Develop with NWDI using External Libraries —with EP Example

Applies To:

NW2004
NW2004S

Article Summary

This technical article will outline how a developer who wants to use the NetWeaver Developer Infrastructure (NWDI) for their day to day development but does not have the required APIs delivered in a predefined Software Component can leverage external libraries for build and run time dependencies. For example, most portal libraries (including the PCD Generic Layer API, data caching service API, etc) are not delivered in a Software Component until a future release of NW2004S, although they are available in the portal at runtime. Thus the developer needs to know how to build their DC's for deployment into the target environment. This technical paper will focus on the Enterprise Portal and how a developer can use common portal libraries as external libraries to create public parts for their Development Components (DCs) within NWDI.

By: Marty McCormick
Title: Technical Consultant – SAP America
Date: 09 February 2006

Brief Overview

This article assumes that the developer is already familiar with NWDI and its components—the Design Time Repository (DTR), Component Build Service (CBS) and Change Management Service (CMS).

Currently, there is not a software component (SC) delivered with NW2004 or 2004S (although 2004S is planned to ship portal SCs at time of this publication) that contains the commonly used portal libraries. SAP Note 906973 states that developers should use External Library DC's to satisfy build time dependencies for portal development (par files). This is a different approach from the days of being able to use class finding tools such as jarclassfinder and right clicking to add a jar to the project's classpath. This no longer works with DC's because at build time they recreate their classpath based on the "Used DCs" listed in the DC's definition (thus deleting any entries you added). However this initial work is more than worth it because using public dependencies based on external libraries allows any developer to check out your project from the DTR and be build ready. This avoids the tedious process of having a developer send you their par and you having to figure out which import belongs to which jar—which is a very common complaint of Java developers today.

Building an External DC

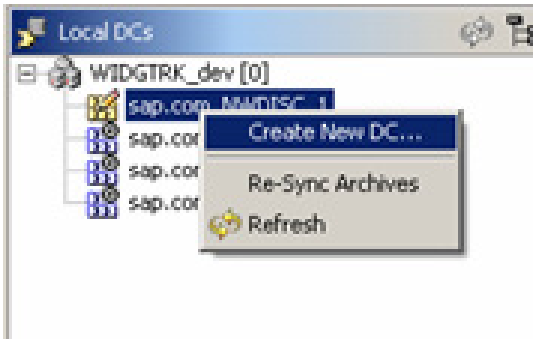
The following example will assume that a developer wants to use the following import in their Java Class file:

```
import com.sapportals.portal.pcd.gl.IPcdGlService;
```

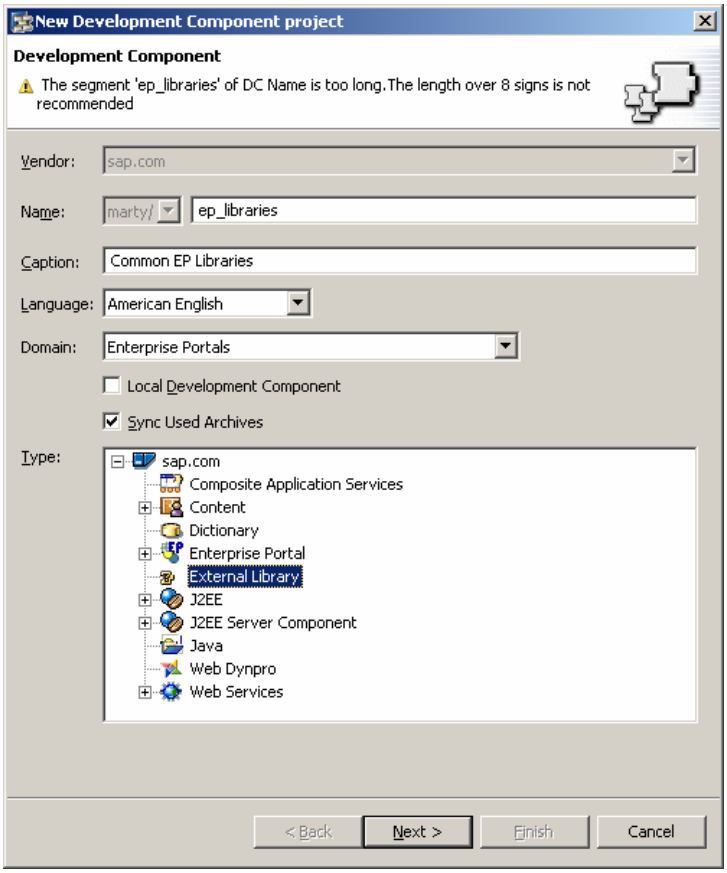
This class is available in the com.sap.portal.pcd.glserviceapi jar file. If a developer wants to use this class in their DC, they will need to declare a dependency to this jar file. Thus, the developer would need to reference a dependency to a public part that is made available via an external library.

After launching NetWeaver Developer Studio, switch to the “Development Configurations Perspective” (Window → Open Perspective→Development Configurations Perspective).

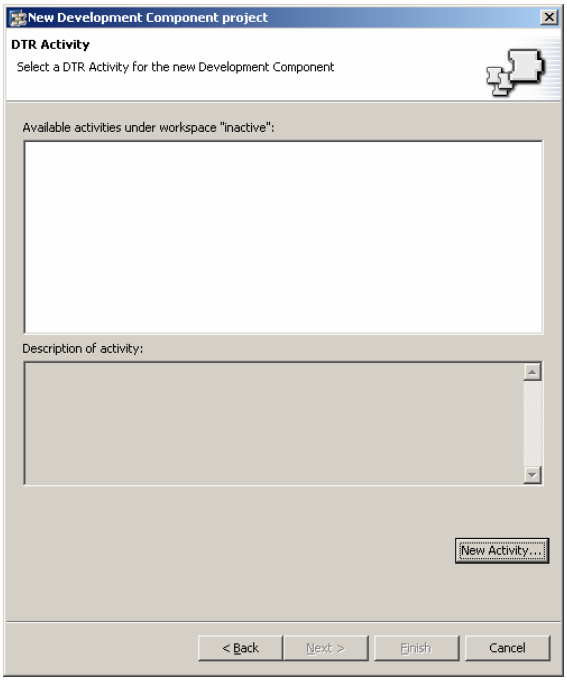
From within the Local DC’s view, right click on your Software Component and select **Create New DC**.

