

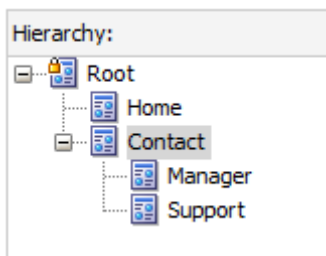
Filter resources

When you create a navigation model or resource catalog, you often want to limit the resources based upon some conditions. You could do this by creating multiple resource catalogs or navigation models or by creating a filter that will be applied to your navigation model or resource catalog.

In this blog post I will explain how to use a filter to limit the resources in a navigation or RC.

You can download the example [here](#).

I created a very simple navigation model with some additional pages as shown in this image:



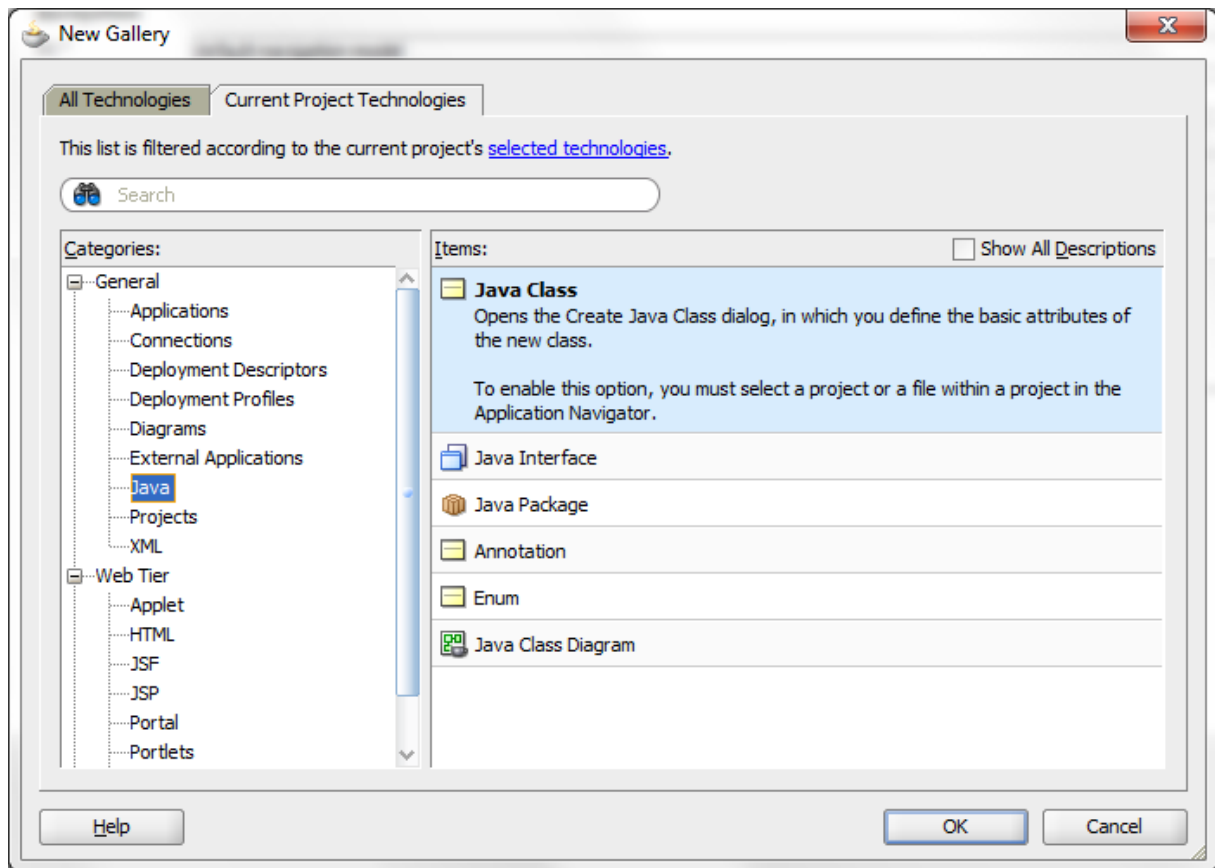
I also added to additional users to the application:

- support (pwd: support123)
- manager (pwd: manager123)

