Remember my decision

If you get this alert, click “Yes” that’s good, that’s exactly what we want, we want to open the “Java perspective” – you can also check the “Remember my decision” so that next time you create a Java project it will automatically open the associated Java Perspective.

D.A bit of perspective

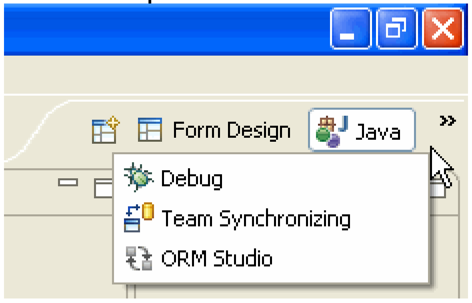
For those of you who don't know, a perspective in Eclipse is a set of "Views" and tools already laid out in the IDE. It allows you to organize the views for a particular project, for particular tasks. ("Views" are all the different panels that hold the tools that you will use for a task).
In our case we want to work on a Java project (a Servoy plugin IS a java project), so it is best if we use the standard "Java perspective" for a start*.

*You can always customize your perspectives (which views are opened and how they are laid out), and in fact Eclipse will remember how you organized its interface and will open a perspective to the same state as you have left it (if you have screwed up, don't worry, there is always an option to "Reset" your perspective to the default layout", use the menu "Window > Reset Perspective" or look at the top right corner you will see these icons, you can right-click on it an choose "Reset".

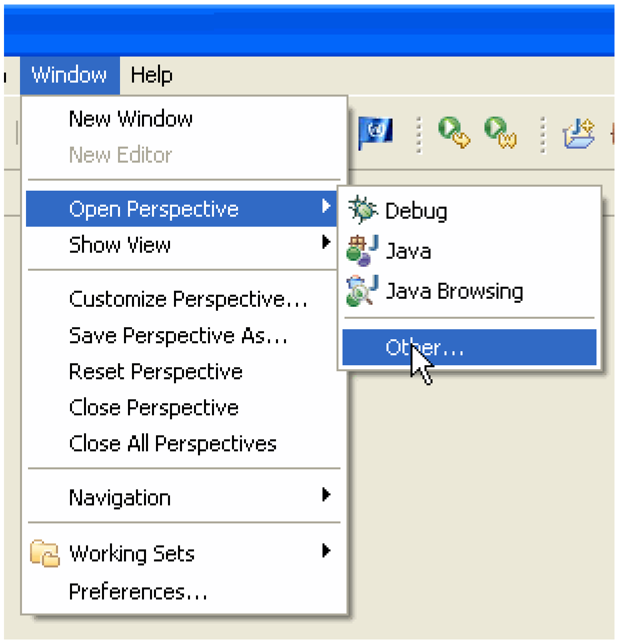


(You might see less or more - depending on what you have opened previously and the kind of plugins you have installed in Eclipse), but you should be able to see at least "Form Design" – the Servoy perspective – and "Java" – the one we are going to use).
You can click on these buttons to switch from one perspective to another, and you will see that all your views changes in one go. That's a pretty powerful way to switch tasks in Eclipse.

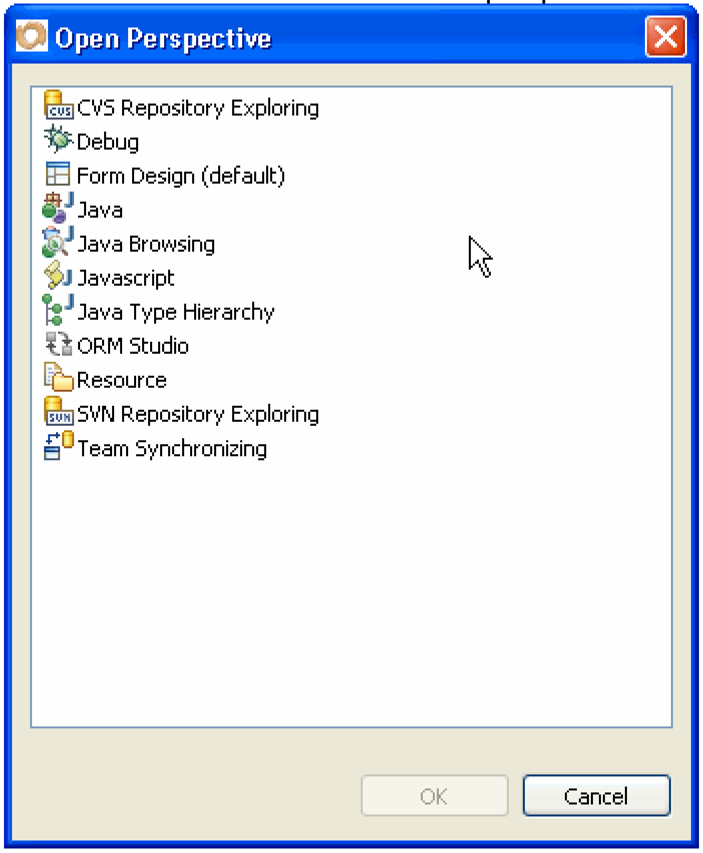
Note that you might have some hidden perspectives that you can switch too – in which case there will be a little ">>" arrow that you can click to reveal some others that you can't see because of the size of the panel:



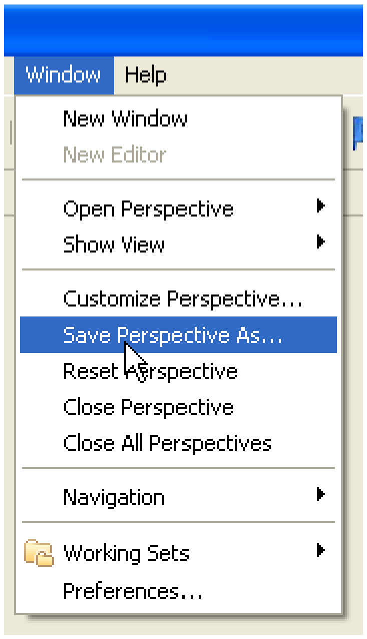
And if you can't find the perspective you want to use, you can always go to "Window > Open Perspective > Other..."