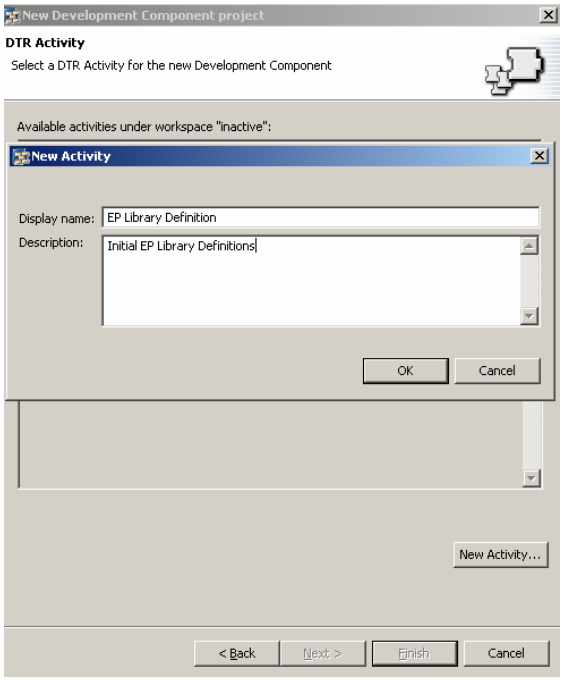


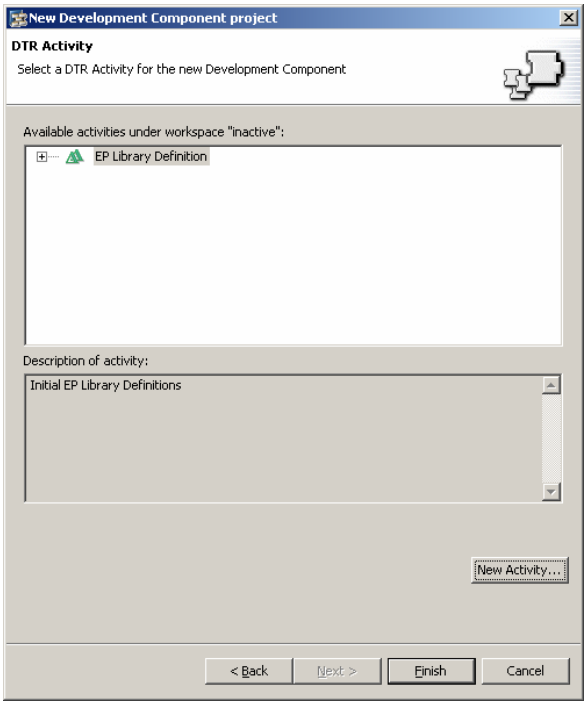
Enter your data as required by your organization. Even though we’re only going to expose one jar file as a public part in this example, you could still expose multiple jar files as public parts for one development component. Hence, we’re going to name the DC “ep_libraries” so that we can create public parts as required. Select the External Library type DC. Also, we’re going to assume that you immediately want to add it to the DTR and activate the component for other developers to use which is why I left “Local Development Component” unchecked.



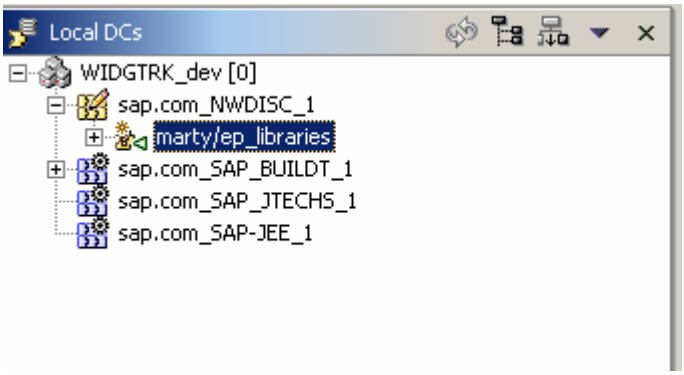
If necessary, create a new activity when required.







You should now have a new DC created like such:



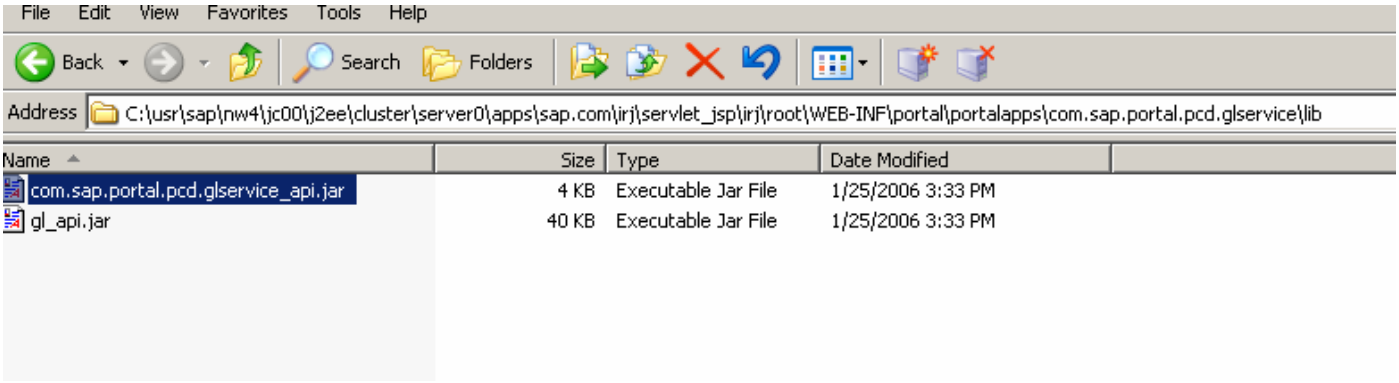
Add the Required Library (jar file) to the Project

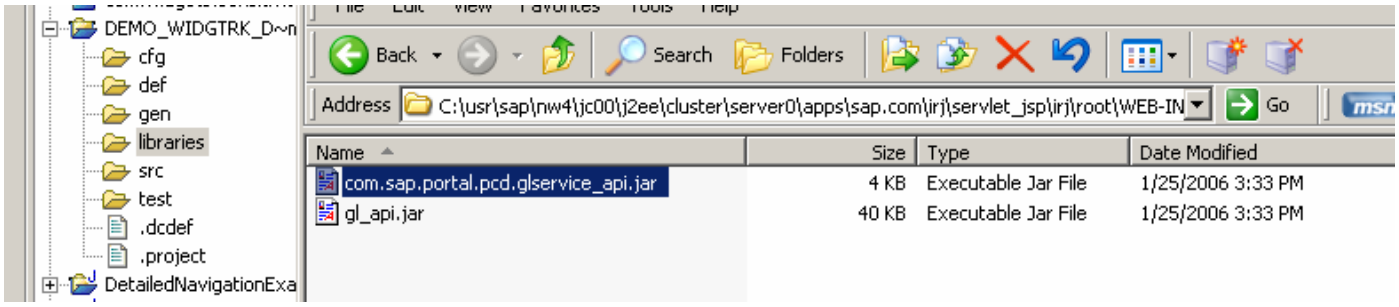
Open up the Resource Perspective (Window→Open Perspective→Resource Perspective) or by clicking on the left hand toolbar icon.

(NOTE: In this next step you can choose to add the file by right clicking on the libraries folder and adding a file to the folder by linking to on the file system or drag and dropping it. I'll show the drag and drop approach.)

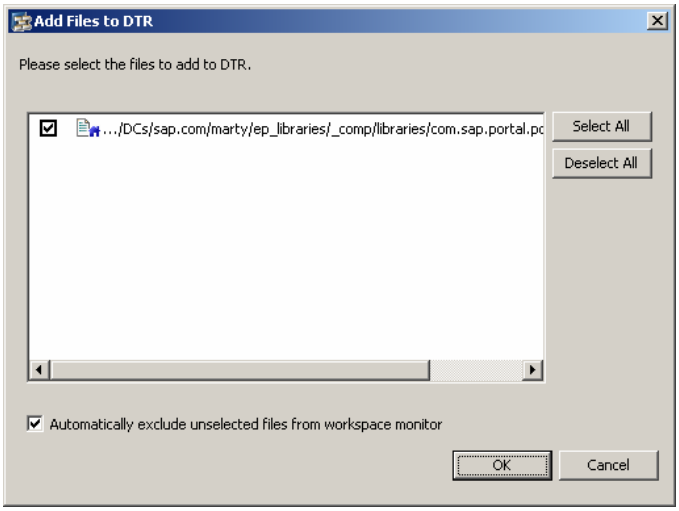
After opening the resource perspective, you should see your project listed in the workspace.

