

## 58. How-to use the optimized component search in Oracle ADF Faces

**ORACLE**

**CODE CORNER**



[twitter.com/adfcodecorner](https://twitter.com/adfcodecorner)

### **Abstract:**

How do you search for a component on a view? Calling `findComponent` on a parent container or the view root ? I think that most of us are familiar with this technique for finding components by their Id.

However, there are optimized options available that not only search for a component or a set of components but also perform operations on the component, which may be security related or just related to the context of the current business. This article introduces the `invokeOnComponent` and `visitTree` methods that you can use with ADF Faces components to optimize your search in ADF views and page fragments.

**Author:**

Frank Nimphius, Oracle Corporation  
[twitter.com/fnimphiu](https://twitter.com/fnimphiu)  
30-SEP-2010

*Oracle ADF Code Corner is a loose blog-style series of how-to documents that provide solutions to real world coding problems.*

*Disclaimer: All samples are provided as is with no guarantee for future upgrades or error correction. No support can be given through Oracle customer support.*

*Please post questions or report problems related to the samples in this series on the OTN forum for Oracle JDeveloper: <http://forums.oracle.com/forums/forum.jspa?forumID=83>*

## Introduction

There motivation of developers to parse a view's component tree differ. In the example shown in the image below, components within a specific path of a view hierarchy are printed into a table layout, showing the component id, client id, implementation class and the number of children and facets. To make it more interesting: components are checked if they are naming container, and if they are, printed with a yellow background color.

Id	Class Name	Client Id	Child Count	Facet Count
sdl5	class oracle.adf.view.rich.component.rich.layout.RichShowDetailItem	sdl5	1	0
psl3	class oracle.adf.view.rich.component.rich.layout.RichPanelStretchLayout	psl3	0	1
ps1	class oracle.adf.view.rich.component.rich.layout.RichPanelSplitter	ps1	0	2
t2	class oracle.adf.view.rich.component.rich.data.RichTree	t2	0	1
ot8	class oracle.adf.view.rich.component.rich.output.RichOutputText	t2:ot8	0	0
pgl2	class oracle.adf.view.rich.component.rich.layout.RichPanelGroupLayout	pgl2	1	0
s4	class org.apache.myfaces.trinidad.component.UISwitcher	s4	0	3
pfl4	class oracle.adf.view.rich.component.rich.layout.RichPanelFormLayout	pfl4	5	1
it32	class oracle.adf.view.rich.component.rich.input.RichInputText	it32	0	0
it31	class oracle.adf.view.rich.component.rich.input.RichInputText	it31	0	0
it33	class oracle.adf.view.rich.component.rich.input.RichInputText	it33	0	0
it29	class oracle.adf.view.rich.component.rich.input.RichInputText	it29	0	0
it30	class oracle.adf.view.rich.component.rich.input.RichInputText	it30	0	0
cb4	class oracle.adf.view.rich.component.rich.nav.RichCommandButton	cb4	0	0
pfl2	class oracle.adf.view.rich.component.rich.layout.RichPanelFormLayout	pfl2	6	1
it13	class oracle.adf.view.rich.component.rich.input.RichInputText	it13	0	0
it18	class oracle.adf.view.rich.component.rich.input.RichInputText	it18	0	0
it15	class oracle.adf.view.rich.component.rich.input.RichInputText	it15	0	0

Iteratively calling `findComponent` on a parent component would work here as well, but possibly is harder to write and most likely takes longer in the execution than what the performance optimized `visitTree` method provides for MyFacesTrinidad and ADF Faces.

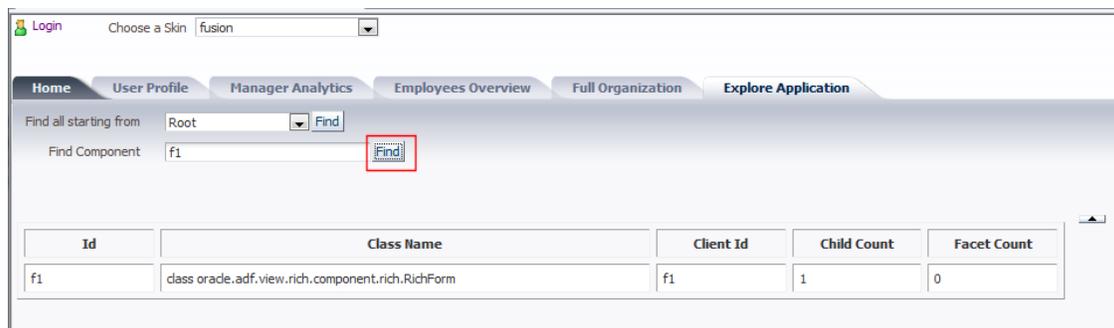
The `visitTree` method invokes a developer defined callback method for each component it finds, allowing developers to update components on the go. So based on a condition, like a business rule or security check, component states can be changed in a bulk update with only a few lines of code to write.