And choose from the list of available perspectives:



Note that you can also create you own perspective and that it's a very practical way of organizing your tasks for a project, all you have to do is (once you are satisfied with the layout of the different views and tools you have for a task) go to "Window > Save Perspective As...":

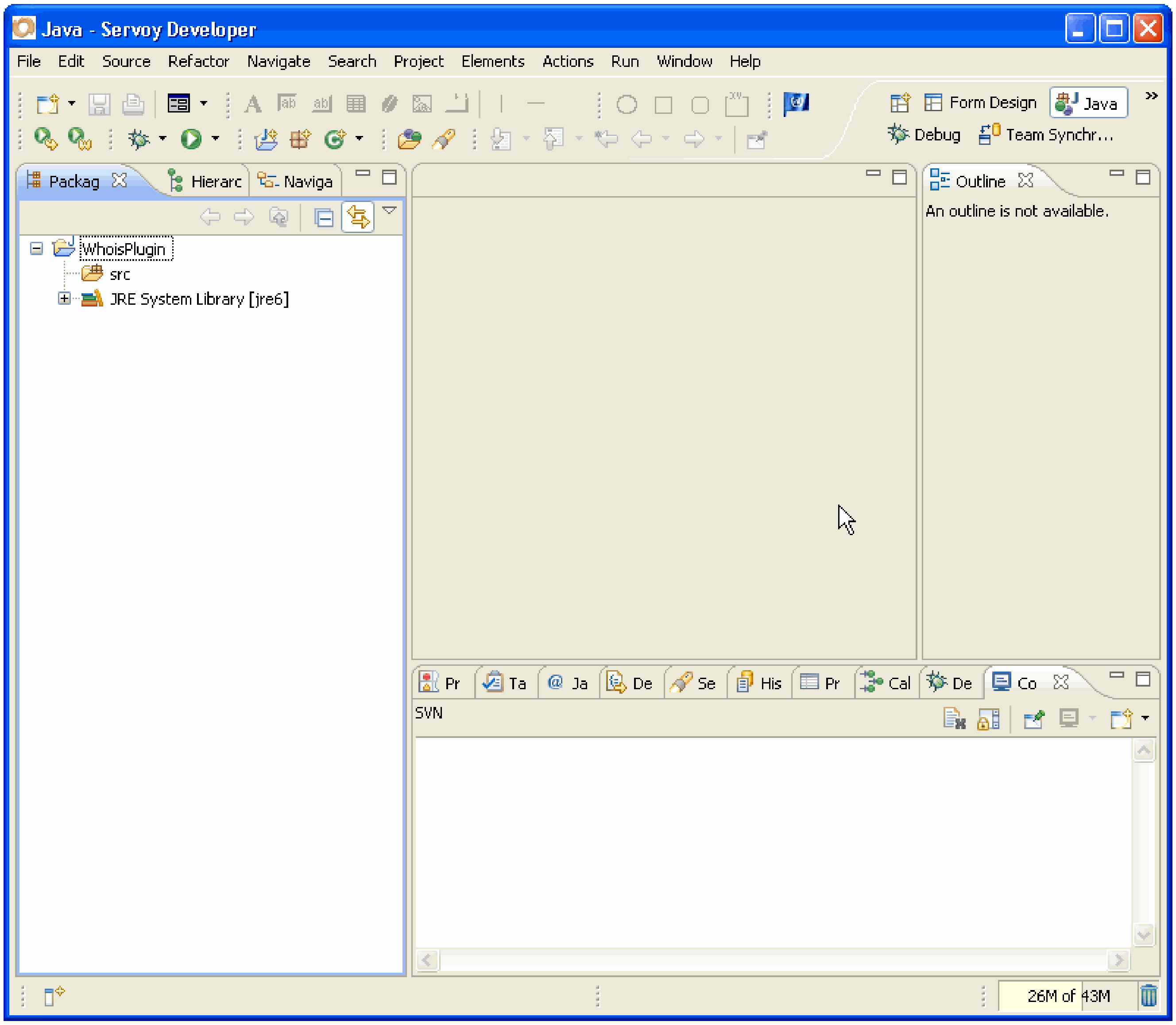


You can see that there is also a “Customize Perspective...” and as you have guessed you can pretty much configure your perspectives the way you want with a wealth of options.

See the post from David Workman of DataMosaic where he discuss the merit of his Mylin perspective: http://forum.servoy.com/viewtopic.php?f=12&t=12361&p=62544

E. Back to work!

After this big parenthesis about Perspectives, let’s get back to our project. We now have a WhoisPlugin Java project open in the Java perspective; it should look something like this:


Java - Servoy Developer
File Edit Source Refactor Navigate Search Project Elements Actions Run Window Help
Form Design
Java
Debug
Team Synchr...
Packag
Hierarc
Naviga
WhoisPlugin
src
JRE System Library [jre6]
Outline
An outline is not available.
Pr Ta Ja De Se His Pr Cal De Co
SVN
26M of 43M

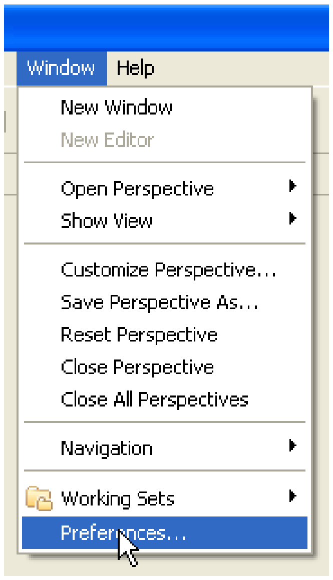
On the left panel you see the “Package” view opened, this is where we are going to select and store our java sources, you should also see a “JRE System Library [jreX]” where X is the version of the Java Runtime Environment that Eclipse used by default to work in your project. We will leave it at that for now. - If you open it though you will see the different jars that make the standard java language and related resources.