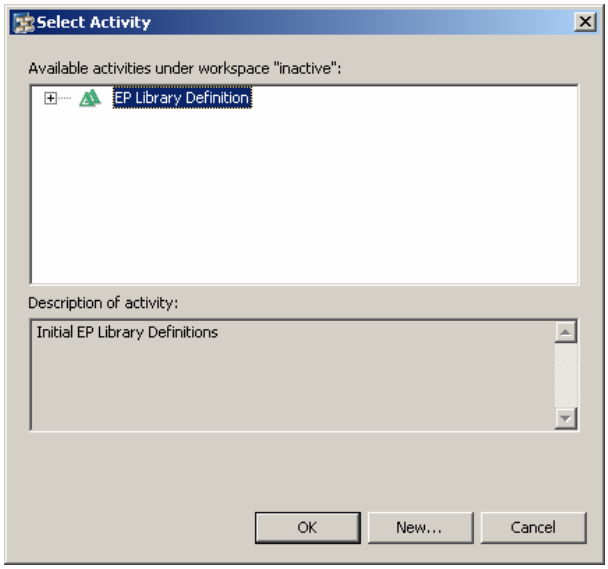
Open the folder that contains the jar file you wish to add as a library:



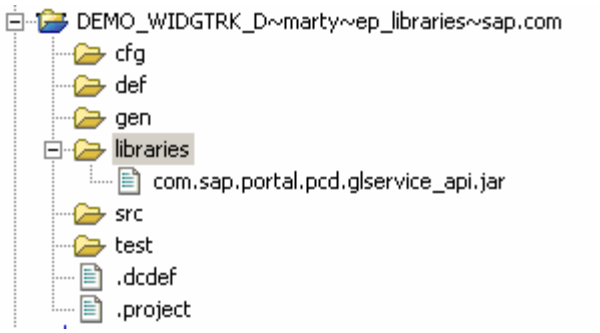


If you selected to add the DC directly to the DTR, you will be prompted to add the files to the DTR and assign them to an Activity.



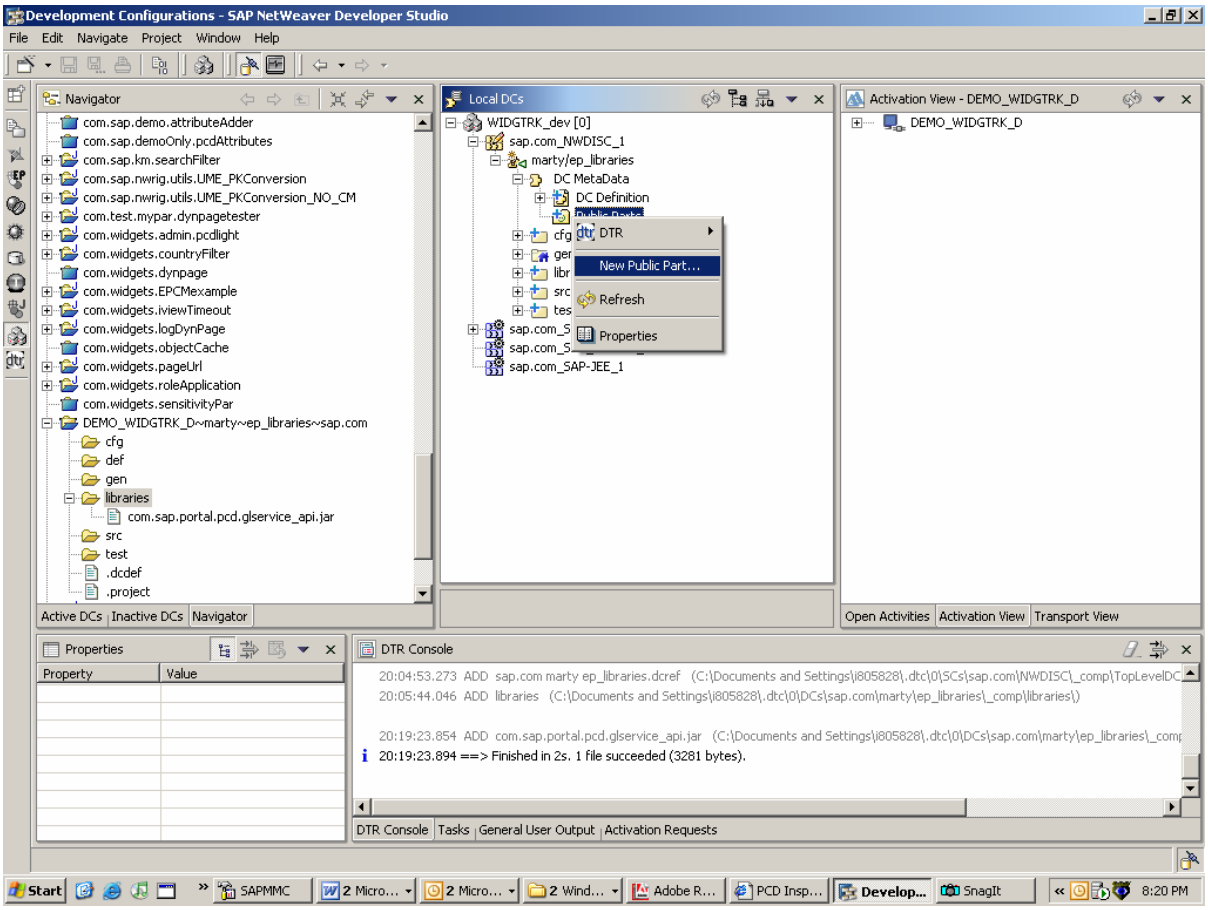


You should now see your new jar file listed under the libraries folder like such:

