

ADF Code Corner

68. Solving the known range change event problem in ADF contextual events



twitter.com/adfcodecorner

Abstract:

Contextual events is an ADF binding feature that provides a public-subscribe communication channel for regions to interact with other regions or the parent view they reside in. Contextual events are usually invoked in response to a user action on a component, but may also be invoked from Java. Bug 10045872 reports a problem with row change events published as contextual events in response to users selecting a table row different from the current. While waiting for the bug fix, which may not be before Oracle JDeveloper 11.1.1.6, I describe the work around that also explains some aspects of contextual events.

Author:

Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
16-JAN-2011

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

The native functionality to invoke a contextual event in response to a row currency change in a table is defined on the ADF tree binding. Bug 10045872 reports a known issue with the default implementation, which prevents the event to be propagated to interested listeners. In this blog article I explain a work around that provides the same functionality for events invoked from the af:table component **SelectionListener** property.

To summarize the work around, you

- Create an **eventBinding** in the **PageDef** file of the ADF view that hosts the table for which you want to publish the row change event
- Set the eventBinding **Listener** property to the Apache Trinidad **SelectionListener** class
- Define a custom payload to pass the selected rowKey or a similar information to pass to the contextual event handling method
- Override the af:table **SelectListener** property value with a reference to a managed bean method handling the selection event
- In the managed bean method, first process the default ADF selection behavior, which is to set the selected row to be current in the ADF iterator
- Write some Java in the managed bean to invoke the **eventBinding** in response to a user changing the table row selection

The image below shows the sample application that you can download as Sample 68 from the ADF Code Corner website

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

The departments table is located in the parent page. The detail table resides in a bounded task flow that is added as an ADF region to the parent view. The parent view and the detail view in the ADF region don't share the same binding file – PageDef.xml – so that communication needs to be established using ADF region interaction. Contextual events is most generic and powerful option for establishing region interaction and can be used to notify the detail about the parent data row change.

DepartmentId	DepartmentName	ManagerId	LocationId
357	Sales II		1800
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	103	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700

EmployeeId	FirstName	LastName	Email	PhoneNumber
200	Jennifer	Whalen	JWHALEN	515.123.4444

Changing the selected row in the departments table sends an event to the ADF region with the information – the payload – about the new row key or, in this case, the new primary key to use for querying the detail data.

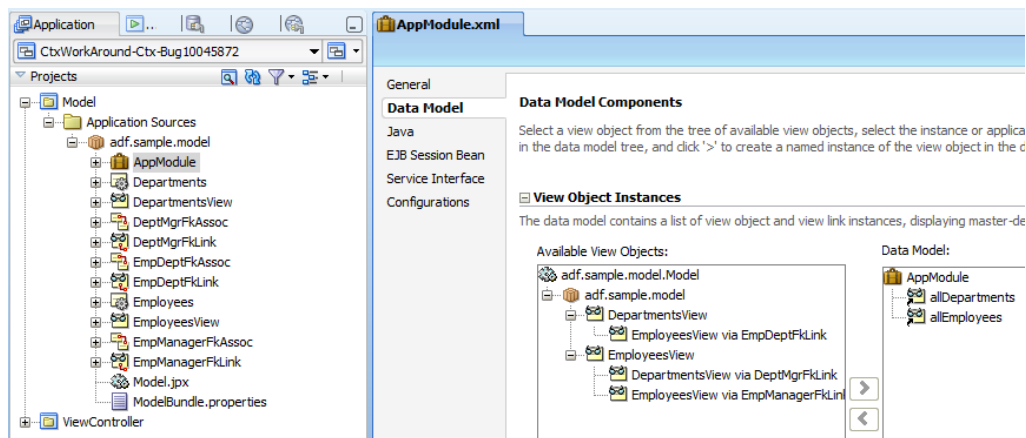
DepartmentId	DepartmentName	ManagerId	LocationId
357	Sales II		1800
10	Administration	200	1700
20	Marketing	201	1800
30	Purchasing	114	1700
40	Human Resources	203	2400
50	Shipping	103	1500
60	IT	103	1400
70	Public Relations	204	2700
80	Sales	145	2500
90	Executive	100	1700
100	Finance	108	1700
110	Accounting	205	1700

EmployeeId	FirstName	LastName	Email	PhoneNumber	HireDate
103	Alexander	Hunold	AHUNOLD	590.423.4567	1/3/2006
104	Bruce	Ernst	BERNST	590.423.4568	5/21/2006
105	David	Austin	DAUSTIN	590.423.4569	6/25/2006
106	Valli	Pataballa	VPATABAL	590.423.4560	2/5/2006
107	Diana	Lorentz	DLORENTZ	590.423.5567	2/7/2007

Note: The View Objects used in the example are independent. If the ADF region is not meant to be reusable and the business service is ADF Business Components, then the same functionality can be implemented using dependent View Objects and partial refresh, in which case the Data Control performs the refresh. However, this is another topic that is not in the subject of this article.