We’re not done configuring our environment yet, because to build a plugin for Servoy we will need to add references to the Servoy’s jars.
The best way to do that is to add a “User Library”. A “User Library” is a predefined set of jars that you organize so that it will be available all the time for all the relevant projects, it is similar to the “JRE System Library” (that holds the java language jars) except that it is a “User” one – you define its content yourself.

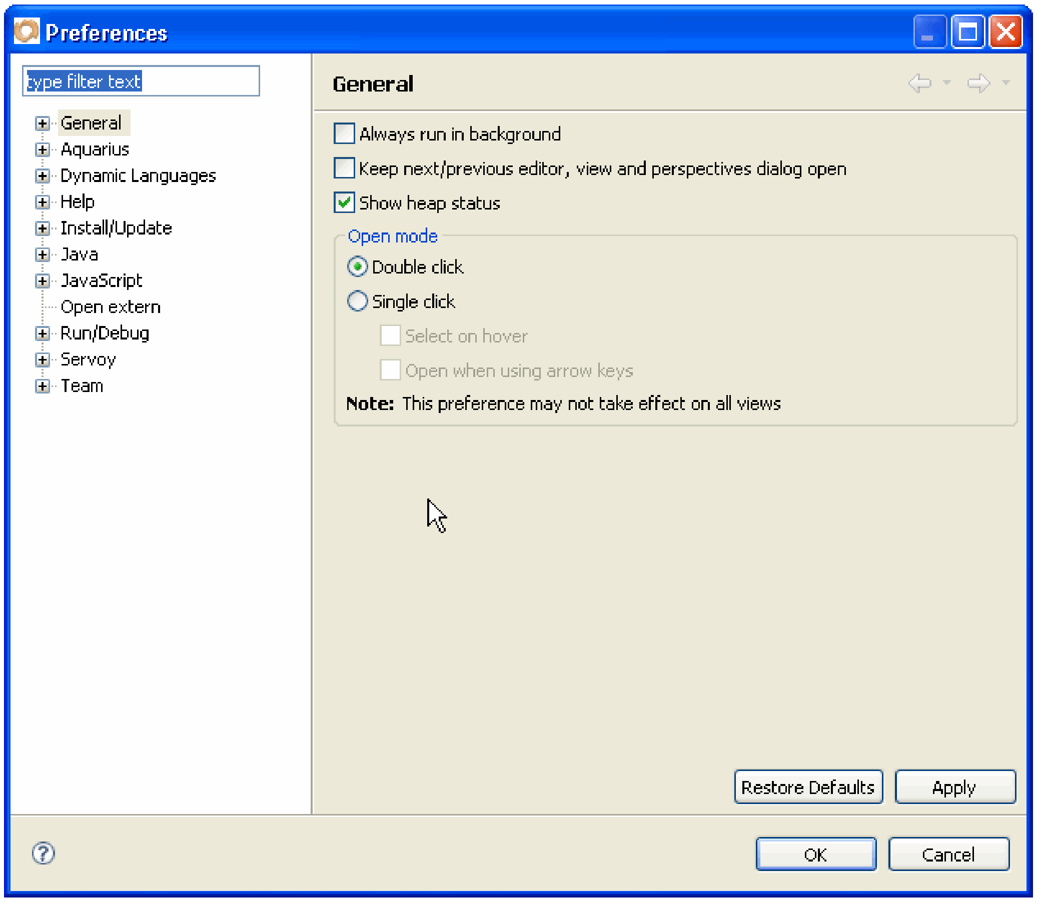
Since you are going to build lots of new and exciting plugins and beans for Servoy, it’s best to get that done once and for all!

1) Define the “Servoy User library”

There are lots of ways to define a library, and you will discover that by yourself or by following one of the many Eclipse tutorials but for now we are going to do it like this: go the “Window > Preferences...” menu



You will get the Eclipse “Preferences” dialog:



General

- [ ] Always run in background
- [ ] Keep next/previous editor, view and perspectives dialog open
- [✓] Show heap status

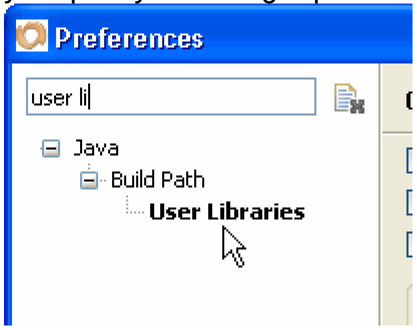
Open mode

- [● Double click]
- Single click
  - [ ] Select on hover
  - [ ] Open when using arrow keys