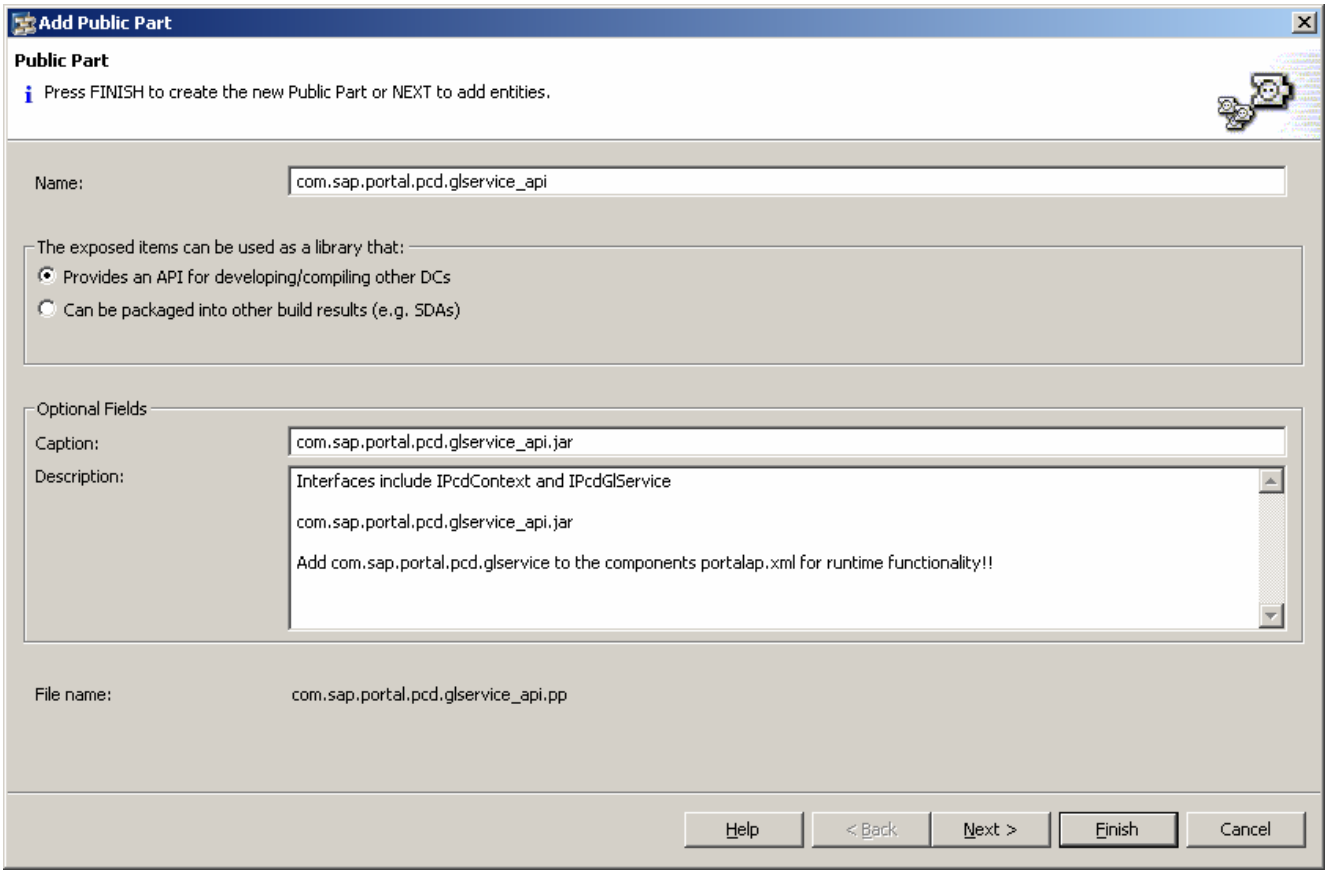


Create Public Part for Library

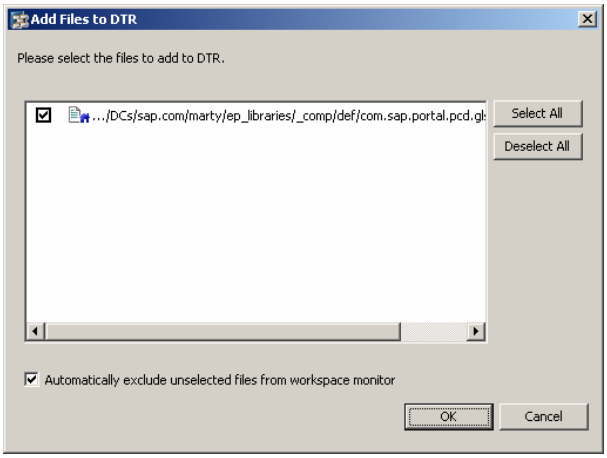
The next step is to create a public part for the library. Open the Development Configurations Perspective, expand out your DC project and right click on the Public Parts folder. Select “New Public Part….”

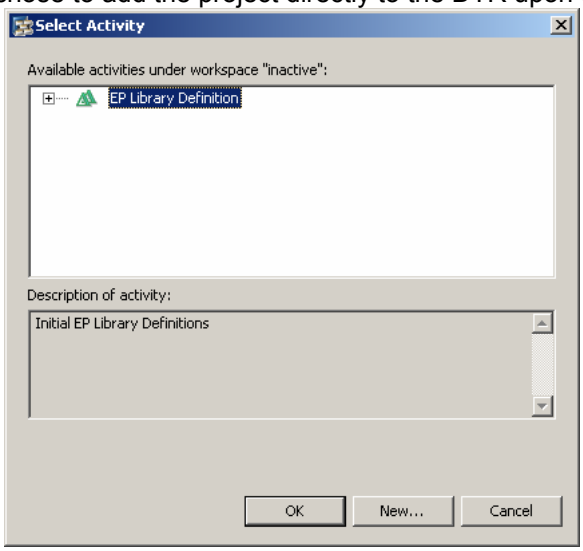


In the “Add Public Part” part wizard, enter the relevant information that pertains to your public part library



Again, you will be asked for DTR information if you chose to add the project directly to the DTR upon creation.

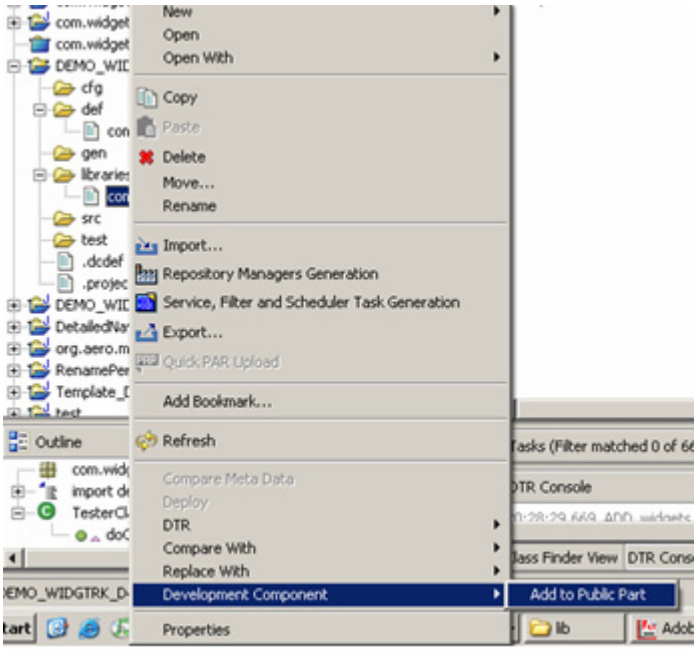




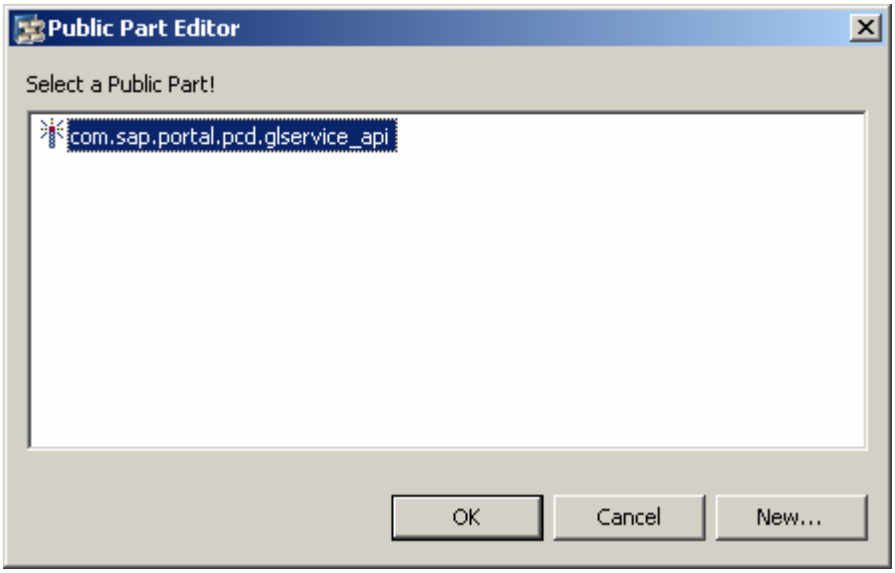
You have now created the public part of your component.

