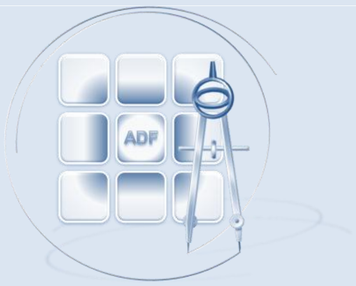
ADF Architecture Square

Region Interaction: Contextual Events Callback Pattern



twitter.com/adfArchSquare

Abstract

ADF contextual events are a communication channel for implemented through the ADF binding layer that you can use for communications between independent regions or between a region and its parent view in your ADF applications.

A common use case for contextual events is to invoke a managed bean configured in a region to perform data actions and to refresh the region's user interface.

This article details a generic solution which you may deploy and reuse from an ADF library.

Author: Frank Nimphius
Date: 15/Oct/2012

Introduction

Contextual events are a publish-subscribe communication mechanism implemented through the ADF binding layer that allows one part of the page to publish events to another part of the page to consume, typically allowing parent-page-to-region, a-region-to-the-parent-page, or region-to-region messaging. Typically this is configured in the pageDef of each view activity, where the publisher defines the message and the consumer subscribes to the message and what method to call on receiving the message.

The sender or publisher of the message is also referred to as the producer. The message producer can be defined as a generic event binding or, more specifically, associated with an attribute-, operation-, action- or collection binding in the producer page's PageDef file. For purposes of demonstration, this article will assume an event binding in the "parent" page's PageDef file that is created for the attribute binding referenced from the **ManagerId** field of a Department form. Whenever the **ManagerId** field value for a department changes, a value change event is raised on the attribute binding and propagated as a contextual event for other regions to consume and act upon.

The event is propagated to all active PageDef files (which are binding containers at runtime). On its way it "knocks" on every door asking the PageDef file to handle the event. If a PageDef file of a view displayed in one of the regions is configured to respond to this specific event, the message is passed to the message receiver (also known as the event or message handler).

Usually the message handler is defined as a method binding in the PageDef or an action binding like the ADF BC, Next, Previous, Delete or Commit operations in the PageDef file of the view displayed in the region. When the event handler receives the event notification, it also receives the message payload, like the changed **ManagerId** from the example above. The payload definition is defined by the message sender, requiring the event receiver to know about the format.

Event handlers usually are specific for a view and are programmed as a JavaBean method, which after receiving the event, would typically find a component on a page or a value in manage bean to update as a result. For a JavaBean method to be exposed as a methd binding in a PageDef file, you need to expose it as a JavaBean data as explained in the Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework product documentation.

What's left in completing the plain English explanation of contextual events is to look at who opens the door when an events "knocks" on the door of a PageDef file. The gatekeeper for contextual events is a subscriber facility (also know as the event mapping) that knows which events a PageDef file in a region is interested in and which method or action binding to invoke.

Now, here's the catch. Bounded taskflows don't consist of a single view and thus regions may expose many different views at a time. To avoid creating a JavaBean and data control as an event handler for each view, you need to be creative. Usually this creativity is expressed by a single JavaBean and data control defined for handling the event. The method in the JavaBean that handles the event uses a JSF `ValueExpression` to resolve the reference to a managed bean in the region bounded task flow to delegate the event handling to.

The following paper explains this concept in more detail:
http://www.oracle.com/technetwork/issue-archive/2011/11-may/o31adf-352561.html

While the approach explained in this paper works, there is a more elegant and generic way using managed bean callback configuration that this paper will demonstrate.

Credits: This article is based on an idea of Jan Vervecken from Contribute in Belgium (http://www.contribute.be/).

Prerequisite

This article assumes the reader to be familiar with the contextual event framework in Oracle ADF as documented in the Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework product documentation.