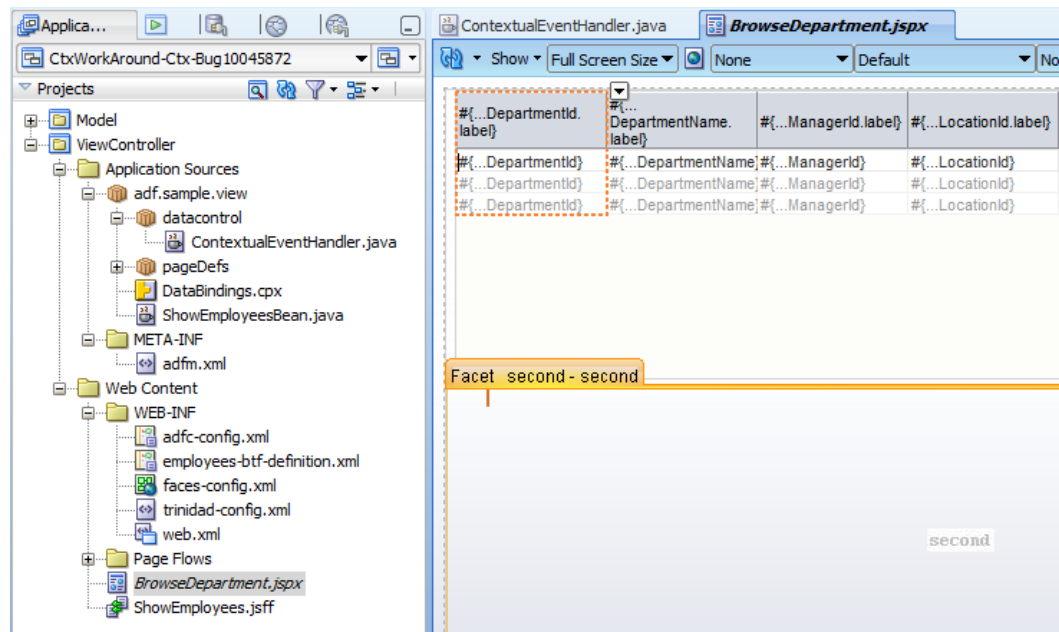
About the sample setup

The ADF Business Components model used in the sample consists of two independent View Objects as shown below. The **allEmployees** View Object instance has a **View Criteria** assigned that filters the table by the department Id.



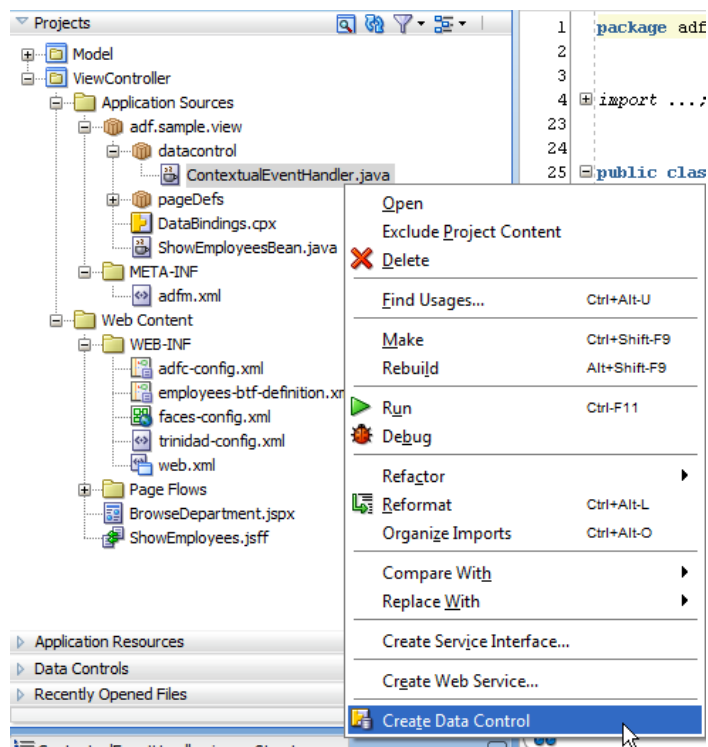
The **BrowseDepartments.jspx** page owns the departments table based on the **allDepartments** View Object. Further, the **ViewController** project contains

- A bounded task flow that holds the page fragment with the **allEmployees** View Object table. The default task flow activity is a method call activity that executes the **ExecuteWithParams** operation of the Employees View Object, passing the department Id of the parent table to the binding variable.
- A Java class to handle the contextual event in the **PageDef** file of the page fragment holding the employees table. The Java class is **ContextualEventHandler.java** and contains a single public method that accepts a single argument: the department Id. The **ContextualEventHandler.java** class is exposed through a JavaBean Data Control so the contained method can be configured as a method binding in the Pagedef file of the detail page fragment.
- A Java class **ShowEmployeesBean.java** is configured as a managed bean in the **employees-btf-definition.xml** and referenced from the employees table. The managed bean contains the setter / getter methods for the **af:table** component binding. The **ContextualEventHandler.java** class references the managed bean to pass the **department id** of the new selected parent table row to the bind variable, re-execute the iterator query and refresh the table.



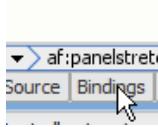
Handling the table row change as a contextual event

Contextual events use the ADF binding layer as a channel. To expose the event handler method in ADF, you need to expose it in a Data Control. In the example, **ContextualEventHandler.java** is exposed in a JavaBean Data Control. For this, I selected the **ContextualEventHandler.java** class and chose **Create Data Control** from the context menu (as shown in the image below)

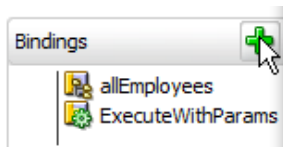


Note: The event handler is not specific for the work around. It is only needed in this sample to illustrate the work around.

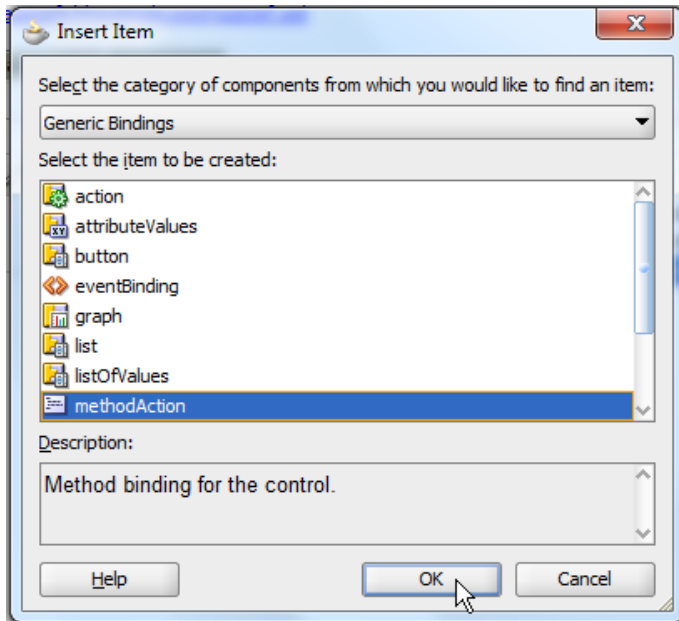
In the **ShowEmployees.jsff** page fragment, I pressed the **Bindings** tab to create a method binding that exposes the public event handler method.