Associate Library with Public Part

The final step is to associate your jar file with the public part. Browse back to the Resource Perspective and right click on the jar file that you added under your libraries folder. Select “Development Component” → “Add To Public Part”



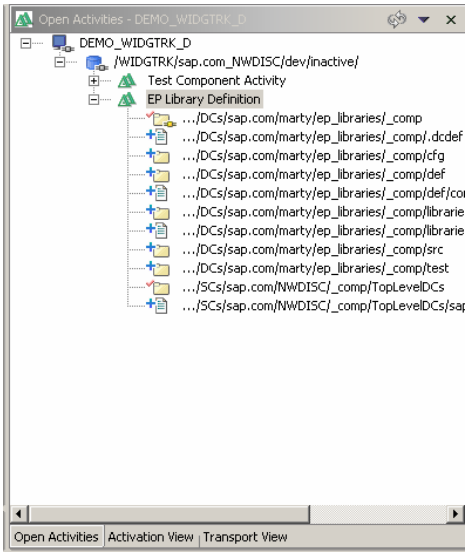
Choose the name of your public part interface created in the prior step.

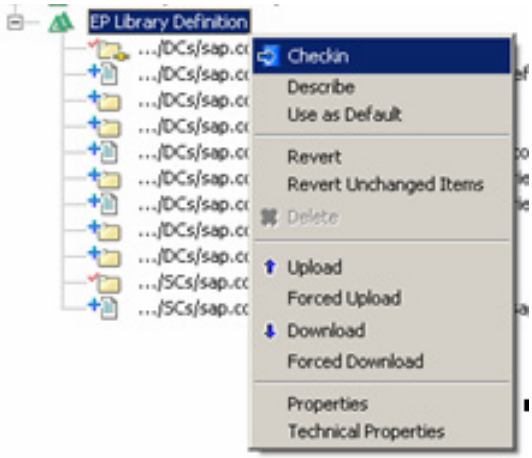


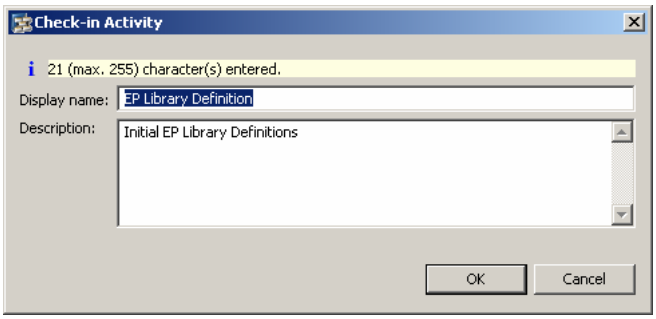
You have now associated the correct library with the public part.

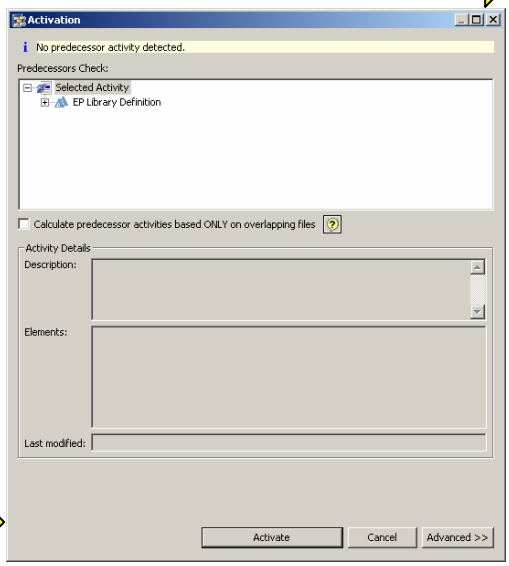
Check in and Activate DC

In order for you to use the public part in your DC's (and for others to use), check in the activity like a normal DC. You may also wish to activate the DC it as well. This follows the normal procedure for checking in and activating DCs.