In addition to the `visitTree` method, an equivalent functionality exists on the JSF components for single updates: `invokeOnComponent`. Both methods are briefly explain in this article as a pointer for your own research into optimized component search and manipulation.

**Note:** This article consist mostly of the sample sources code. If you don't like reading sources codes and the comments therein, skip to the end of this paper to read about how to access and run the sample application.

**Note:** The sample you can download from the ADF Code Corner site also showcases ADF Security, skinning and tree/edit form synchronization. Its a sample that I use for ADF Insider recordings.

## Finding components in a view



The code shown below is taken from the "Find Component" search sample shown in the image above.

A single client component Id is passed to an action method to find the component for it. The search starts on the `UIViewRoot` component for a full page search but could also be invoked on any other JSF component. The difference between this search and `findComponent` is that a callback handler is passed to the method that performs an action on or reports information about the found component.

In this example, the component id, class and naming container information is read into a helper class, which is then used to display rows in a table.

**Note:** The helper class is listed later in this paper

```
/**
 * Finds a component identified by its id entered in a free text
 * field. The result is written to the output of a panel tab
 *
 * @param actionEvent
 */
public void onSearchSingleComponent(ActionEvent actionEvent) {
    //reset list
    resultComponentList = new ArrayList<UIComponentDef>();
    //find starting component
    FacesContext fctx = FacesContext.getCurrentInstance();
    UIViewRoot root = fctx.getViewRoot();
    root.invokeOnComponent(
        fctx, freeSearchComponentSelector,
        new ContextCallback() {
            public void invokeContextCallback(
                FacesContext facesContext,
                UIComponent uiComponent) {
                singleComponentSearchResult = uiComponent;
            }
        });
    if(singleComponentSearchResult != null){
        //describe the component that is found using a helper
        //object represneting the table row
        UIComponentDef uicompDef = new UIComponentDef();
        uicompDef.setComponentId(singleComponentSearchResult.getId());
        uicompDef.setClientId(singleComponentSearchResult.getClientId(
            FacesContext.getCurrentInstance()));
        uicompDef.setChildCount(singleComponentSearchResult
            .getChildCount());
        uicompDef.setFacetCount(singleComponentSearchResult
            .getFacetCount());
        uicompDef.setIsNamingContainer(
            singleComponentSearchResult instanceof NamingContainer);
        resultComponentList.add(uicompDef);
    }
}
```

After finding the component and passing the row object to an ArrayList – resultComponentList is of type ArrayList<UIComponentDef> - the iterator that is used to render the table is refreshed using PPR.

```

<af:panelGroupLayout id="pgl4" layout="scroll"
    partialTriggers="cb5 commandButton1"
    inlineStyle="height:500px;">
  <trh:tableLayout id="tl1" borderWidth="1" width="900px;"
    cellPadding="5" cellSpacing="5">
    <!--Building the table header -->
    <trh:rowLayout id="rl1" halign="left" valign="top">
      <trh:cellFormat id="cf1" header="true" width="50px;">
        <af:outputText value="Id" id="ot9"/>
      </trh:cellFormat>
      <trh:cellFormat id="cf2" header="true" width="200px;">
        <af:outputText value="Class Name" id="ot10"/>
      </trh:cellFormat>
      <trh:cellFormat id="cf3" header="true" width="50px;">
        <af:outputText value="Client Id" id="ot11"/>
      </trh:cellFormat>
      <trh:cellFormat id="cf4" header="true" width="50px;">
        <af:outputText value="Child Count" id="ot12"/>
      </trh:cellFormat>
      <trh:cellFormat id="cf5" header="true" width="50px;">
        <af:outputText value="Facet Count" id="ot13"/>
      </trh:cellFormat>
    </trh:rowLayout>
    <!--building the table -->
    <af:iterator id="i2" var="list"
      value="#{viewScope.PageExploreBean.resultComponentList}"
      rows="0">
      <trh:rowLayout id="rowLayout1" halign="left" valign="top"
        inlineStyle="#{list.isNamingContainer ?
          'background-color:yellow;' :
          'background-color:white;'}">
        <trh:cellFormat id="cellFormat1" width="50px;">
          <af:outputText id="outputText1"
            value="#{list.componentId}"/>
        </trh:cellFormat>
        <trh:cellFormat id="cellFormat2" width="200px;">
          <af:outputText id="outputText2"
            value="#{list.componentType}"/>
          ...
        </trh:cellFormat>
      </trh:rowLayout>
    </af:iterator>
  </trh:tableLayout>
</af:panelGroupLayout>

```

```
</trh:tableLayout>  
</af:panelGroupLayout>
```

Note the part printed in blue. The iterator stamps over rows to print the table cells. The iterator reads from the `ArrayList` of UI component definitions that are created by callback handler defined for the `invokeOnComponent` method call. The same table is used in the next section, when using the `visitTree` method to print a complete component hierarchy.