Note: This preference may not take effect on all views

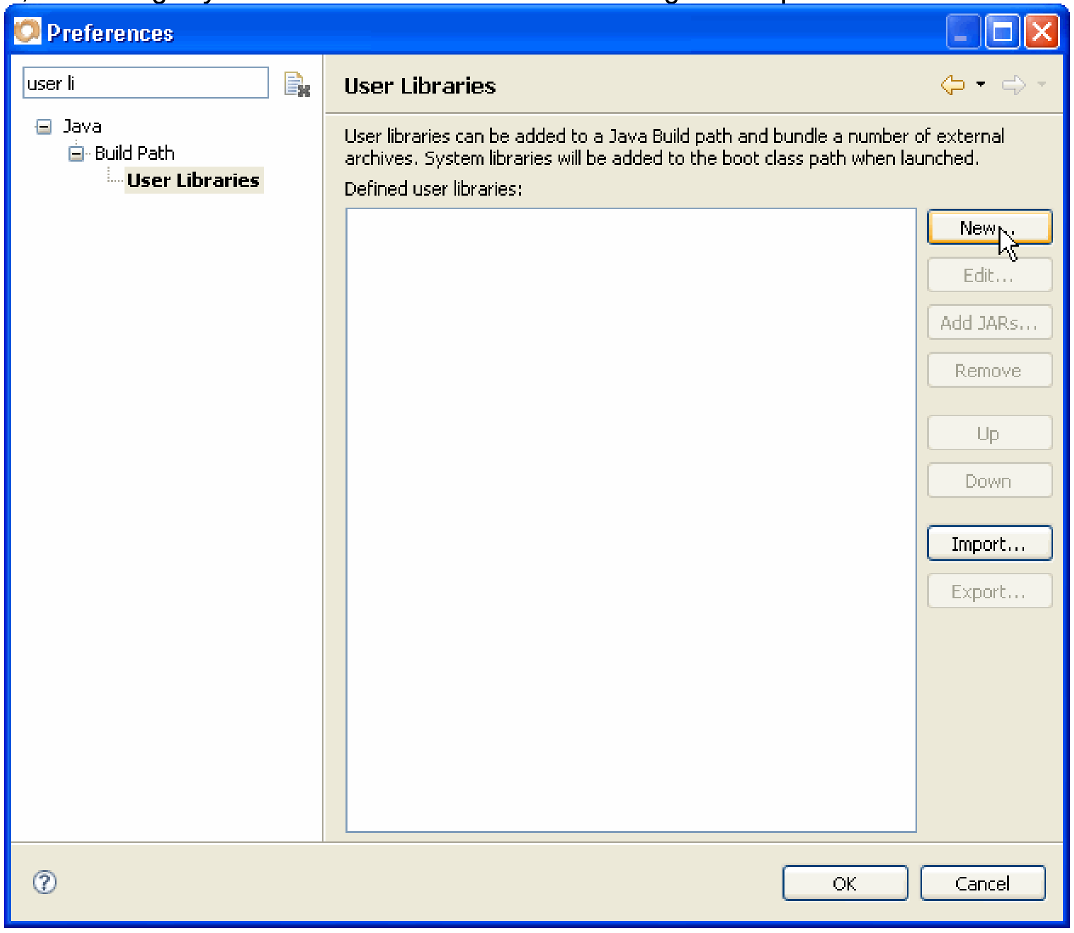
This is the one that makes people shy away from Eclipse because there is so much stuff that you can configure that it can be a little bit daunting the first time you come across it. But you shouldn't be afraid of it, it's very neatly organized by topics as you can see from the tree on the left, you can open each of the nodes and poke around, even if you're not changing a thing it's good to have a feel of how much you can customize your IDE to your needs and taste.

There is also a very neat little text box on the top left which can filter the features for you and get you quickly to the right parameters, that's what we are going to use right now by typing "user li"



You see that each time you time a letter the tree refreshes to filter only the relevant options (the kind of trick that you might want to implement in you own Servoy solutions, don't you think?)

By typing "user l" you will see that Eclipse found the right parameter, all you have to do is to click on it, and straight you are in the "User Libraries" configuration panel