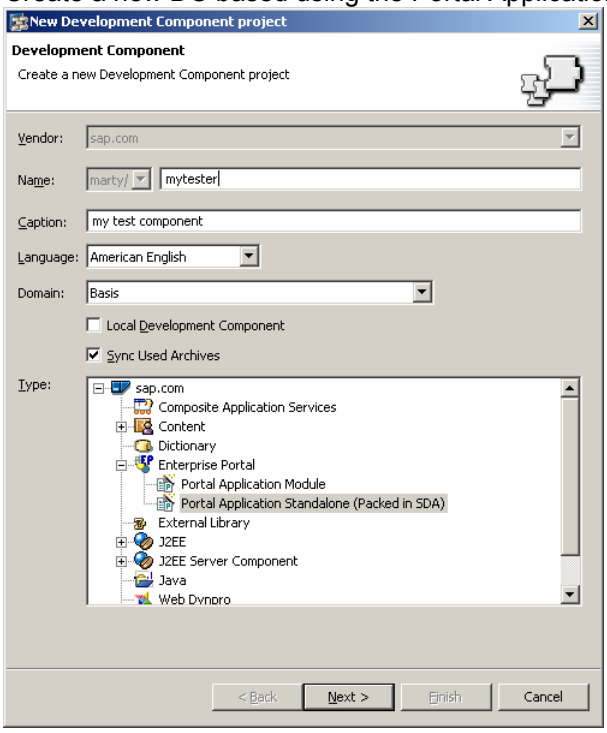


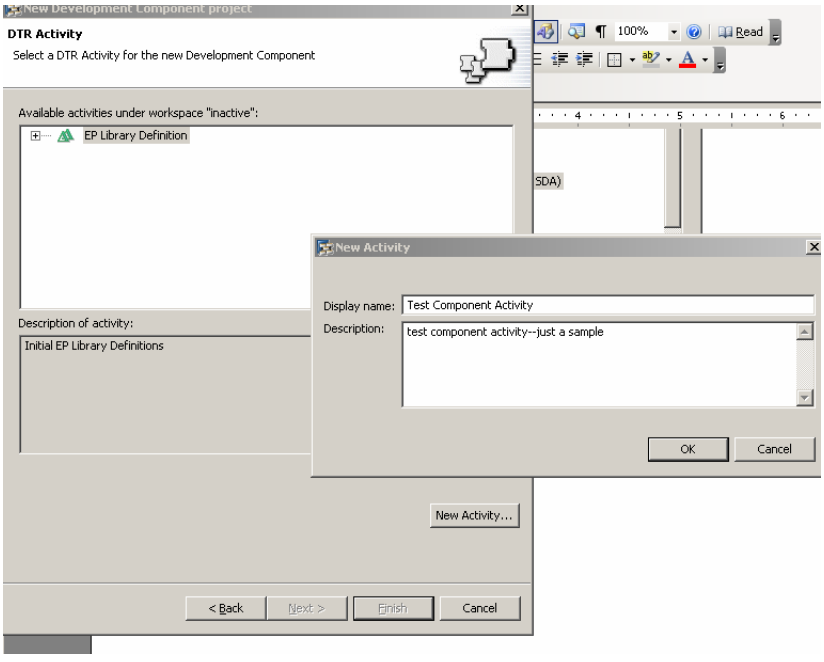


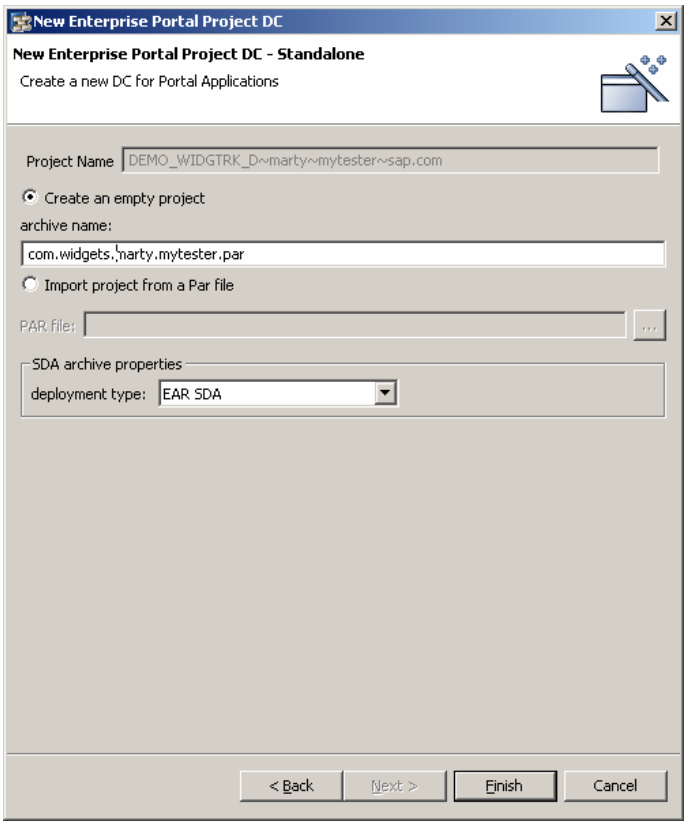


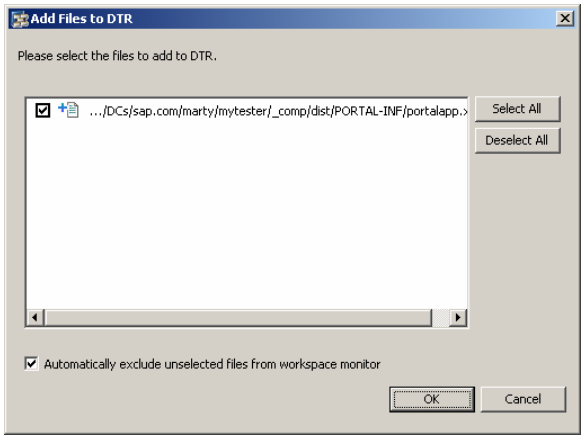
How to Use Public Part

Create a new DC based using the Portal Application project type.

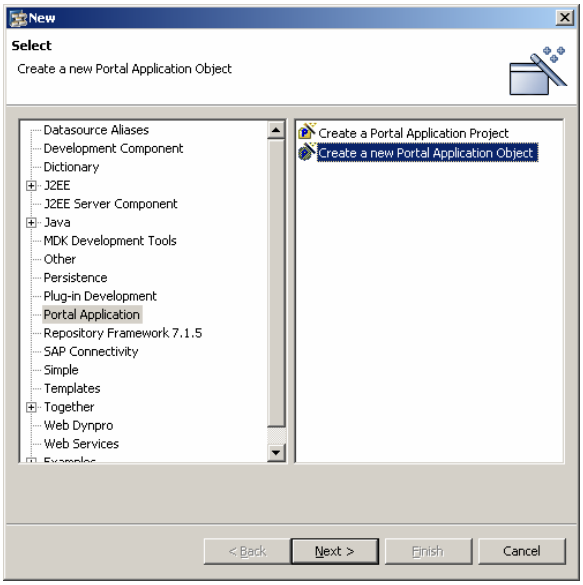


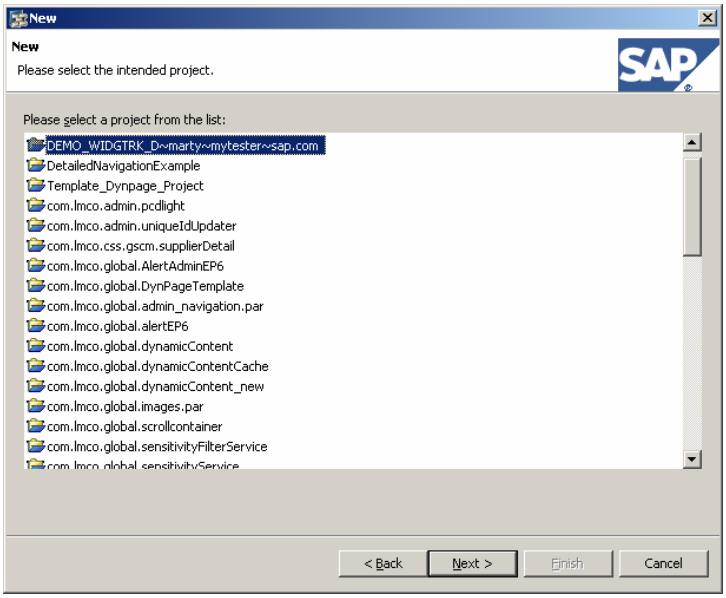


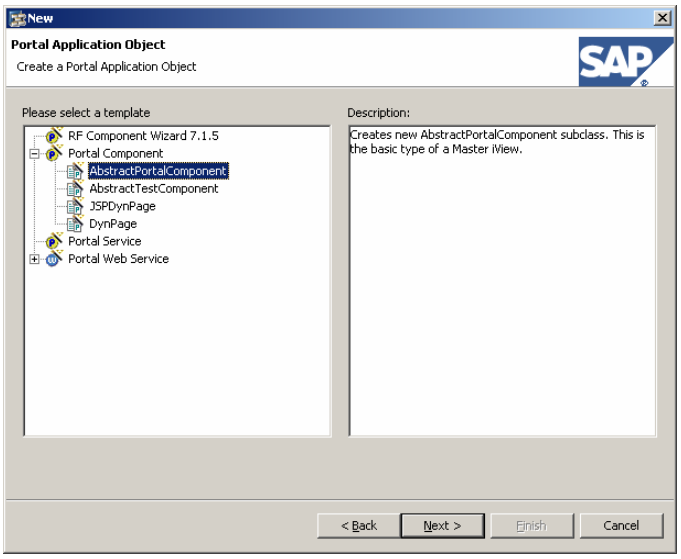


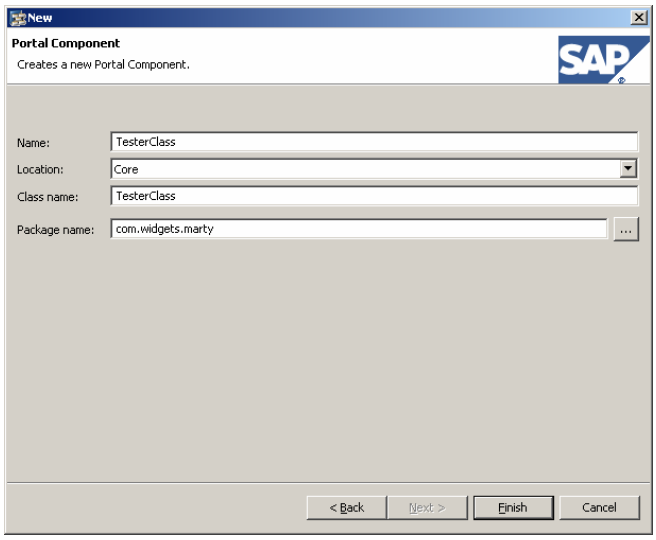


After the project is created, add a new Portal Application Object. In this example, we'll just use an AbstractPortalComponent.