## Parsing a view using `visitTree`

The `visitTree` method is exposed on subclasses of `UIXComponent`, which are all Trinidad and ADF Faces components. This feature, at least until JSF 2.0, does not exist in the JSF standard.

In the sample shown in the image above, the user selects a starting point for searching the page hierarchy from the select one choice component. A helper class is used to store information about the components in the view. Instances of the helper class are added to an `ArrayList`, which then is used with an `af:iterator` to print the component tree.

If the component is a naming container, CSS is used to create a yellow background color. Instead of coloring the background, you can imagine that you could also manipulate the component instance.

**Note:** If you want to run this code in a phase listener then the "after INVOKE APPLICATION" is a good time for this.

**Note:** If you wanted to use this approach of parsing components on a page to apply security, then I suggest you do so in decorating the default view handler. This way changes are applied when the view renders allowing you to e.g. dynamically switch between secret fields and readable fields.

The helper class is defined as shown below:

```
public class UIComponentDef {  
    String componentId;  
    String componentType;  
    String clientId;  
    int facetCount = 0;  
    int childCount = 0;  
    boolean isNamingContainer = false;  
  
    public UIComponentDef()  
        super();  
    }  
  
    public void setComponentId(String componentId) {  
        this.componentId = componentId;  
    }  
  
    public String getComponentId() {  
        return componentId;  
    }  
}
```

```
public void setComponentType(String componentType) {
    this.componentType = componentType;
}

public String getComponentType() {
    return componentType;
}

public void setClientId(String clientId) {
    this.clientId = clientId;
}

public String getClientId() {
    return clientId;
}

public void setFacetCount(int facetCount) {
    this.facetCount = facetCount;
}

public int getFacetCount() {
    return facetCount;
}

public void setChildCount(int childCount) {
    this.childCount = childCount;
}

public int getChildCount() {
    return childCount;
}

public void setIsNamingContainer(boolean isNamingContainer) {
    this.isNamingContainer = isNamingContainer;
}

public boolean isIsNamingContainer() {
    return isNamingContainer;
}
}
```