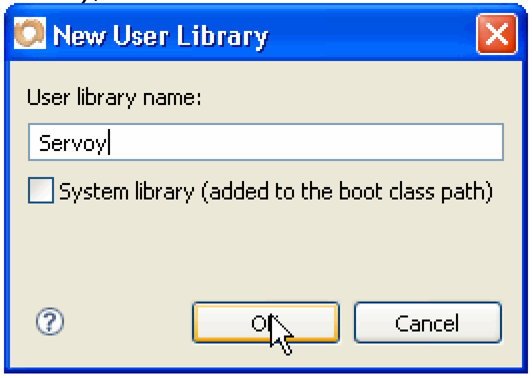


User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

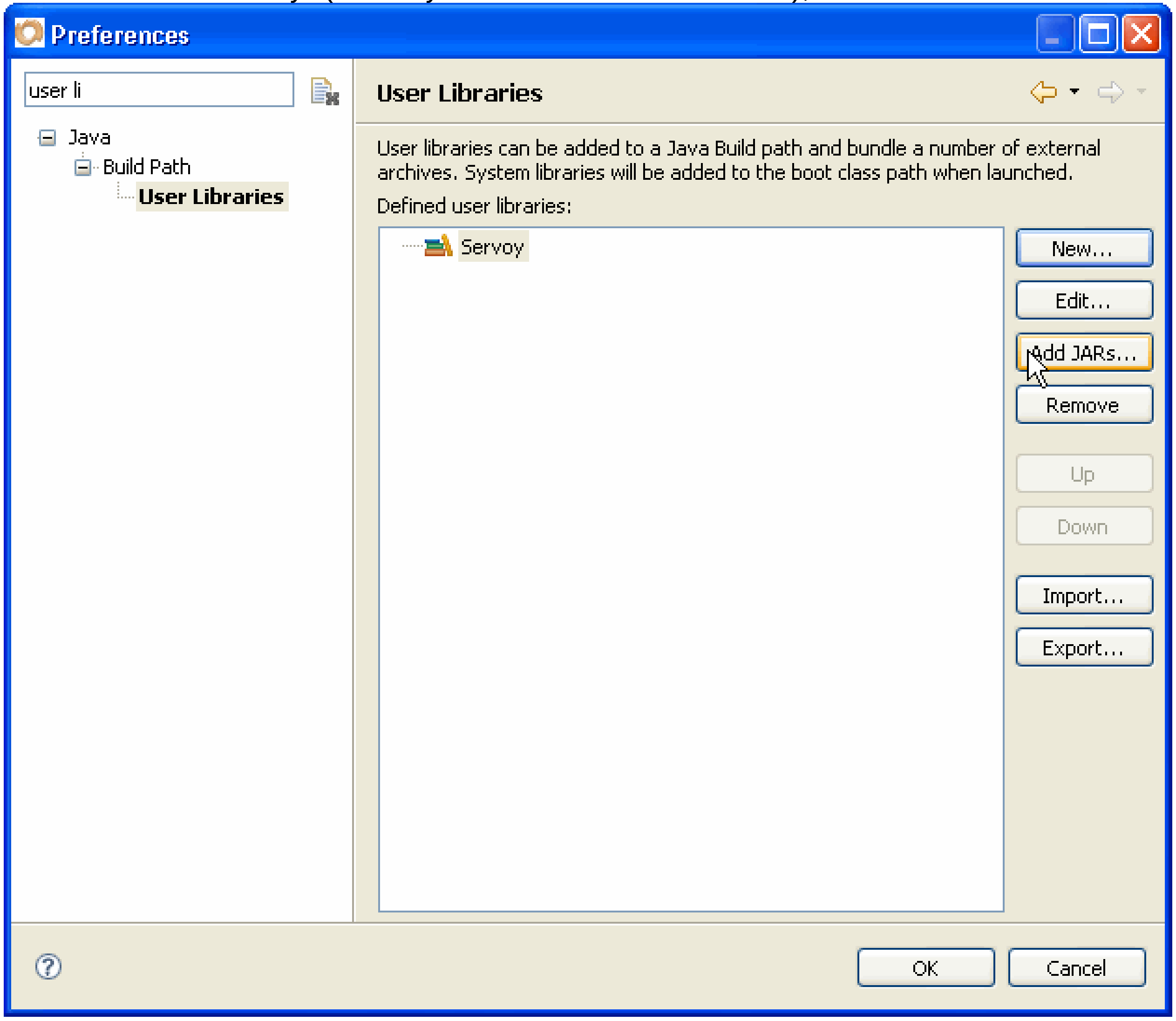
Defined user libraries:

The list of your user libraries might be empty or you might have already configured some before (in which case I don't know why you are losing your time with this part, you can skip directly to the next!☺).

We want to add a "Servoy" library to use in our project (and future projects for Servoy plugins and beans), so click "New..."



Give it the name "Servoy" (so that you will know what's in it later), click "OK".



User Libraries

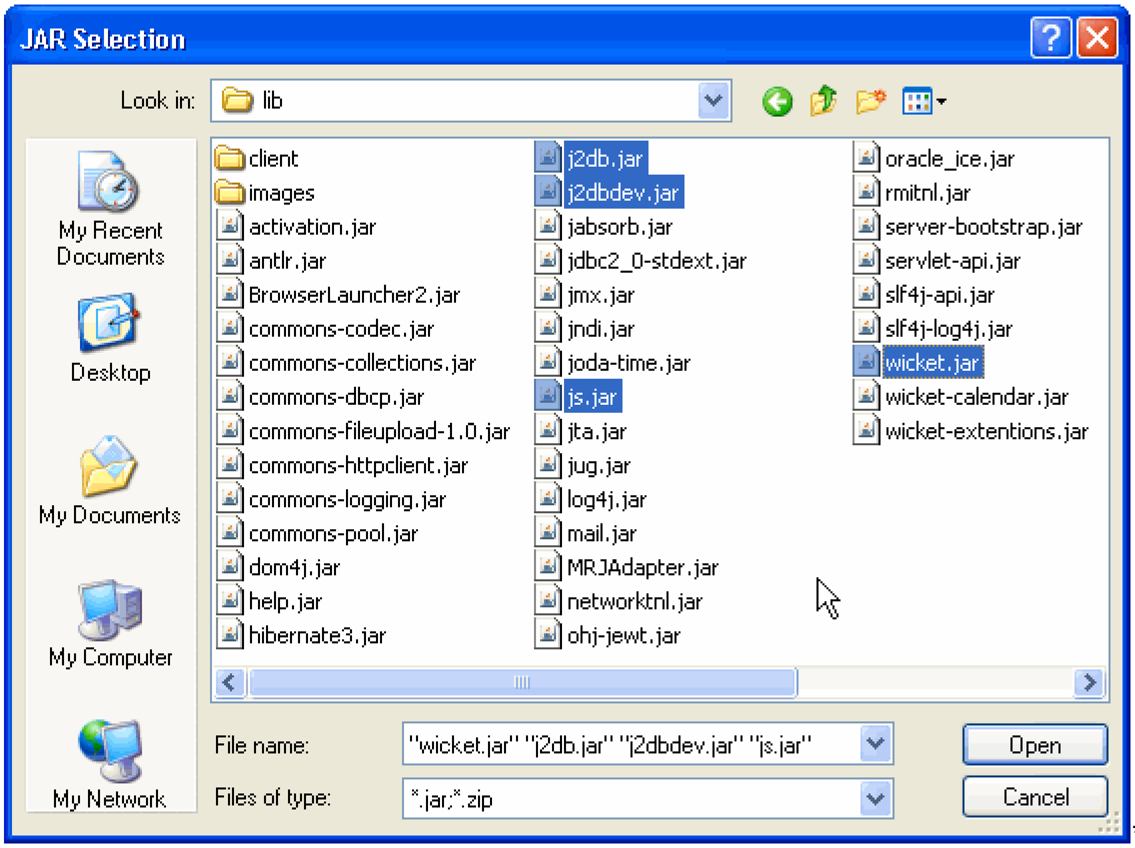
User libraries can be added to a Java Build path and bundle a number of external archives. System libraries will be added to the boot class path when launched.