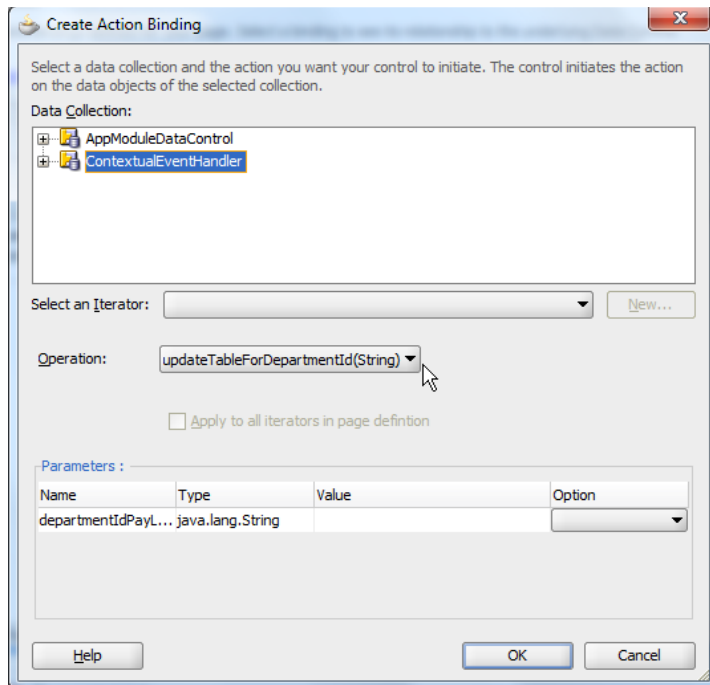
Pressing the **green plus icon** opens the ADF binding editor to create a method binding. Note that the **PageDef** file also has an operation binding entry for the **ExecuteWithParams** operation exposed on the Employees View Object.



This operation binding is called from the **ContextualEventHandler.java** class method to set the department Id to the bind variable and execute the iterator query

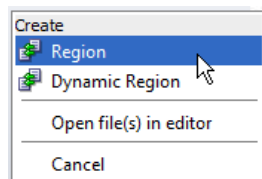
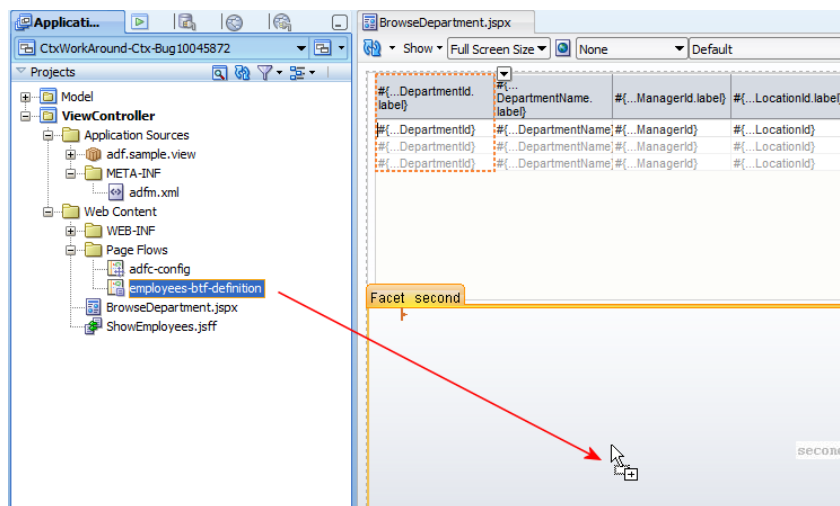


In the **Create Action Binding** dialog, I selected the **ContextualEventHandler** Data Control entry and the contained method in the **Operations** list.



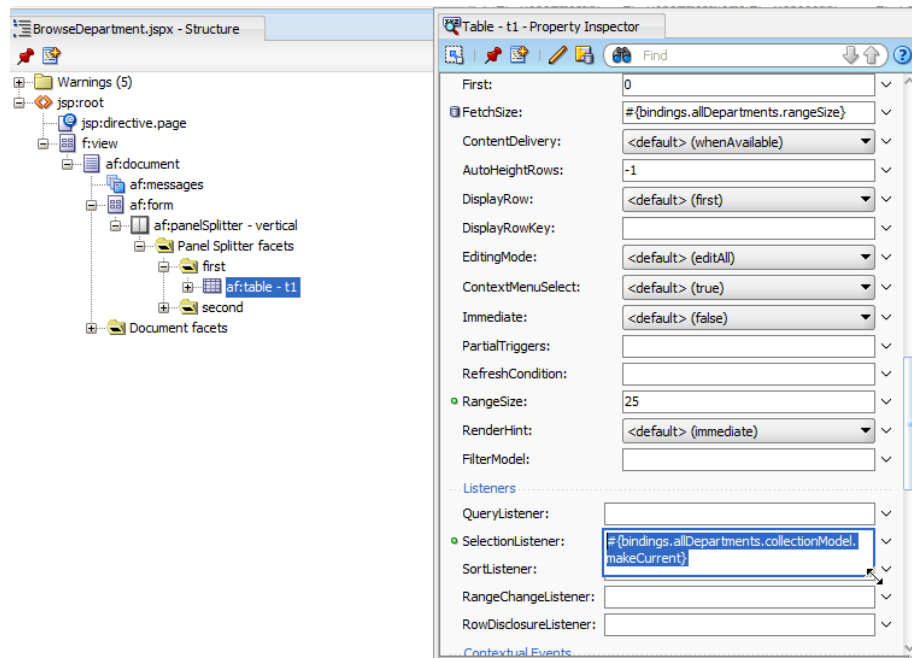
The method argument **departmentIdPayload** is exposed on the method binding but does not need to have a value defined in this dialog. The value is provided by mapping the producer event, the table select event, to the handled event, **updateTableForDepartmentId**.

When dragging the bounded task flow as an ADF region to the parent view, the **PageDef** file of the parent view will have a task flow binding added, a requirement for using contextual events.

