Enabling and Disabling ADF Components

[Guest Author](https://blogs.oracle.com/author/guest-contributor) 

The purpose of this is to show different alternatives to enable and disable ADF components. There are 3 separate sample workspaces (11.1.1.2.0), one for each one of the ways to enable and disable ADF components that will be shown.

**1. Using isDisabled() and setDisabled() methods on the managed bean.**

In this example, the first af:commandButton (cb1) calls a method in the backing bean which enables/disabled the other af:commandButton:

<af:commandButton text="Press Me!" binding="#{backing\_test1.cb1}"

                  id="cb1" action="#{backing\_test1.toggleComponent}"/>

<br/>

<af:goLink text="goLink 1" binding="#{backing\_test1.gl1}" id="gl1"

           destination="http://www.oracle.com"/>

This is the method used in the backing bean which simply uses the isDisabled() and setDisabled() methods of the second button:

public void toggleComponent() {

    if (gl1.isDisabled()) {

        gl1.setDisabled(false);

        gl1.setText("This goLink is now enabled");

    } else {

        gl1.setDisabled(true);

        gl1.setText("This goLink is now disabled");

    }

}

Further information about the isDisabled() and setDisabled() methods (that are also applicable to other ADF components) can be seen at the [Java API Reference for Oracle ADF Faces](http://download.oracle.com/docs/cd/E12839_01/apirefs.1111/e10684/oracle/adf/view/rich/component/rich/nav/RichCommandButton.html).

Download sample workspace [here](http://blogs.oracle.com/csoto/resource/2010-05-17/ToggleNoEL.zip).

**2. Using expression language to check if the component is disabled or not.**

As in the previous example, the first af:commandButton (cb1) calls a method in the backing bean which enables/disabled the other af:commandButton:

<af:commandButton text="Press Me!" binding="#{backing\_test1.cb1}"

                  id="cb1"

                  actionListener="#{backing\_test1.toggleComponent}"/>

<br/>

<af:inputDate label="Date:" binding="#{backing\_test1.id1}" id="id1"/>

This is the method used in the backing bean which uses Expression Language to check if the second button is disabled or not:

public void toggleComponent(ActionEvent evento) {

    FacesContext fctx = FacesContext.getCurrentInstance();

    ELContext elctx = fctx.getELContext();

    Application jsfApp = fctx.getApplication();

    //create a ValueExpression that points to the ADF binding layer

    ExpressionFactory exprFactory = jsfApp.getExpressionFactory();

    ValueExpression valueExpr = exprFactory.createValueExpression(

                                 elctx,

                                 "#{!backing\_test1.id1.disabled}",

                                  Object.class

                                 );

    id1.setDisabled(Boolean.parseBoolean(valueExpr.getValue(elctx).toString()));

}

Further information about  
Expression Language Techniques can be seen in [this  
Advanced Expression Language Techniques OTN article](http://www.oracle.com/technology/products/jdev/tips/fnimphius/advancedeltechniques/advancedeltechniques.html).

Download sample workspace [here](http://blogs.oracle.com/csoto/resource/2010-05-17/ToggleELCTX.zip).

**3. Using expression language on the Disabled property of the ADF component on the JSPX.**

In this example the managed bean method to disable the second button is also called from the first button, but the second button has its disabled property set on the JSPX to take the EL value from the managed bean attribute:

<af:commandButton text="Click me" id="cb1"

                  actionListener="#{ToggleTestBk.toggleComponent}"/>

<br/>

<af:commandButton text="commandButton 2" id="cb2"

                  disabled="#{ToggleTestBk.isButtonEnabled}"

                  partialSubmit="true"/>

The managed bean method simply uses a private Boolean attribute which indicates whether thebutton is enabled or not and changes the value every time is invoked.

private boolean isButtonEnabled;

...

public void toggleComponent(ActionEvent event) {

    try{

        this.setIsButtonEnabled(!isButtonEnabled);

    }

    catch(Exception ex){

      System.out.println("Exception in toggleButton: " + ex);

    }

}

In the downloadable sample workspace provided for this example, the managed bean scope configured for the managed bean is application, but the application works with session scope as well. Further information about the 4 possible values for the managed bean scope element can be  
seen in [this  
JSF Managed Bean Facility OTN article](http://www.oracle.com/technology/tech/java/newsletter/articles/jsf_pojo/index.html).

Download sample workspace [here](http://blogs.oracle.com/csoto/resource/2010-05-17/ToggleELJSPX.zip).

On the first workspace a goLink is enabled and disabled, on the second one an inputDate is used, and on the third one a commandButton. There are many other ADF components that have the Disabled property.

Three approaches, one similar behaviour. Obviously, you can figure out many other ways to get this enable/disable functionality on ADF components.  It's up to you to decide which alternative is best for you.

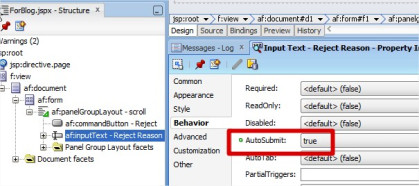
## Automatic Disable / Enable A CommandButon Base On InputText Field

July 2, 2014 by [Arik Lalo](https://ariklalo.com/author/ariklalo/" \o "Posts by Arik Lalo)

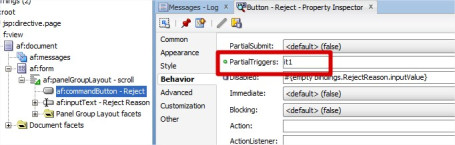
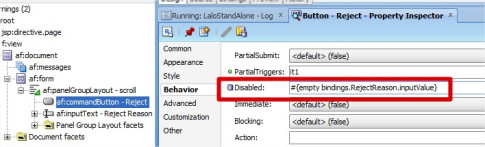
Here is a small example of a way to enable or disable a command button using a standard functionality of an ADF.  
Take for example the next scenario: You want to reject a form, and you must enter a reject reason.