Defined user libraries:

Servoy

Now we have our “Servoy” library, but it’s still empty, so we need to add the Servoy’s jar needed to build our plugins and beans, so click the “Add JARs…” button, this will open the “Open File” system dialog, prompting you to select the jars you want to add, so navigate to your “/ServoyInstallDir/application_server/lib/” folder and grab the following jars (that’s the minimal configuration if you are serious about making a plugin):

- j2db.jar
- j2dbdev.jar
- js.jar
- wicket.jar



File name: "wicket.jar" "j2db.jar" "j2dbdev.jar" "js.jar"

Files of type: *.jar;*.zip

*

*You can select one at a time or select all of them in one go, depending on whether you are cautious or just lazy like me and want to get on with the rest of your life as quick as possible.

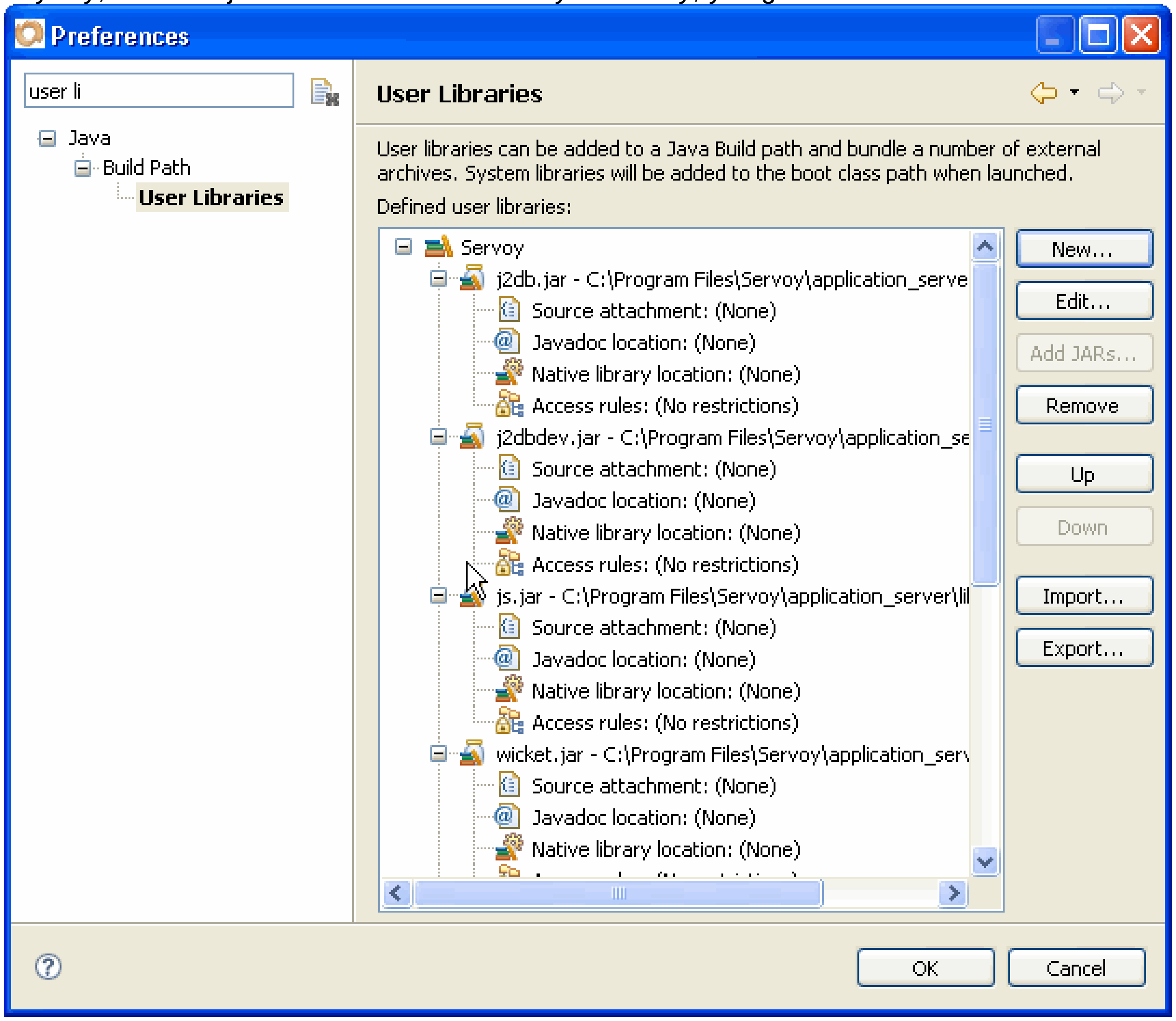
Note about the “minimum” selection of jars:
Personally I also added the wicket-calendar.jar, the wicket-extensions.jar and the joda-time.jar to my Servoy library (because depending on the kind of development you do you might need them, for example the DateChooserBean I did made use of the wicket-calendar for the web client bean and the joda-time library to manipulate date and time), but it’s up to you: you can always add them later.

Remember that even if you did add a reference to one jar, if you are not using it, it won’t hurt anyway, your project size won’t be bigger! Consider that it’s just an alias; in fact it will just add a reference to these libs into the classpath of the project you are going to add it to.

***Instead of linking directly to the “/ServoyInstallDir/application_server/lib/” folder that holds the Servoy library, you might prefer to make a copy of the relevant jars in another folder. Why would you

do that? You need to understand that since you made a reference only to these jars (they are not copied in your project as such, their location only will be added to the classpath of your projects), the next time you update Servoy, these libs might change, so the project that was working so well with 4.1.3 might be broken for Servoy 4.1.4 if the API changed or some behaviour changed. It is debatable whether that's a good thing or a bad thing.
You might want to copy these jars in a safe place and update them manually because this way you will be able to save them and if needed build a different version of your project for a different Servoy version target. That's especially true for major updates, API have a greater chance of changing for these updates than they will for minor ones. In any case, that's something to keep in mind.