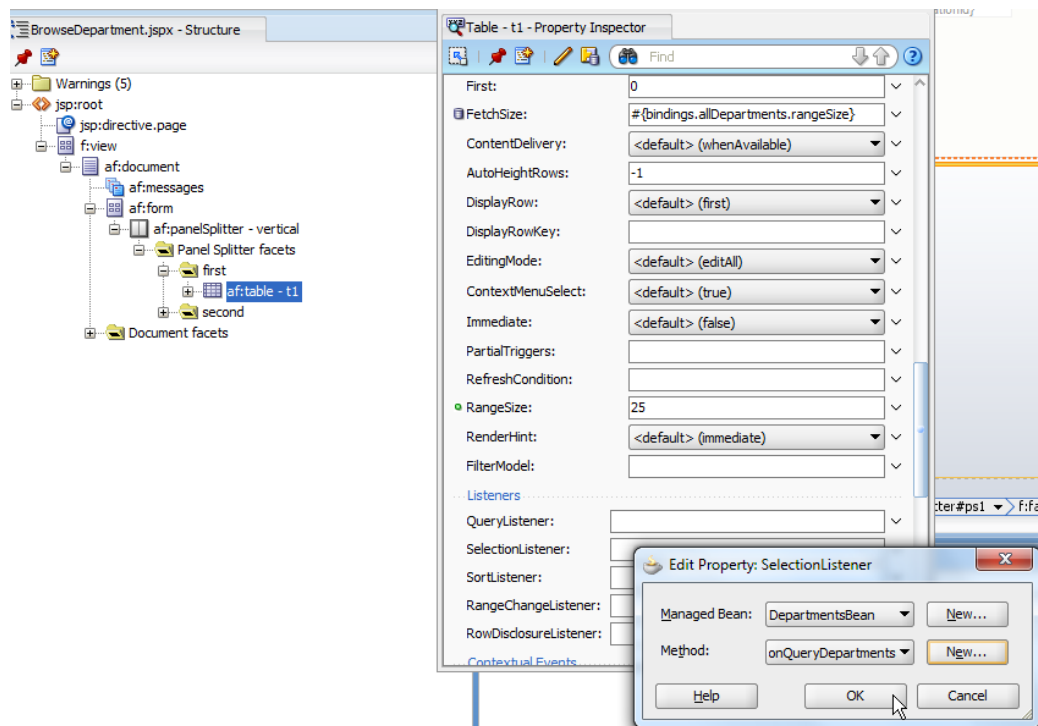


The departments table has a **SelectionListener** defined that uses Expression Language to synchronize the current row in the table with the iterator in the ADF binding (PageDef). To preserve the default

behavior, this EL command must be called before invoking the contextual event to propagate the selection change event. So I copied the existing EL value to the clipboard and ...



... pressed the **arrow icon** on the right. In the opened context menu I chose the **Edit** option to create a managed bean method to handle the **SelectEvent**.



The selection handler in the downloadable sample is defined in the **onQueryDepartments** method of the **DepartmentsBean**. When a user selects a table row, a select event is fired and passed to the managed bean.


```

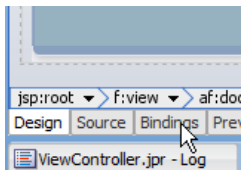
public void onQueryDepartments(SelectionEvent selectionEvent) {
    //preserve default Selection Event behavior
    String mexpr = "#{bindings.allDepartments.collectionModel.makeCurrent}";
    processMethodExpression(mexpr, selectionEvent, SelectionEvent.class);
}

```

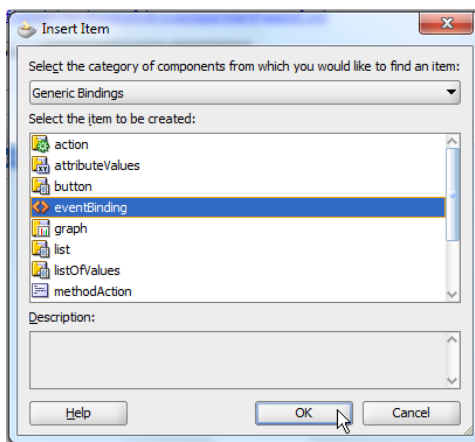
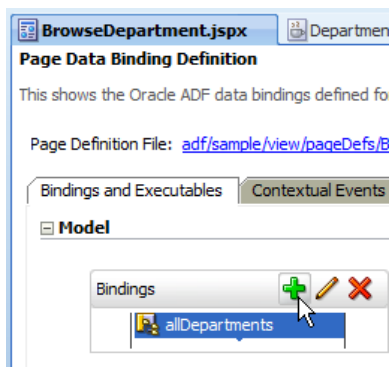
The code lines above ensure the default behavior, to synchronize the selected table row with the current row in the binding layer.

Note: The complete source code is listed below

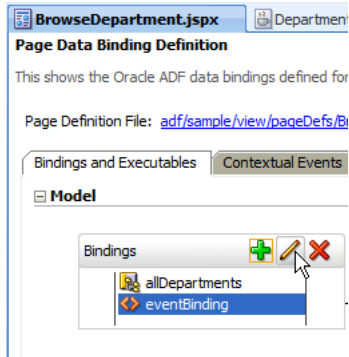
In the **BrowseDepartments.jspx** page, I clicked the **Bindings** tab to create a new **eventBinding**.



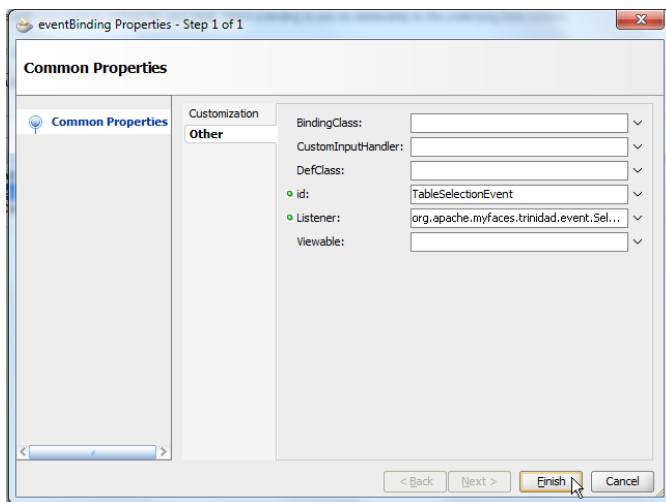
Event bindings in ADF are created for no-ADF bound component events. They are EL And Java accessible and defined under the **bindings** node in the PageDef file.