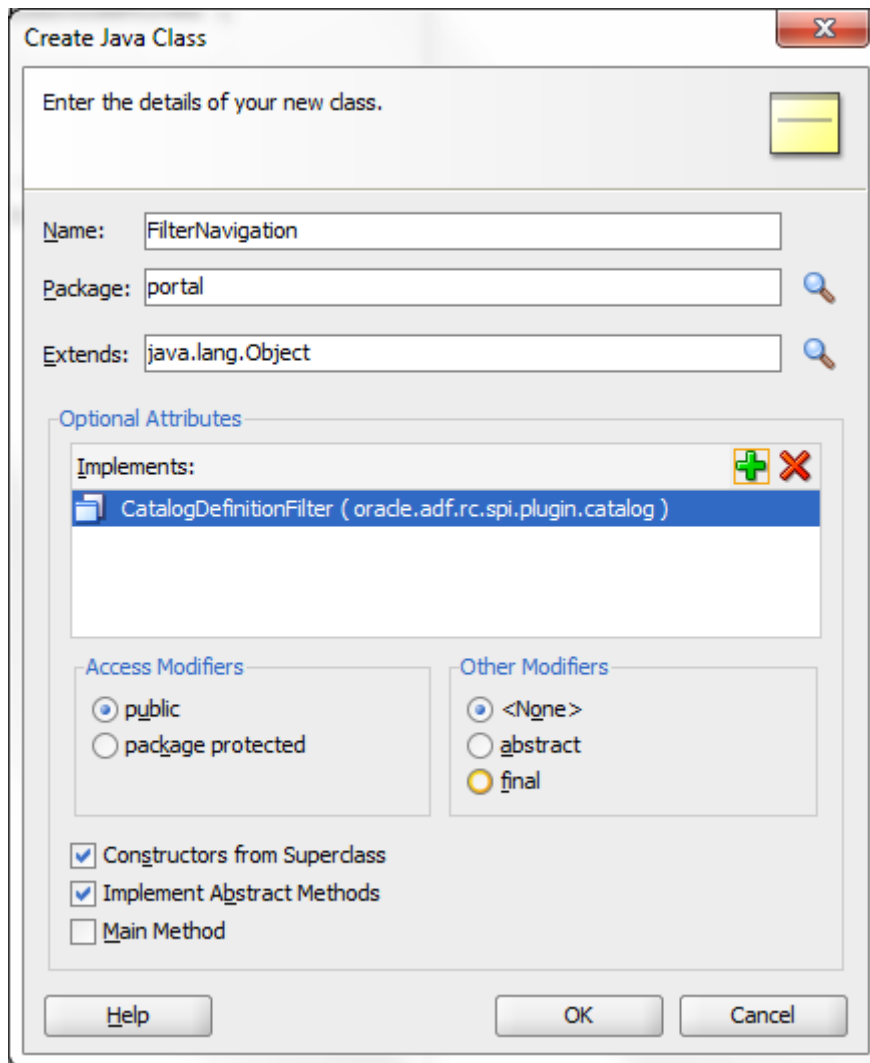
We will now create a filter so when the support user logs in, he does not see the manager contact page. The same for the manager user but then he only sees the manager page. Of course this can easily be done by configuring the proper security in the page hierarchy. I just selected this case to show how to implement a filter.

In order to create a filter, we need to create a java class that implements the `CatalogDefinitionFilter` interface.

Select the portal project in JDeveloper and press `ctrl+N` to open the new gallery. Select Java, Java Class and press OK.



Name the class FilterNavigation. Add a class in the Implements section:
oracle.adf.rc.spi.plugin.catalog.CatalogDefinitionFilter
Press ok.



When you open the source code of the FilterNavigation java class, you should see the implemented method:

```
public boolean includeInCatalog(CatalogElement catalogElement, Hashtable hashtable) {  
    return false;  
}
```

The includeInCatalog method will be called for every resource in your navigation or RC. When this method returns false, the resource will not show. This way you can easily filter the resources. The CatalogElement object contains all the metadata of the resource so you can easily identify the object.