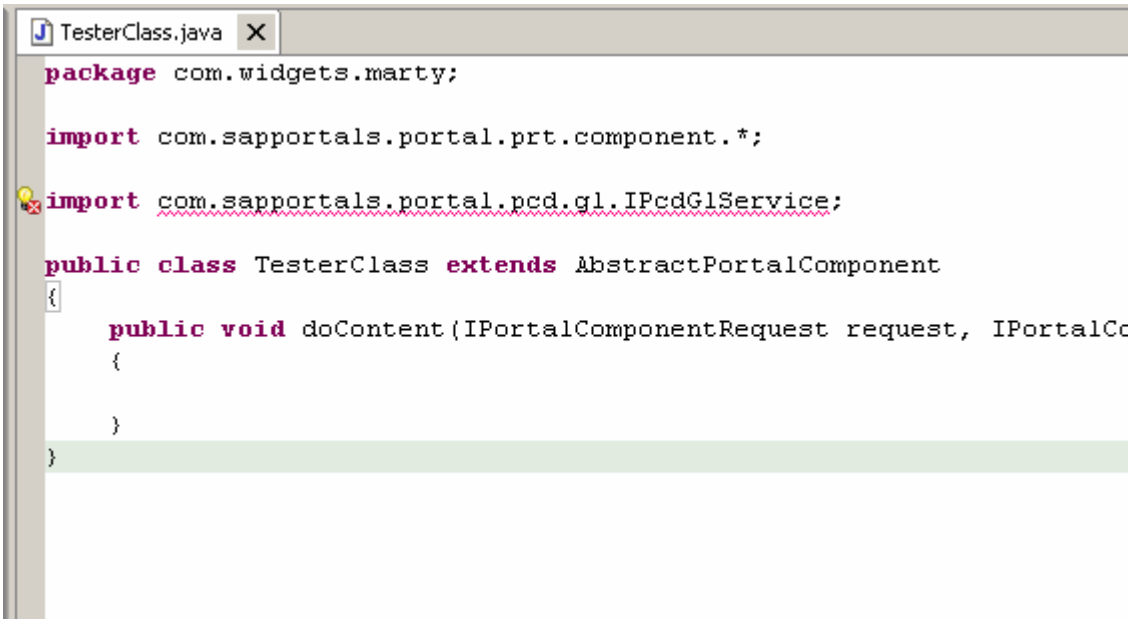




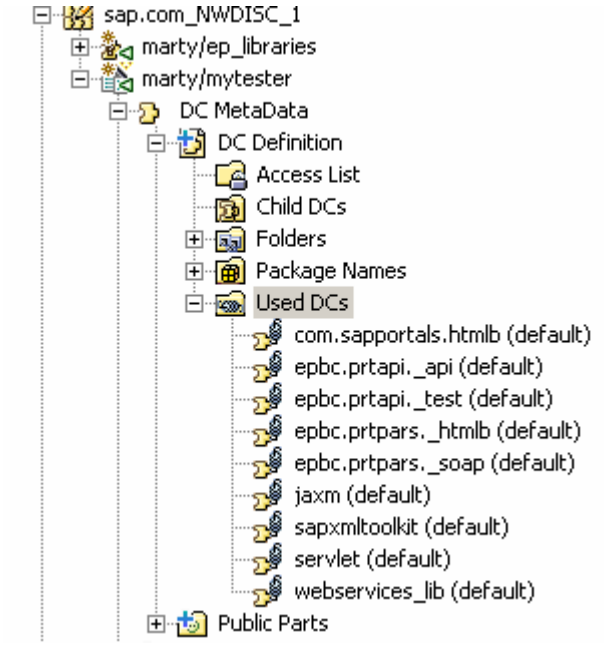




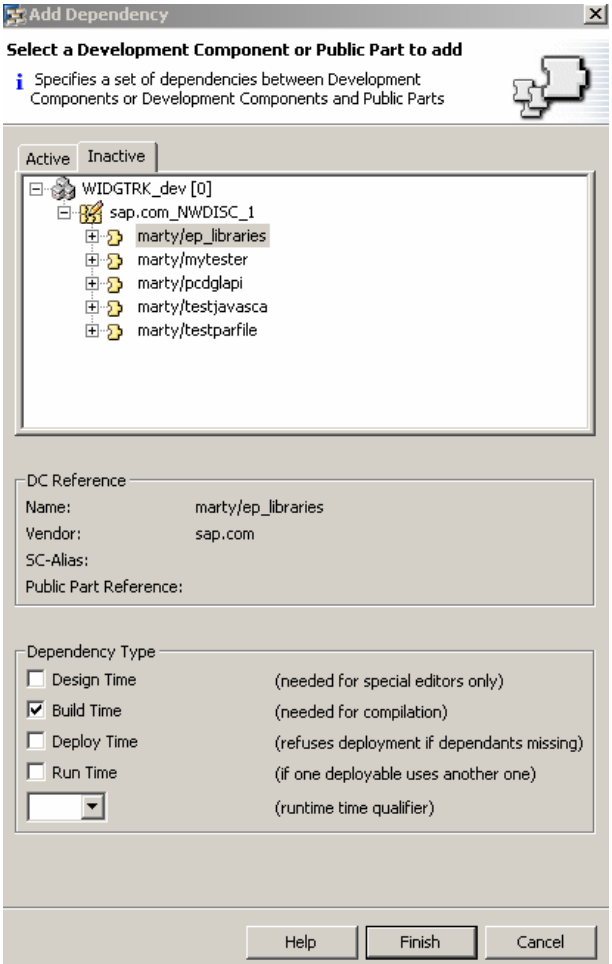
Create the import statement for the IPcdGlService and notice that the interface will not resolve.



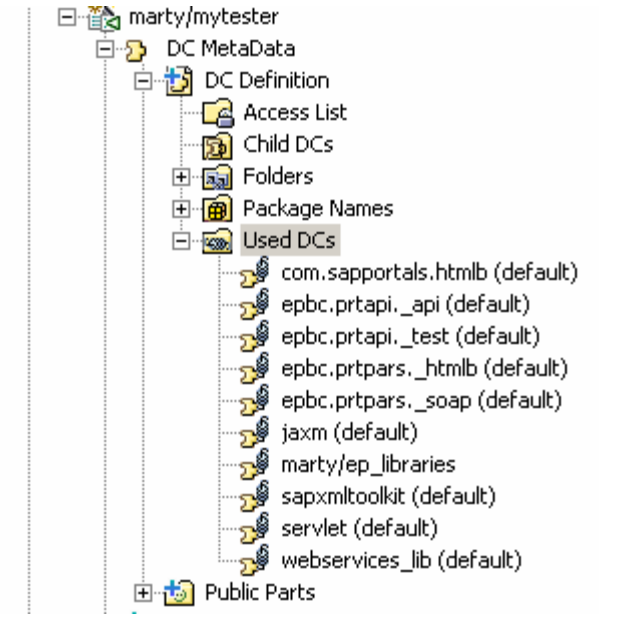
Browse back to the Development Configurations Perspective, open the current project and right click on the “Used DC’s” folder. Select “Add New Used DC…” from the option menu.