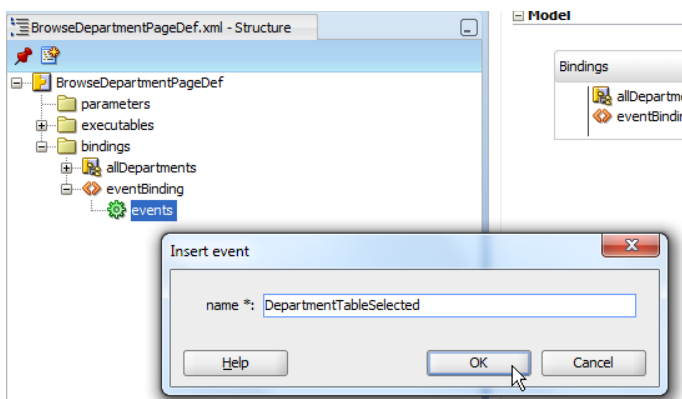
The **eventBinding** needs to be further configured with a unique **id** and a **Listener** that matches the **QueryListener**.



The **id** is used in Java to look up the event binding from the managed bean. The **Listener** type, `org.apache.mayfaces.trinidad.event.SelectListener`, is used to queue the table row selection event.

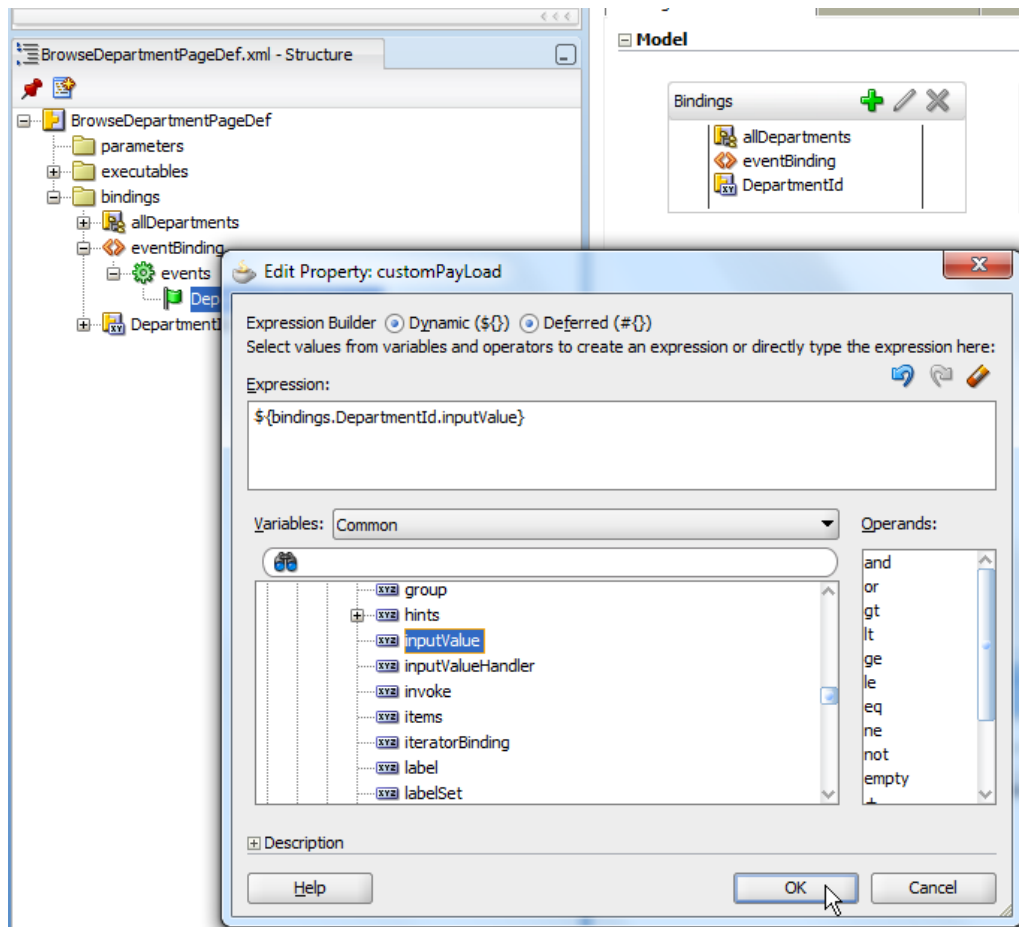


The **eventBinding** is only a container and needs to get two more elements added: **events** and **event**, as shown in the image below. The **events** node name is the name used when mapping the producer event (the Select Event) to the event handler.

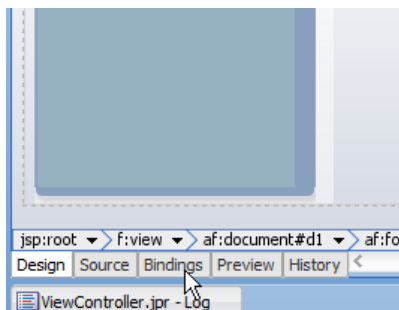


Finally, the **events** element is used to define the payload that should be passed to the event handler as an argument. In the example, I pass the selected table row's department id as an argument. For this I created

an attribute binding for the Departments View **DepartmentId** attribute. The binding layer takes care of the synchronization with the iterator that is bound to the departments table.

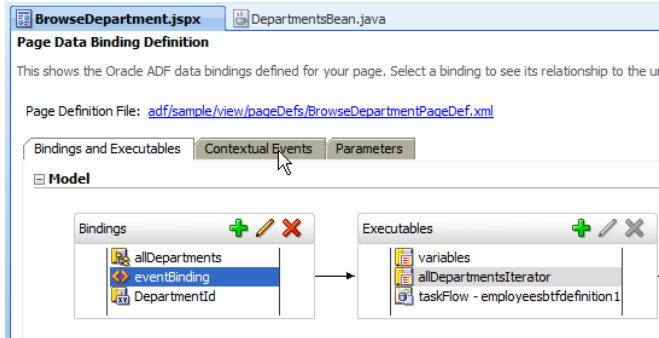


Important: Expression binding in the **PageDef** file use "\$" to eagerly resolve the expression. Custom payload references in contextual events **must** use eager fetching of the EL value. Otherwise the expression itself is passed as string.