When hitting the find button for the selectOneChoice component, the following managed bean method is invoked to parse the view and dump the component. Note that the managed bean is in view scope as it does not act as a backing bean but only creates and holds the List with the table data.

```
public void onFindAll(ActionEvent actionEvent) {
    //reset list
    resultComponentList = new ArrayList<UIComponentDef>();

    //find starting component
    FacesContext fctx = FacesContext.getCurrentInstance();
```

```
UIViewRoot root = fctx.getViewRoot();
//find the component to start the search on
root.invokeOnComponent(fctx,componentSelector,new ContextCallback(){
public void invokeContextCallback(FacesContext facesContext,
                                UIComponent uiComponent) {
    singleComponentSearchResult = uiComponent;
}
});

if(singleComponentSearchResult != null){
    UIXComponent start = (UIXComponent)singleComponentSearchResult;
    HashSet<VisitHint> hints = new HashSet<VisitHint>();
    //skip components not rendered on the page
    hints.add(VisitHint.SKIP_UNRENDERED);

    /*
    * Perform a tree visit starting at this node in the tree.
    * UIXComponent.visitTree() implementations do not invoke the
    * VisitCallback directly, but instead call
    * VisitContext.invokeVisitCallback() to invoke the
    * callback. This allows VisitContext implementations to provide
    * optimized tree traversals, for example by only calling the
    * VisitCallback for a subset of components. Note that for this
    * sample, the SampleVisitContext class is used, which performs a
    * full search for component Ids. If you need to optimize the
    * search, make sure you create a custom implementation of
    * VisitContext based on - but not extending - the Trinidad
    * PartialVisitContext class
    *
    * Parameters:
    *   visitContext - the VisitContext for this visit
    *   callback - the VisitCallback instance whose visit method will
    *               be called for each node visited.
    *
    * Returns:
    *   return true to indicate that the tree visit is complete (eg.
    *   all components that need to be visited have been visited).
    *   This results in the tree visit being short-circuited such that
    *   no more components are visited.
    */

    start.visitTree(new SampleVisitContext(fctx,hints,null),
                    new VisitCallback(){

    /*
    * VisitResult.ACCEPT
    *   This result indicates that the tree visit should descend
```



```
public SampleVisitContext(FacesContext fctx) {
    this(fctx, null, null);
}

/**
 * full view search
 * @param fctx FacesContext instance to search in
 * @param hints VisitHint list chosen from the VisitHint class
 * enumeration
 * EXECUTE_LIFECYCLE    Hint that indicates that the visit is being
 *                       performed as part of lifecycle phase
 *                       execution and as such phase-specific
 *                       actions (initialization) may be taken.
 * SKIP_TRANSIENT       Hint that indicates that only non-transient
 *                       subtrees should be visited.
 * SKIP_UNRENDERED      Hint that indicates that only the rendered
 *                       subtree should be visited.
 * @param phaseId
 */
public SampleVisitContext(FacesContext fctx,
    Set<VisitHint> hints, PhaseId phaseId) {
    if (fctx == null)
        throw new NullPointerException("FacesContext must not be
            null");

    if ((phaseId != null) && ((hints == null) ||
        (!hints.contains(VisitHint.EXECUTE_LIFECYCLE))))
        throw new IllegalArgumentException("Hints must contain
            VisitHint.EXECUTE_LIFECYCLE if phaseId is passed");
    _fctx = fctx;
    _phaseId = phaseId;

    EnumSet<VisitHint> hintsEnumSet = ((hints == null) ||
        (hints.isEmpty()))?
        EnumSet.noneOf(VisitHint.class): EnumSet.copyOf(hints);
    _hints = Collections.unmodifiableSet(hintsEnumSet);
}

public FacesContext getFacesContext() {
    return _fctx;
}

public PhaseId getPhaseId() {
    return _phaseId;
}
```

```

public Collection<String> getIdsToVisit() {
    //return all ids of a subtree of full view search
    return ALL_IDS;
}

public Collection<String> getSubtreeIdsToVisit(
    UIComponent uIComponent) {
    if (!(uIComponent instanceof NamingContainer))
    {
        throw new IllegalArgumentException("Component is not a
            NamingContainer: " + uIComponent);
    }
    return ALL_IDS;
}

public VisitResult invokeVisitCallback(UIComponent uIComponent,
    VisitCallback visitCallback)
{
    return visitCallback.visit(this, uIComponent);
}

public Set<VisitHint> getHints() {
    return _hints;
}
}

```

## Sample Download

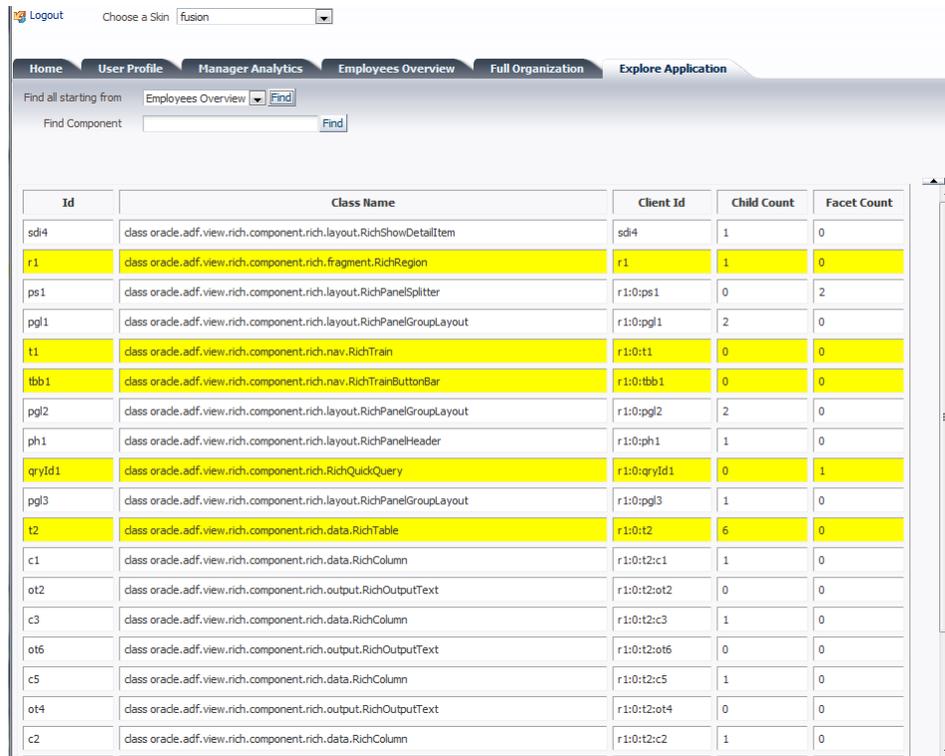
The Oracle JDeveloper 11.1.1.3 workspace with the sample application can be downloaded from the ADF Code Corner website:

<http://www.oracle.com/technetwork/developer-tools/adf/learnmore/index-101235.html>

You need to configure the model project to connect to the HR schema of your local database. When you run the application, you can directly select the "Explore Application" tab. Choose an entry in the select list to indicate where to start the search from. If you select "Employees Overview" then you see that only a few components are listed for this tab.

Id	Class Name	Client Id	Child Count	Facet Count
sdi4	class oracle.adf.view.rich.component.rich.layout.RichShowDetailItem	sdi4	1	0
r1	class oracle.adf.view.rich.component.rich.fragment.RichRegion	r1	0	0

This is because the main content is in an ADF region that requires authorization to display its content. Press the "login" link and authenticate as **sking/welcome1** and again try the page dump for the "Employee Overview". This time the content of the ADF region is displayed as well.



Id	Class Name	Client Id	Child Count	Facet Count
sdi4	class oracle.adf.view.rich.component.rich.layout.RichShowDetailItem	sdi4	1	0
r1	class oracle.adf.view.rich.component.rich.fragment.RichRegion	r1	1	0
ps1	class oracle.adf.view.rich.component.rich.layout.RichPanelSplitter	r1:0:ps1	0	2
pgl1	class oracle.adf.view.rich.component.rich.layout.RichPanelGroupLayout	r1:0:pgl1	2	0
t1	class oracle.adf.view.rich.component.rich.nav.RichTrain	r1:0:t1	0	0
tbb1	class oracle.adf.view.rich.component.rich.nav.RichTrainButtonBar	r1:0:tbb1	0	0
pgl2	class oracle.adf.view.rich.component.rich.layout.RichPanelGroupLayout	r1:0:pgl2	2	0
ph1	class oracle.adf.view.rich.component.rich.layout.RichPanelHeader	r1:0:ph1	1	0
qryId1	class oracle.adf.view.rich.component.rich.RichQuickQuery	r1:0:qryId1	0	1
pgl3	class oracle.adf.view.rich.component.rich.layout.RichPanelGroupLayout	r1:0:pgl3	1	0
t2	class oracle.adf.view.rich.component.rich.data.RichTable	r1:0:t2	6	0
c1	class oracle.adf.view.rich.component.rich.data.RichColumn	r1:0:t2:c1	1	0
ot2	class oracle.adf.view.rich.component.rich.output.RichOutputText	r1:0:t2:ot2	0	0
c3	class oracle.adf.view.rich.component.rich.data.RichColumn	r1:0:t2:c3	1	0
ot6	class oracle.adf.view.rich.component.rich.output.RichOutputText	r1:0:t2:ot6	0	0
c5	class oracle.adf.view.rich.component.rich.data.RichColumn	r1:0:t2:c5	1	0
ot4	class oracle.adf.view.rich.component.rich.output.RichOutputText	r1:0:t2:ot4	0	0
c2	class oracle.adf.view.rich.component.rich.data.RichColumn	r1:0:t2:c2	1	0

You can copy a clientId to the "Find Component" field and hit the "Find" button next to it. This then uses the invokeOnComponent method to find a specific component in a view.

**Note:** The select list and the "find component" field in the sample are not dependent

**Note:** This is a sample that did not handle exceptions for an invalid clientId entered in the search field. So don't try this ;-).

Also, the logout link does not work because of basic authentication being used. As soon as you log out, the browser re-authenticates the session. You will have to change authentication to form based to see the logout working.

**Note:** The sample application also demos the use of ADF Security and skinning. A second – less privileged – account is **ahunold/welcome1**. Try it out.

---

**RELATED DOCUMENTATION**

---

<input type="checkbox"/>	<a href="#">JSF UIComponent class documentation – invoke on component</a>
<input type="checkbox"/>	<a href="#">Trinidad – VisitResult class</a>
<input type="checkbox"/>	<a href="#">Trinidad – VisitContext abstract class</a>
<input type="checkbox"/>	<a href="#">Trinidad – VisitHint class doc</a>
<input type="checkbox"/>	<a href="#">Oracle Fusion Developer Guide – McGraw Hill Oracle Press, Frank Nimphius, Lynn Munsinger</a>