Anyway, once the jars selected and added to your library, you get a window like that:



User Libraries

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

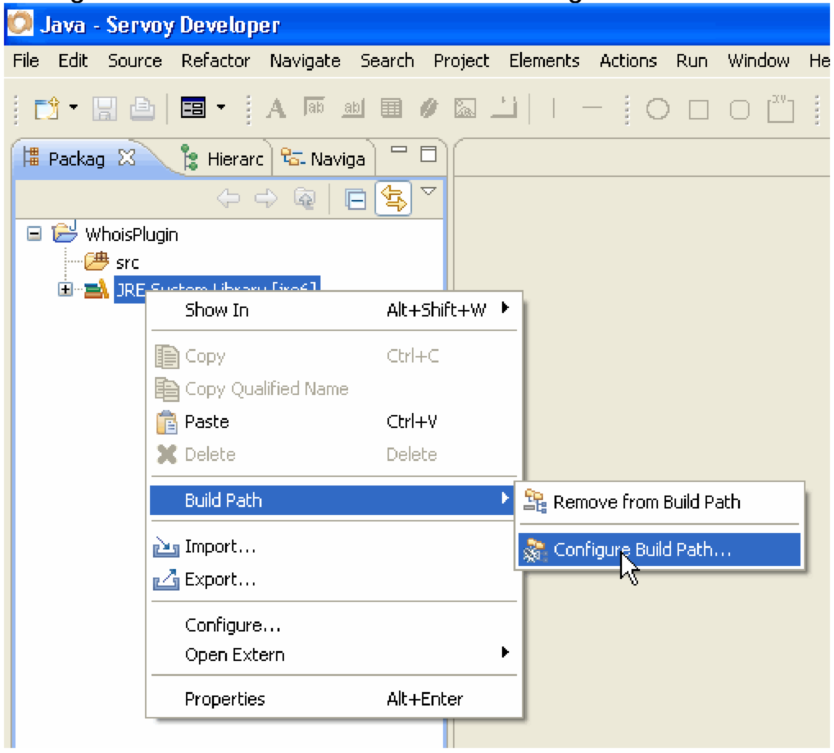
Defined user libraries:

```
Servoy
  j2db.jar - C:\Program Files\Servoy\application_serve
    Source attachment: (None)
    Javadoc location: (None)
    Native library location: (None)
    Access rules: (No restrictions)
  j2dbdev.jar - C:\Program Files\Servoy\application_se
    Source attachment: (None)
    Javadoc location: (None)
    Native library location: (None)
    Access rules: (No restrictions)
  js.jar - C:\Program Files\Servoy\application_server\li
    Source attachment: (None)
    Javadoc location: (None)
    Native library location: (None)
    Access rules: (No restrictions)
  wicket.jar - C:\Program Files\Servoy\application_serv
    Source attachment: (None)
    Javadoc location: (None)
    Native library location: (None)
```

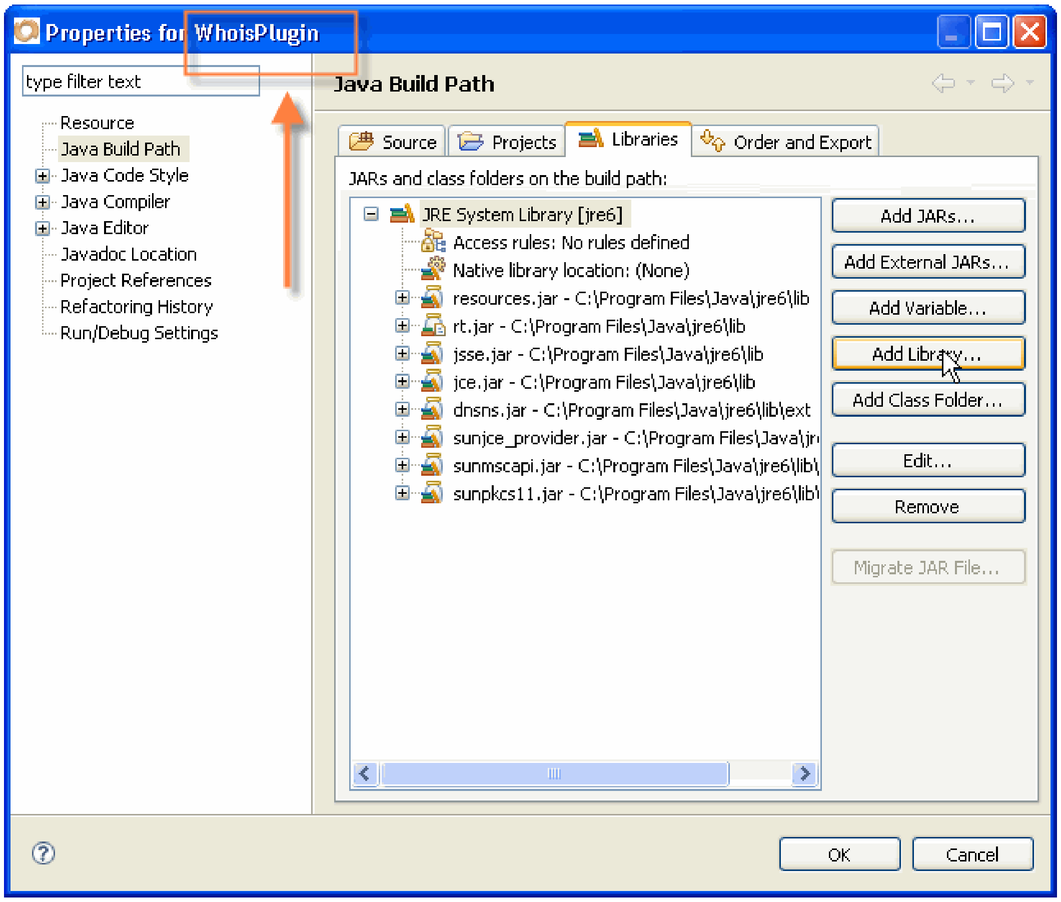
Note that you can add Source and Javadoc (if you have them), in the case of js.jar (which is Rhino) and of Wicket, they are freely available on the internet on their respective project sites (send me a note if you can't find them, but these projects are not exactly confidential, so a quick search on the internet will get you right to them).