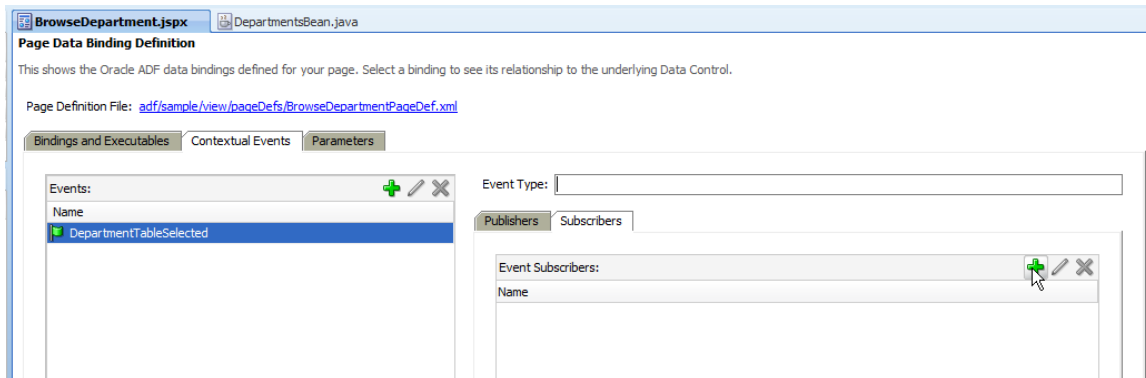


To this time, I created a producer method in the managed bean referenced from the departments table in the **BrowseDepartments.jspx** file, as well as the event handler exposed in the JavaBean Data Control. In the next step, I map the two in the **PageDef** file of the **BrowseDepartments.jspx** file.

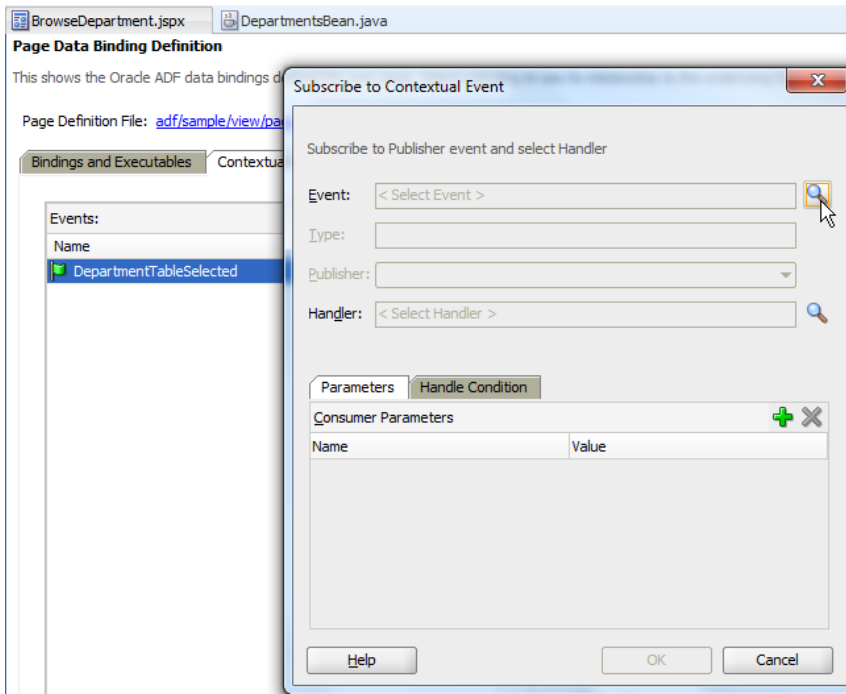
To access the event mapping dialog, I clicked the **Bindings** tab on the **BrowseDepartments.jspx** page.



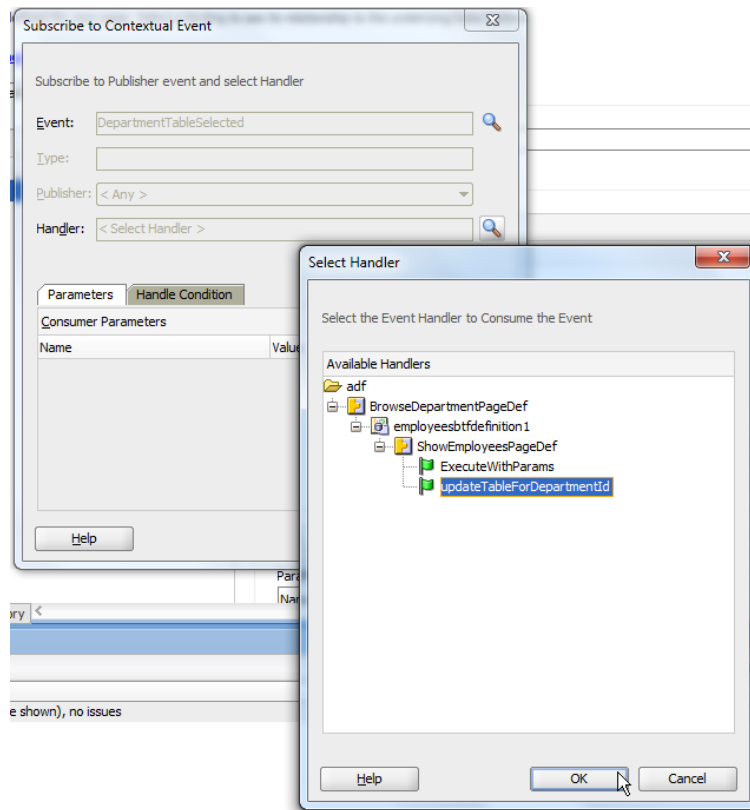
In the binding editor, the **Contextual Events** tab gives access to the event mapping editor, which is in the **Subscriber** tab of the contextual event dialog.