For that we will use a commandButton (for the reject action) and an inputText (for the reject reason).  
Change both objects’ attributes as follows:

**inputText:**

* Set the **AutoSubmit** property to ‘true’:[](https://ariklalo.files.wordpress.com/2014/07/autosubmit.jpg)

**CommandButton:**

* Set the **PartialTriggers** property to the Reject InputTextId Id:[](https://ariklalo.files.wordpress.com/2014/07/partialtriggers.jpg)
* Set the **Disabled** property to have a condition base on the InputText value:  
  In my example: **#{empty bindings.RejectReason.inputValue}**[](https://ariklalo.files.wordpress.com/2014/07/reject-reason.jpg)

Run the page and test it.

Good Luck!

# [How to hide/show adf component automatically](https://community.oracle.com/message/13319797#13319797)

Hello,

Are you still getting method not found?

is your valueChangeListener being executed?

Try to use this valueChangeListener:

public void socVCL(ValueChangeEvent valueChangeEvent) {

           Object newValue = valueChangeEvent.getNewValue();

           if (newValue.equals("2")){

              RichSelectOneChoice sl = (RichSelectOneChoice)JSFUtils.findComponentInRoot("soc3");

               sl.setVisible(true);

               AdfFacesContext.getCurrentInstance().addPartialTarget(sl);

           }

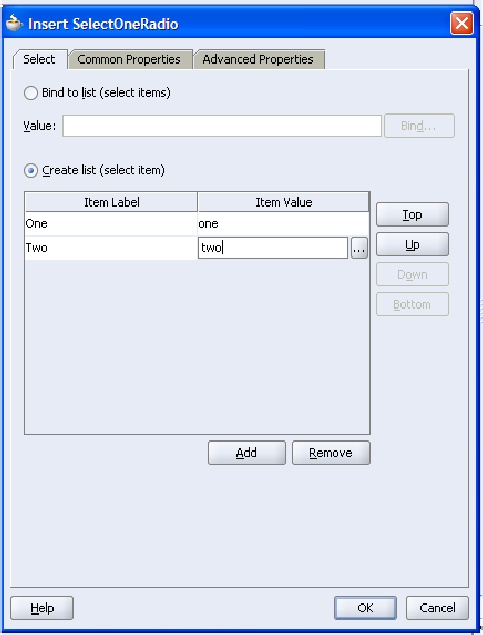
       }

regards,

Ruben.

visible="#{binding.filedName.inputValue != null}"

In this example we will show/hide a command button based on the two radio buttonsselections.1.Create a ShowHide.jspx page with name and create the managed bean for the page.2.Add the panelPage to the page.3.Select the SelectOneRadio button from the component palette and drop it on tothe panelPage.4.In the Insert SelectOneRadio button, select the “Create list “radio button.5.Click on add button to add an item into the list with Item label=’One’ and itemvalue=’one’.6.Again click on add button to add an item into the list with Item label=’Two’ anditem value=’two’.



7.Add a variable “private String radioBtnValue;” inside the managed bean of the page and generate the getter and setter methods for the variable.8.Set the value property of the selectOneRadio item to the#{backing\_ShowHide.radioBtnValue}.It means that the value of the selected radio button will be assigned to theradioBtnValue.9.Set the autosubmit property of the selectOneRadio item to true.10.Set the Id of the selectOneRadio button as “RadioBtn”11.Set the partialTriggers property of the af:PanelPage to RadioBtn.12.Add a command button to the panelPage.13.Select the Rendered property of the command button and click on the “Bind toData” icon on top of the property inspector window.14.Under the Expression section of the Rendered window set the EL as =#{backing\_ShowHide.radioBtnValue =='one'}This EL expression will be evaluated to either true or false based on whichcommand button rendered property is set.’15.Run your page

### How to hide “Advance” button and “Saved Search” dropdown in query panel in UI layer

This blog we will see how to hide the “Advance” button and “Saved Search” dropdown in query panel. The below screen is the query panel   
                               
  
 For hiding the “Advance” button we need to set **modeChangeVisible="false"** and for hiding the “Saved Search” dropdown, we need to set **saveQueryMode="hidden"** in af:query component. The below af:query code is  
  
<af:query id="qryId1" headerText="Search" disclosed="true"             
value="#{bindings.EmployeesVOCriteriaQuery.queryDescriptor}"   
model="#{bindings.EmployeesVOCriteriaQuery.queryModel}"   
queryListener="#{bindings.EmployeesVOCriteriaQuery.processQuery}" queryOperationListener="#{bindings.EmployeesVOCriteriaQuery.processQueryOperation}"  resultComponentId="::resId1" binding="#{backingBeanScope.backing\_QueryPanel.qryId1}" styleClass="AFQueryHideAddFields" modeChangeVisible="false" saveQueryMode="hidden"/>