When we implement the includeInCatalog as described above, it will look something like this:

```
public boolean includeInCatalog(CatalogElement catalogElement,Hashtable hashtable) {  
    ADFContext context = ADFContext.getCurrent();  
    String user = context.getSecurityContext().getUserName();  
    boolean isManager = user.equals("manager");
```

```

boolean isSupport = user.equals("support");

if(catalogElement.getId().equals("Support") && isManager)

    return false;

if(catalogElement.getId().equals("Manager") && isSupport)

    return false;

return true;
}

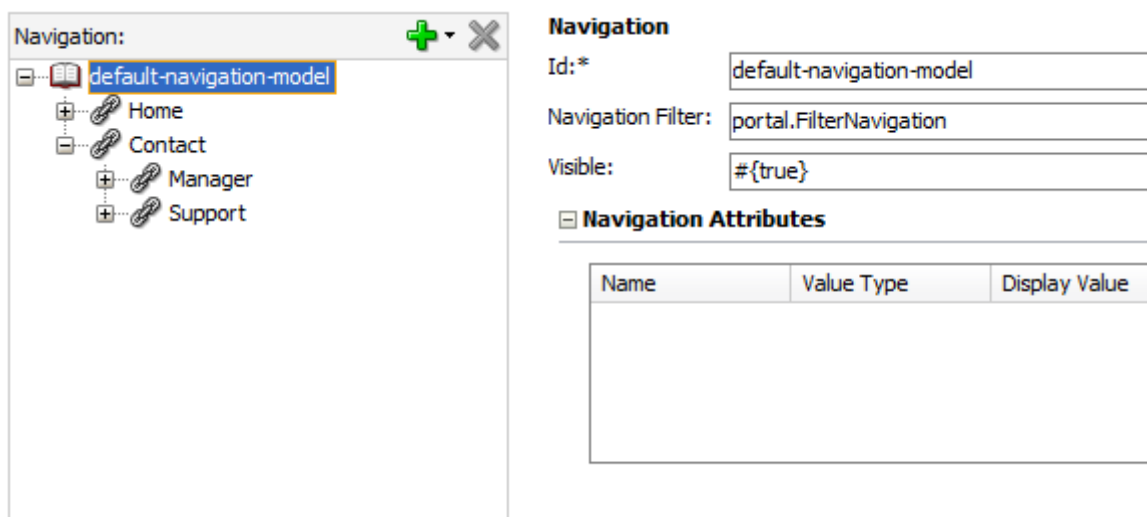
```

As you can see we request the current user and check for a specific resource based upon the id. The getId property of the catalogElement returns the Id we have defined in the navigation model or RC for the resource.

You also need to make sure return a default value like the last line in the above snippet. This is because each and every resource will call this method so you need to provide a true or false for each resource. That's why we return true at the end because every other resource can be added to the navigation or RC.

After we have created the filter we need to apply this to the navigation model or RC. In our example we will apply this to the navigation model so open the default-navigation-model.xml.

Select the root node (default-navigation-mode). In the Navigation Filter field provide the class we have just created. When you have followed the instruction, this should be portal.FilterNavigation.



Navigation

Id: * default-navigation-model

Navigation Filter: portal.FilterNavigation

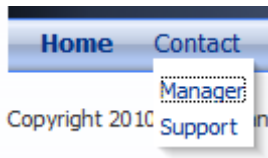
Visible: #{true}

Navigation Attributes

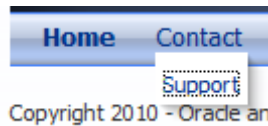
Name	Value Type	Display Value
------	------------	---------------

Now we can run the portal.

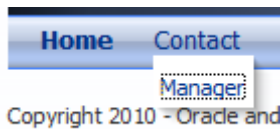
When you don't login you will see both pages:



When we log in with support/support123 we only see the Support page as a child of the contact page:



When you login with manager/manager123 we only see the Manager page as a child of the contact page:



This is how you can filter resources from both a navigation model or resource catalog. The filter from a resource catalog is the exact same interface so you could use this filter for a resource catalog as well. The only requirement is that you also have a resource with the id Manager and Support in your resource catalog if you want to apply this filter.

You can write whatever custom code in the filter. This is a very powerful feature but you have notice that it is not the most performing way of filtering. This is because each and every resource will call the filter so if you write complex code in the filter, this can take a while because it will be executed for each resource...

Beside using a filter, you can also use expression language in the visible attribute of a resource. This is very useful if you have only a few resources that need some additional logic. This way of filtering is faster because the expression language will only be executed for the element you have defined it for.

We could also use the case explained in this example to show the visible expression language. In this example, the business logic is quiet easy. The support page should not be shown when the user is manager and the manager page should not be visible for the support user.

In order for this to work, remove the Navigation filter from the default navigation model. Select the Manager node from the navigation model and specify following EL expression for the visible attribute:

```
#securityContext.userName != 'support'}
```

Do the same for the Support node but replace 'support' by 'manager':

Navigation:

default-navigation-model

Home

Contact

Manager

Support

Link

Id: *

Support

Type:

Page

Factory Class: *

oracle.webcenter.portalframework.sitestru...rc.AdfPageResourceFactory

URL: *

page://oracle/webcenter/portalapp/pages/Support.jspx

Render URL in Page Template:

Default Page Template

Redirect to URL

Visible:

#

{securityContext.userName != 'manager'}

When you now log in with the support or manager user, you should see the same behavior as when using the filter.

This shows that there are multiple ways of filtering. It all depends on the requirements and common sense. When you have a navigation model with lots of resources and they all require logic to be filtered than you best use a filter so you can group the filtering logic together.

When you have a large model or RC and only a few resources need to have filter logic, than you should use the visible attribute and write the logic in a managed bean.