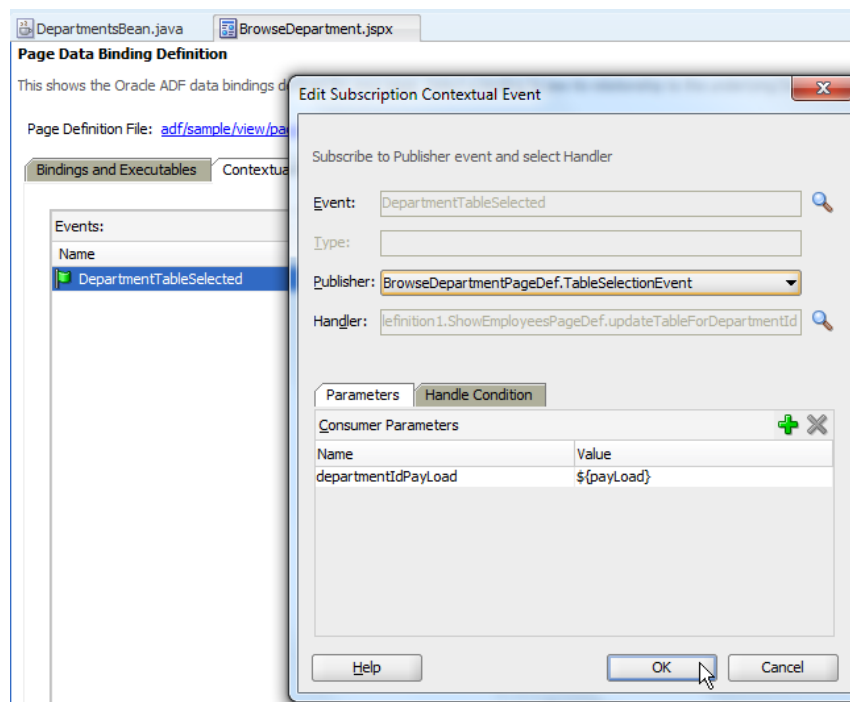
Pressing the **green plus** icon opens the dialog to map the producer event with the event handler.



Search buttons are available in the dialog to query the producer event (**Event**) and the event handler (**Handler**).



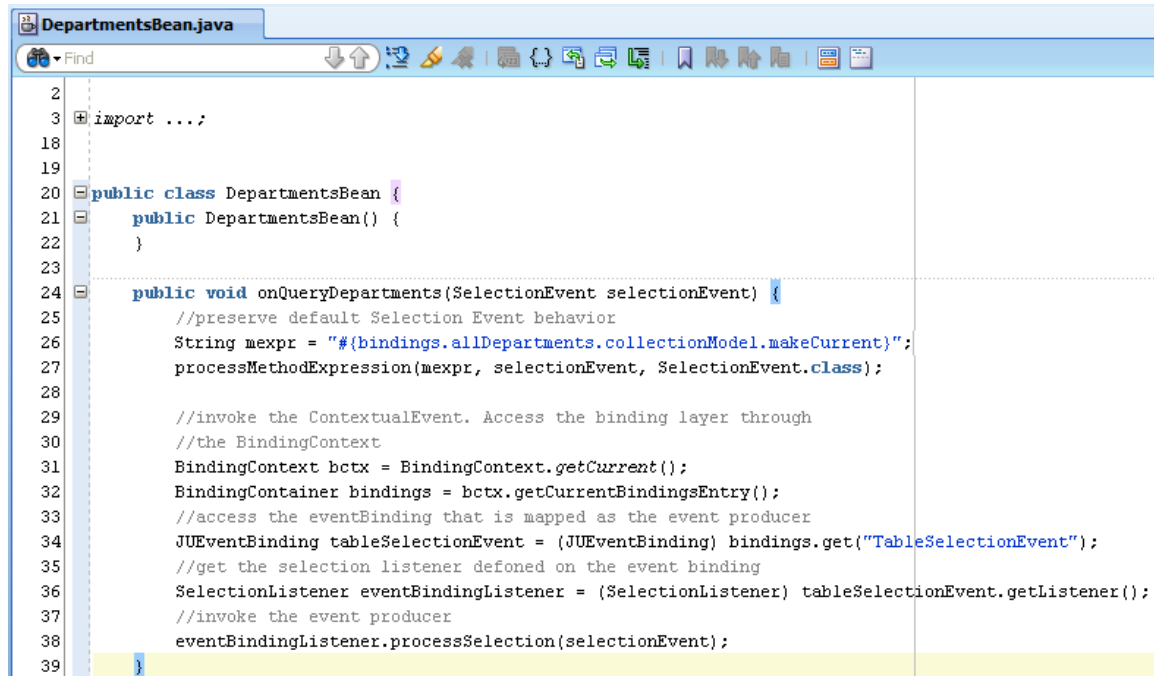
The producer event in this sample is the name of the **eventBinding** binding. The event binding is invoked from Java when the **SelectListener** is invoked in response to users selecting a new table row.



The **Parameters** tab defined the mapping of the **payload** object to the method arguments defined in the event handler method. In the sample, the only argument is the **departmentId**. The **custom payload** on

the **events** binding is defined to point to the **DepartmentId** attribute value, so no further mapping is required.

Note: The payload object is **always** referenced by the **#{payload}** EL reference. Also note the uppercase **L** in "payload", which is important for contextual events to pass the argument object.



```

2
3 import ...;
18
19
20 public class DepartmentsBean {
21     public DepartmentsBean() {
22     }
23
24     public void onQueryDepartments(SelectionEvent selectionEvent) {
25         //preserve default Selection Event behavior
26         String mexpr = "#{bindings.allDepartments.collectionModel.makeCurrent}";
27         processMethodExpression(mexpr, selectionEvent, SelectionEvent.class);
28
29         //invoke the ContextualEvent. Access the binding layer through
30         //the BindingContext
31         BindingContext bctx = BindingContext.getCurrent();
32         BindingContainer bindings = bctx.getCurrentBindingsEntry();
33         //access the eventBinding that is mapped as the event producer
34         JUEventBinding tableSelectionEvent = (JUEventBinding) bindings.get("TableSelectionEvent");
35         //get the selection listener defoned on the event binding
36         SelectionListener eventBindingListener = (SelectionListener) tableSelectionEvent.getListener();
37         //invoke the event producer
38         eventBindingListener.processSelection(selectionEvent);
39     }

```

The image above shows the completed SelectionEvent method in the managed bean. After setting the selected table row to become the current row in the binding layer of the **BrowseDepartments.jspx** page, it calls the **eventBinding** producer entry to invoke the contextual event. The QueryEvent that is passed into the managed bean method is provided as the argument to the **eventBinding** listener.