A Generic Callback Handler Approach to Contextual Event Handling

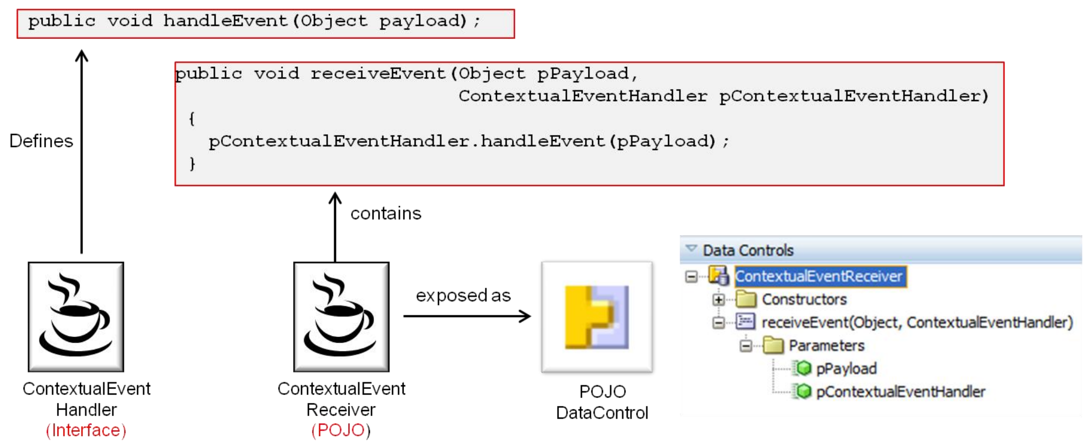
The change suggested in this article is to replace the `ValueExpression` in the JavaBean method that handles the contextual event with a call to a registered managed bean. To replace the expression in the JavaBean, three things need to happen

- The message payload must be passed as an argument to the JavaBean method that handles the message.
- A second argument of the method is used to pass an object reference to the managed bean in the region that you want to handle the contextual event.
- The managed bean you pass as reference must contain a commonly known public method that accepts the event message as an input argument. To ensure all managed beans that register as an event handler have this method defined, you use a Java interface class that all of these managed beans implement.

Figure 1 shows an example implementation of a generic contextual event receiver that allows developers to configure a managed bean to be called when an event is received. For this, the event receiver method defines two arguments: i) the payload delivered by the contextual event and ii) a callback handler reference to a managed bean implementing the contextual event handler interface (`ContxtualEventHandler` in this sample).

The event handler method in the `ContextualEventReceiver` class receives the managed bean reference and invokes its `handleEvent()` method defined by the contextual event handler interface. By invoking the `handleEvent()` method, the event receiver passes the payload it receives from the contextual event to the managed bean.

The event receiver class, `ContextualEventReceiver` in the sample, is exposed by the JavaBean Data Control so the `receiveEvent()` method it contains can be exposed as a method binding on a PageDef file.



```
public void handleEvent(Object payload);
```

```
public void receiveEvent(Object pPayload,
                         ContextualEventHandler pContextualEventHandler)
 {
   pContextualEventHandler.handleEvent(pPayload);
 }
```

Figure 1: Managed bean interface and event handler method

You can save the bean data control, the receiver bean and the interface class in an ADF library for reuse in other projects.

Configuring the Generic Callback Message Handler

Assuming you deployed the files shown in Figure 1 in an ADF library, adding the ADF library to a view controller project makes the POJO Data Control available in the Oracle JDeveloper Data Control panel for declarative addition to a PageDef file.

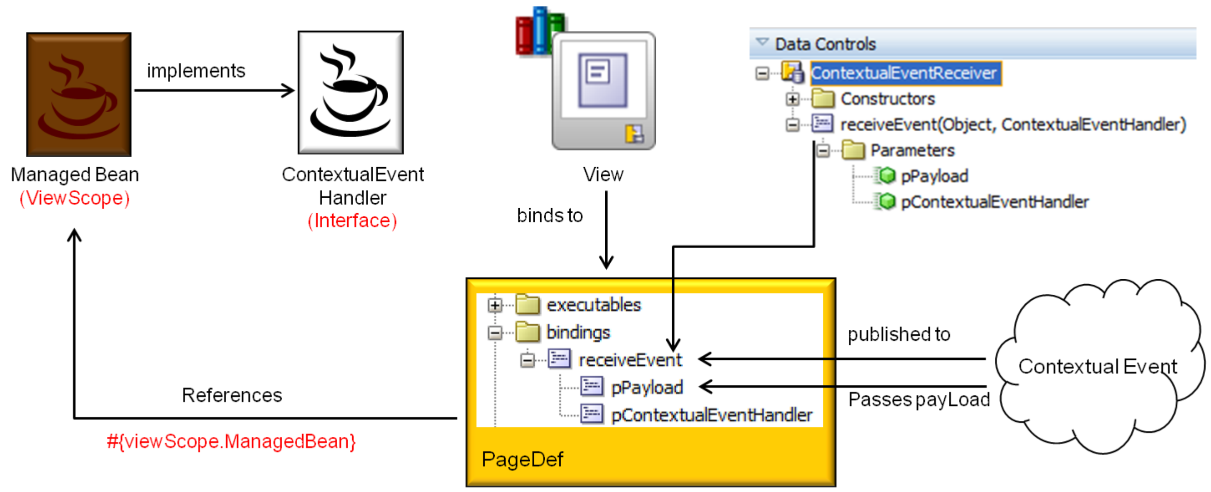


Figure 2: Contextual event callback handler in action

To configure the generic event receiver for a view in a task flow:

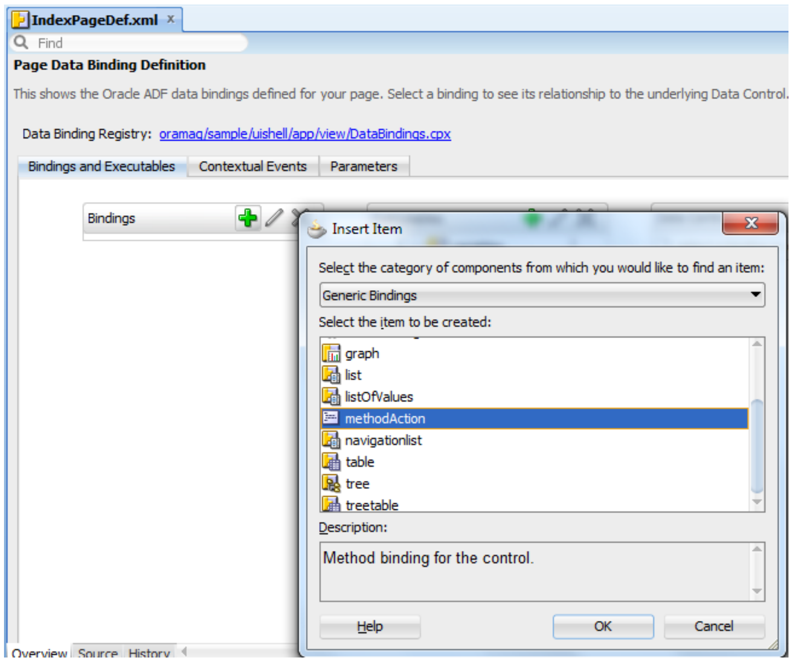
1. Create or download the ADF library containing the generic contextual event handler and configure it with your ViewController project

2. **Create a managed bean** in backing bean scope that implements the `ContextualEventHandler` interface. Implementing the `ContextualEventHandler` interface you define the call back method – `handleEvent(…)` – that will be called by the generic contextual event receiver to pass control to the managed bean.

3. **Create a method binding** in the PageDef file of the view that you want to listen for contextual events.

For this, open the view's PageDef file in Oracle JDeveloper and press the plus icon in the "Bindings" section, as shown in Figure 3. In the **Insert Item** dialog, click the **methodAction** item in the **Generic Bindings** category and select the POJO Data Control that exposes the `receiveEvent` method.

Use the EL editor to reference the managed bean you created before for the **pContextualEventHandler** argument. Leave the **pPayLoad** argument blank as this will be provided by the contextual event mapper.



IndexPageDef.xml

Page Data Binding Definition

This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship to the underlying Data Control.

Data Binding Registry: oramag/sample/uishell/app/view/DataBindings.cpx

Insert Item

Select the category of components from which you would like to find an item: Generic Bindings

Select the item to be created:

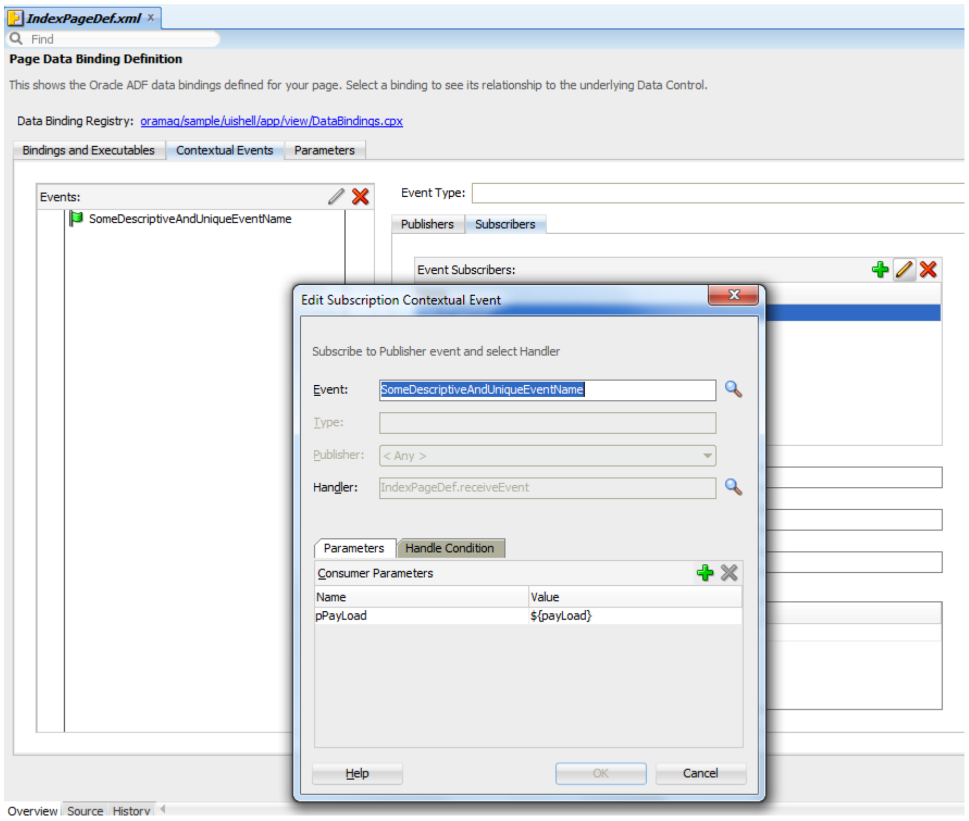
- graph
- list
- listOfValues
- methodAction
- navigationlist
- table
- tree
- treetable

Description:

Method binding for the control.

Figure 3: Creating a method binding manually

3. **Configure a contextual event subscriber definition** (as shown in Figure 4) in the same PageDef file you defined the method binding in



IndexPageDef.xml

Page Data Binding Definition

This shows the Oracle ADF data bindings defined for your page. Select a binding to see its relationship to the underlying Data Control.

Data Binding Registry: oramag/sample/uishell/app/view/DataBindings.cpx

Events: SomeDescriptiveAndUniqueEventName

Edit Subscription Contextual Event

Subscribe to Publisher event and select Handler

Event: SomeDescriptiveAndUniqueEventName

Type:

Publisher: < Any >

Handler: IndexPageDef.receiveEvent

Consumer Parameters

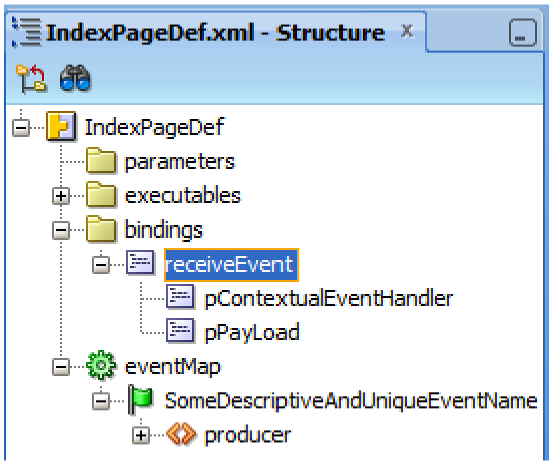
Name	Value
pPayLoad	${payLoad}

Figure 4: Contextual event subscriber definition dialog

Configure the contextual event subscriber as follows

- **Producer name:** choose "Any" in JDeveloper which will add the '*' wild card to the configuration. This basically means that you accept calls from all producers
- **Event name:** Here you define the name of the event this method binding should listen for. The event name is defined by the event producer.
- **Payload**: The value of the payload field should map the **pPayLoad** argument to the payload sent with the incoming event (as shown in Figure 5). The event payload is accessible from **${payLoad}** (note the capital L on Load, this is required)

Figure 5 shows the final configuration as it shows in the view's PageDef file. Note that the event name needs to match name of the producer event you define when setting up the event producer.



```
IndexPageDef
    parameters
    executables
    bindings
        receiveEvent
            pContextualEventHandler
            pPayLoad
    eventMap
        SomeDescriptiveAndUniqueEventName
            producer
```

Figure 5: Method binding and event map definition

Summary

This article explains an implementation for a generic and reusable contextual event handler that invokes a method on a configured managed bean to have the contextual event payload handled in the context of a view to be changed in response of an event.

RELATED DOCUMENTATION

☒	Implement Contextual Events http://www.oracle.com/technetwork/issue-archive/2011/11-may/o31adf-352561.html
☒	ADF Contextual Events in Product Documentation http://docs.oracle.com/cd/E35521_01/web.111230/e16182/contextual_events.htm#BABFCAFI
☒	