2) Add a reference to the “Servoy User Library” to our project

Now that we have our User library nicely configured, just click OK to save it, we will now attach it to our project. To do that, back to the “Package” explorer, click on the “JRE System Library [jreX]” node and right click to choose “Build Path > Configure Build Path...”:



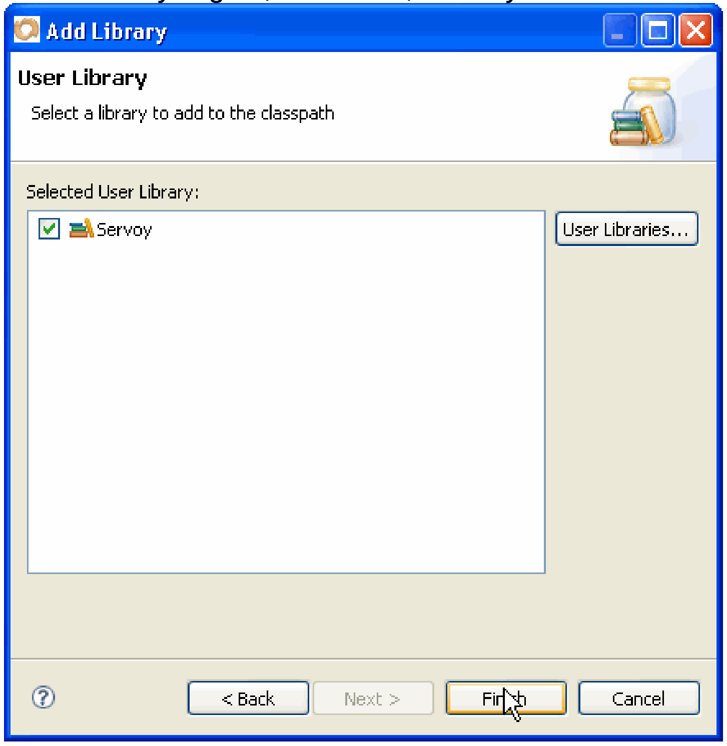
We’re back to our Properties Window but this time with a twist; note the “Properties for WhoisPlugin” title of the window:



JARs and class folders on the build path:

```
JRE System Library [jre6]
    Access rules: No rules defined
    Native library location: (None)
    resources.jar - C:\Program Files\Java\jre6\lib
    rt.jar - C:\Program Files\Java\jre6\lib
    jsse.jar - C:\Program Files\Java\jre6\lib
    jce.jar - C:\Program Files\Java\jre6\lib
    dnsns.jar - C:\Program Files\Java\jre6\lib\ext
    sunjce_provider.jar - C:\Program Files\Java\jr
    sunmscapi.jar - C:\Program Files\Java\jre6\lib\
    sunpkcs11.jar - C:\Program Files\Java\jre6\lib\
```

We are getting close, so choose "Add Library..." - you get the same choice of "JRE System Library" or "User Library" again, click Next, and if you followed everything properly, you should see this window:



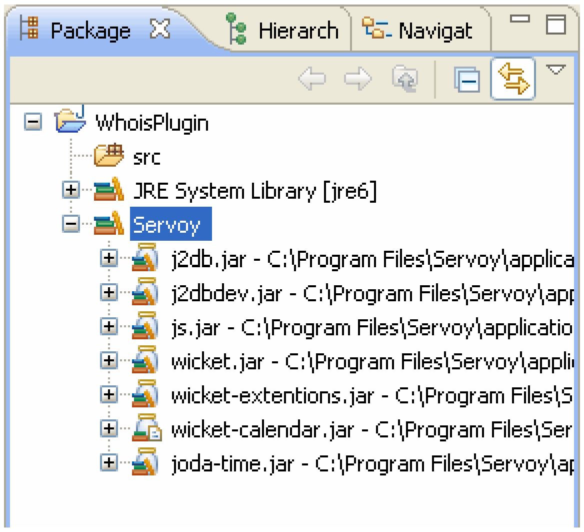
Select a library to add to the classpath

Selected User Library:

☑ Servoy

So check the "Servoy" checkbox, and click "Finish".

That's it! The "Servoy" user library has now been added to our project, you can check it in the Package Explorer, expand the new "Servoy" node to see your added jars, and we are now ready to get to work on our plugin.



```
WhoisPlugin
  src
  JRE System Library [jre6]
  Servoy
    j2db.jar - C:\Program Files\Servoy\applica
    j2dbdev.jar - C:\Program Files\Servoy\app
    js.jar - C:\Program Files\Servoy\applicatio
    wicket.jar - C:\Program Files\Servoy\appli
    wicket-extentions.jar - C:\Program Files\S
    wicket-calendar.jar - C:\Program Files\Ser
    joda-time.jar - C:\Program Files\Servoy\ap
```

Note that next time you'll want to write a plugin/bean project you will be able to skip the first part of that section and get straight to the second: "Add a reference to the Servoy lib".

Now is time to start writing some code; and I know all of you are dying to get your hands dirty, so we'll get right to it after the break ;-)

See you on the next part of the tutorial!

Patrick Talbot
Servoy Stuff
2009-06-07