Source Code

The DepartmentsBean code contains the actual work around for bug 10045872 in that it invokes the **eventBinding** to produce the contextual event propagated to the ADF region containing the employees table.

The ContextualEventHandler class contains the event handler to update the detail based on the selection in the department table.

DepartmentsBean

```

import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.MethodExpression;
import javax.faces.application.Application;
import javax.faces.context.FacesContext;

import oracle.adf.model.BindingContext;
import oracle.binding.BindingContainer;

```

```
import oracle.jbo.uicli.binding.JUEventBinding;
import org.apache.myfaces.trinidad.event.SelectionEvent;
import org.apache.myfaces.trinidad.event.SelectionListener;

public class DepartmentsBean {

    public DepartmentsBean() {

    }

    public void onQueryDepartments(SelectionEvent selectionEvent) {
        //preserve default Selection Event behavior
        String mexpr =
            "#{bindings.allDepartments.collectionModel.makeCurrent}";
        processMethodExpression(mexpr, selectionEvent,
            SelectionEvent.class);
        //invoke the ContextualEvent. Access the binding layer through
        //the BindingContext
        BindingContext bctx = BindingContext.getCurrent();
        BindingContainer bindings = bctx.getCurrentBindingsEntry();
        //access the eventBinding that is mapped as the event producer
        JUEventBinding tableSelectionEvent =
            (JUEventBinding)bindings.get("TableSelectionEvent");
        //get the selection listener defoned on the event binding
        SelectionListener eventBindingListener =
            (SelectionListener) tableSelectionEvent.getListener();

        //invoke the event producer
        eventBindingListener.processSelection(selectionEvent);
    }

    /*
     * simplified method for invoking an EL for a single argument and
     * argument class
     */

    public Object processMethodExpression(String methodExpression,
        Object event,
        Class eventClass) {
        return processMethodExpression(methodExpression,
            new Object[] { event },
            new Class[] { eventClass });
    }

    /*
     * method that executes a method expression
     */

    private Object processMethodExpression(String methodExpression,
```

```
                Object[] parameters,
                Class[] expectedParamTypes) {
    FacesContext fctx = FacesContext.getCurrentInstance();
    ELContext elctx = fctx.getELContext();
    Application app = fctx.getApplication();
    ExpressionFactory exprFactory = app.getExpressionFactory();
    MethodExpression methodExpr =
        exprFactory.createMethodExpression(elctx,
                                           methodExpression,
                                           Object.class,
                                           expectedParamTypes);
    return methodExpr.invoke(elctx, parameters);
}
}
```

Event handler method

```
import adf.sample.view.ShowEmployeesBean;
import javax.el.ELContext;
import javax.el.ExpressionFactory;
import javax.el.ValueExpression;

import javax.faces.application.Application;
import javax.faces.context.FacesContext;
import oracle.adf.model.binding.DCBindingContainer;
import oracle.adf.view.rich.component.rich.data.RichTable;
import oracle.adf.view.rich.context.AdfFacesContext;

import oracle.binding.OperationBinding;

import oracle.jbo.uicli.binding.JUCtrlHierBinding;

import org.apache.myfaces.trinidad.model.CollectionModel;

public class ContextualEventHandler {
    public ContextualEventHandler() {
        super();
    }

    public void updateTableForDepartmentId(String departmentIdPayload) {
        FacesContext fctx = FacesContext.getCurrentInstance();
        Application app = fctx.getApplication();
        ELContext elctx = fctx.getELContext();
        ExpressionFactory expressionFactory =
            app.getExpressionFactory();

        //access the managed bean defined in the bounded task flow for
        //the employees table
        ValueExpression showEmployeeBeanAccess =
            expressionFactory.createValueExpression(elctx,
```



```
        "#{backingBeanScope.showEmployeesBean}",
        Object.class);

//cast it to the bean instance
ShowEmployeesBean employeesBean =
    (ShowEmployeesBean) showEmployeeBeanAccess.getValue(elctx);

//get access to the table component
RichTable table = employeesBean.getEmployeeetable();
//get access to the binding layer used by the table
CollectionModel model = (CollectionModel) table.getValue();
//access the tree binding used by the table
JUCtrlHierBinding employeesTableBinding =
    (JUCtrlHierBinding) model.getWrappedData();

//access the binding container
DCBindingContainer dcbindings =
    employeesTableBinding.getBindingContainer();

//access the EecuteWithParams method exposed in the binding
//layer to re-query the table data based on the contextual event
//payload
OperationBinding operationBinding =
    dcbindings.getOperationBinding("ExecuteWithParams");
operationBinding.getParamsMap().put("departmentIdVar",
    departmentIdPayload);
operationBinding.execute();

//refresh the table without re-loading the bounded task flow
AdfFacesContext.getCurrentInstance().addPartialTarget(table);
    }
}
```

Download and run the Sample

You can download the Oracle JDeveloper 11g (11.1.1.3) workspaces from the ADF Code Corner website. It is published as sample 68.

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

Configure the application database connect information to point to a HR schema in a local database before running the application.

Credits

I like to give credits to Jan Vervecken and John Stegeman, who first reported the problem with the default row change event to the Oracle JDeveloper forum on OTN.

RELATED DOCUMENTATION

<input type="checkbox"/>	Contextual events in ADF product documentation http://download.oracle.com/docs/cd/E15523_01/web.1111/b31974/web_adv.htm#CACJBFGI
<input type="checkbox"/>	Oracle Fusion Developer Guide – McGraw Hill Oracle Press, Frank Nimphius, Lynn Munsinger http://www.mhprofessional.com/product.php?cat=112&isbn=0071622543
<input type="checkbox"/>	