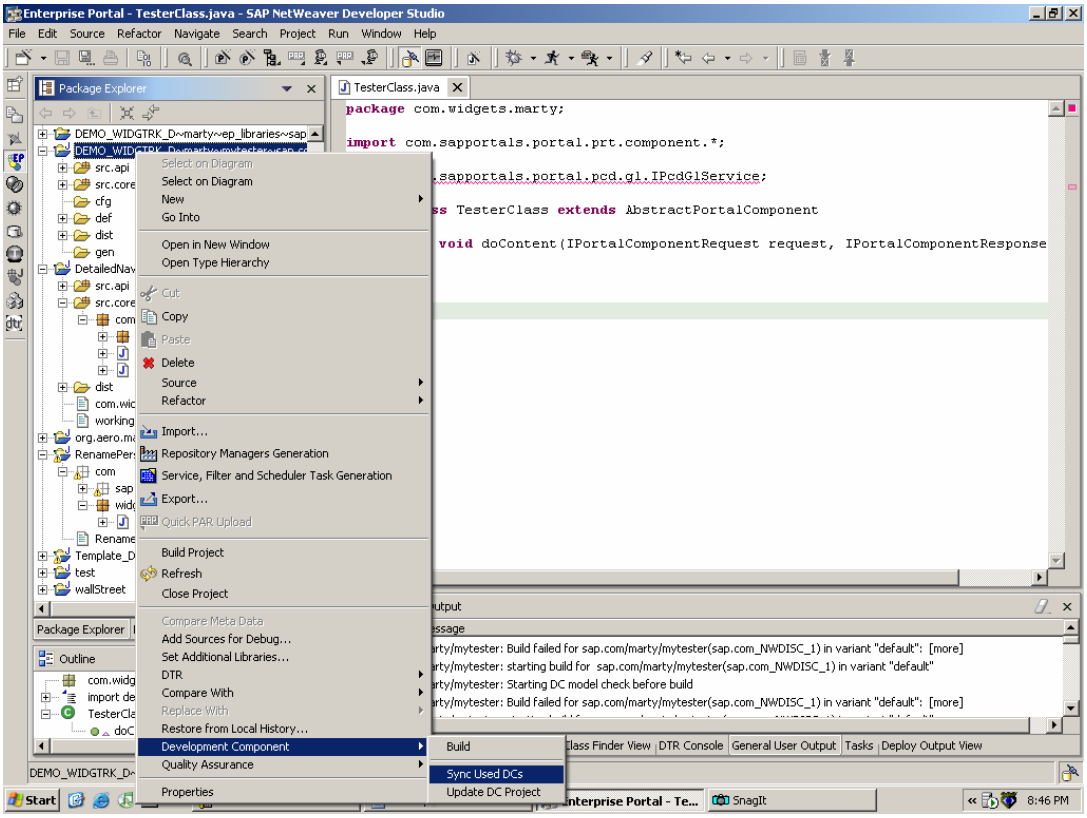
In the “Add Dependency” window--if you’re developing portal components--you’ll only need to check the “Build Time” check box.



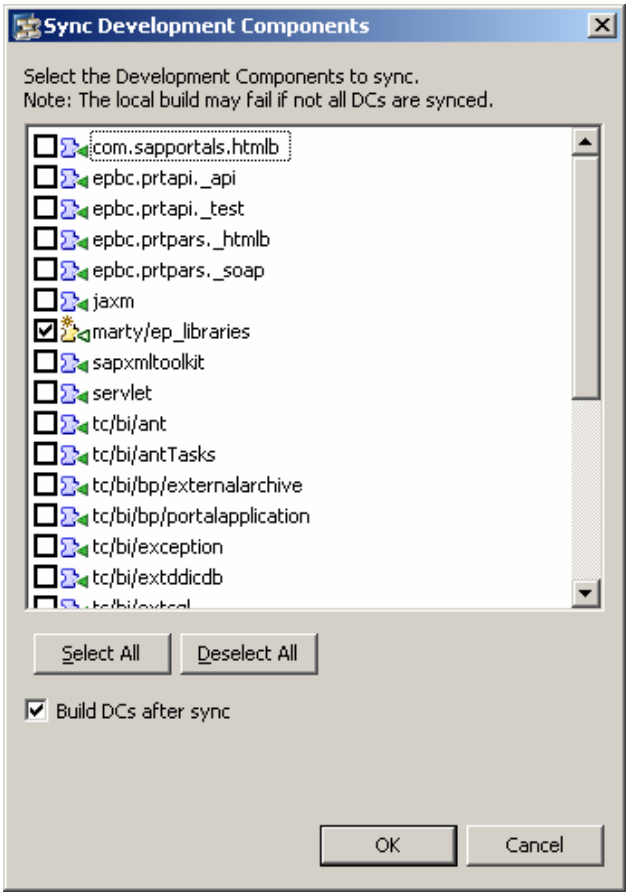
Choose the DC project that contains the portal libraries.



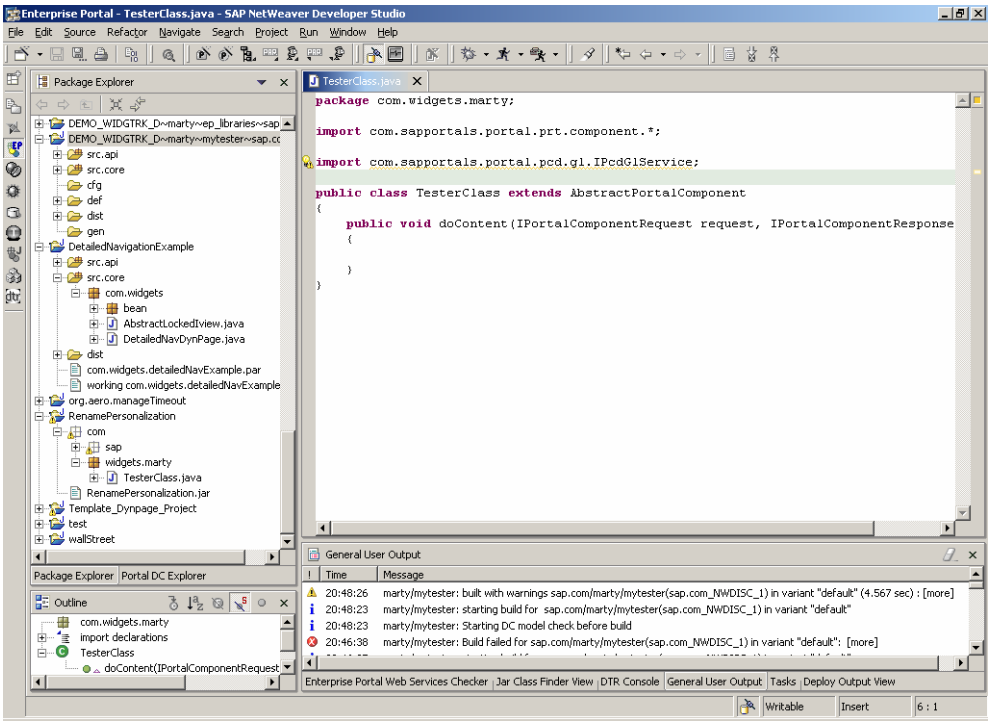
Browse back to the EP perspective and right click on your project. Choose “Development Component”→”Sync Used DCs” from the menu.



When given the option dialog, choose the required DC's that you want to sync. You should select your library project in this window.



You're import statement will now resolve!!



DON'T FORGET RUNTIME REFERENCES IF REQUIRED!

In this particular example, the IPcdGlService also requires a runtime reference as well. This is created in the portalapp.xml under the projects dist/PORTAL-INF folder by using the SharingReference parameter in the application-config tag of the component.

```
<application-config>
  <property name="SharingReference" value="com.sap.portal.pcd.glservice"/>
</application-config>
```

Additional Libraries

If you wish to add additional jar files, you can repeat this same procedure and use the same Development Component. The advantage to use individual public parts is that the component will only need to build the dependent components at compile time, not multiple files that its not using.

Disclaimer & Liability Notice

This document may discuss sample coding, which does not include official interfaces and therefore is not supported. Changes made based on this information are not supported and can be overwritten during an upgrade.

SAP will not be held liable for any damages caused by using or misusing of the code and methods suggested here, and anyone using these methods, is doing it under his/her own responsibility.

SAP offers no guarantees and assumes no responsibility or liability of any type with respect to the content of the technical article, including any liability resulting from incompatibility between the content of the technical article and the materials and services offered by SAP. You agree that you will not hold SAP responsible or liable with respect to the content of the Technical Article or seek to do so.