ADF Code Corner

018 .ERRATA: Oracle Fusion Developer Guide - Building Rich Internet Applications with Oracle ADF Business Components and ADF Faces" (ISBN - 978-0-07-162254-7)

Abstract:

To err' is human. This page lists known issues and correction to the McGraw Hill "Oracle Developer Guide" by Frank Nimphius and Lynn Munsinger. We also use this page to further clarify topics covered in the book and to provide additional samples created after book completion. Its an ERRATA and GOODIES page in one to serve ADF users the best.

twitter.com/adfcodecorner

Author: Frank Nimphius, Oracle Corporation
twitter.com/fnimphiu
2009/2010

Chapter 01:

Page 27	Addition to using scope prefixes in EL when reading memory scope attributes

The book has it correct. However, we want to make sure memory scope prefixes are understood correctly and that there is no question left open in regards to managed beans:

Accessing a managed bean in a standard servlet scope like sessionScope or requestScope using the scope as a prefix fails if the bean instance does not exist. Thus, bean reference like #{sessionScope.myBean} may fail while #{myBean} always succeeds. The reason for this is that #{sessionScope….} and #{requestScope…} reference a Map in memory and not the JSF framework.

Managed beans must be instantiated before they become available in the memory scope, which means they need to be accessed through JSF. Luckily, JSF does not allow to configure two managed beans with the same name in different scopes. So even without a scope prefix, there is no risk that application code accidentally accesses the wrong object.. Note that using ADFc specific scope, like viewScope and pageFlowScope, you always need to use the scope name as a prefix in the EL.

Page 44	Misspelling of partial target

Page 44 lists the following code lines:

```
AdfFacesContext adfFacesContext = AdfFacesContext.getCurrentInstance();
adfFacesContext.addPartialTargets(<target component instance>);
```

It should be of course:

```
AdfFacesContext adfFacesContext = AdfFacesContext.getCurrentInstance();
adfFacesContext.addPartialTarget(<target component instance>);
```

Chapter 03:

Page 100	Use Case; Using af:subform in ADF Forms - new sample provided

A complimentary sample is posted on ADF Code Corner implementing this use case: See it here

Chapter 05:

Page 168	Creating and Registering a custom Exception Handler

The custom exception handler example extends AdfcExceptionHandler, which is a class in an internal package. The risk associated with classes in internal packages is that changes may be made by Oracle without further notice. Oracle updated the upcoming version of the product documentation, "Oracle Fusion Middleware Fusion Developer's Guide for Oracle Application Development Framework 11g" with a sample that explains how to configure custom exception handlers, following our example in the book. They corrected the use of AdfcExceptionHandler by using ExceptionHandler, the class that is good to use with no strings attached. The sample thus would look like

```
import javax.faces.context.ExternalContext;
import javax.faces.context.FacesContext;
import javax.faces.event.PhaseId;
import oracle.adf.view.rich.context.ExceptionHandler;

public class CustomExceptionHandler extends
ExceptionHandler {

    public CustomExceptionHandler() {
        super();
    }

 public void handleException(FacesContext facesContext,
        Throwable throwable,
        PhaseId phaseId) throws Throwable
   {
     String error_message;
     error_message = throwable.getMessage();
     if (error_message != null &&
        error_message.indexOf("ADF_FACES-30108") > -1)
     {
     ExternalContext ectx =
                       FacesContext.getExternalContext();
       ectx.redirect("faces/SessionExpired.jspx");
     }
     else
     {
       throw throwable;
     }
   }
}
```

Note that using `ExceptionHandler`, the handleException method does not call super.handleException(...) but throws the exception so it is handled by the next registered exception handler.

The `ExceptionHandler`, class does not implement the handleException method itself and only acts as a template for defining ADF task Flow exception handlers.

Page 184	**Restoring a Savepoint from Java**

The getSavePointRestoreUrl is available from the ControllerContext, not the AdfFacesContext. So the following code line in our wxample needs to be corrected

```
restoreUrl =
     adfFacesContext.getSavePointRestoreURL(savepointId);
```

The ControllerContext variable is "cc" in the example, so that the code needs to be changed to

```
restoreUrl = cc.getSavePointRestoreURL(savepointId);
```

Page 168	**Creating and Registering a custom Exception Handler**

On page 168, the paragraph about Creating and Registering a Custom Exception Handler Class says that this text file should be created in the following folder:

.adf\META-DATA\\services

The correct folder though is

.adf\META-INF\\services

Chapter 06:

Page 199	**Accessing the Task Flow Binding from Java**

The code example casts the Task Flow Binding to `DCTaskFlowBinding`, which is an internal class used by the ADF framework. To avoid using internally packaged classes, you can cast the region binding to DCBindingContainer, which is the public framework class that `DCTaskFlowBinding` extends.

```
BindingContext bctx = BindingContext.getCurrent();
BindingContainer bindings =
                      bctx.getCurrentBindingsEntry();
DCTaskFlowBinding taskFlowBinding = null;
DCBindingContainer taskFlowBinding =
     (DCBindingContainer) bindings.get("dynamicRegion1");
```

The primary use of accessing the region binding is to get a hold of the referenced binding container and its defined bindings, which you can also do using the `DCBindingContainer`. There is some more infromation available using `DCTaskFlowBinding`, but these you can get from other APIs, like ControllerContext as well.

Chapter 09:

Page 309	**Note says: If the table is a child of the af:panelCollection component, then an implementation of the multiple column sort use case already exists using the PanelCollection View | Sort | Advanced menu option**

The book has it correct and the functionality exists. However, you need to set the table column selection to either single or multiple. This is not apparent because within the chapter, the column selection is enabled at the beginning to handle a different use case. The requirement to enable column selection has nothing to do with sorting but exists in the current JDeveloper 11g release. A bug has been filed to lift this requirement.

Page 309	**Selection Event**

The example shows you how to synchronize the table component row selection with the current row in ADF binding layer using Expression Language. If you prefer a pure Java solution, we released a generic Java handler example on ADF Code Corner

Page 296	**How to navigate in specific row in table.**

ADF Code Corner has an improved version of the sample in the book that is worth looking at. The sources are available for download as well.

Page 281	**"invokeMethodBinding" should be "invokeMethodExpression"**

Chapter 9 uses a helper method to invoke method expression. The main method "invokeMethodExpression" has a overloaded method with a simplified signature. Unfortunately the name of this method in the book is "invokeMethodBinding". It should however look as shown below to work with the samples given in the book

```
/**
* overloaded method as a convenience for the common case
* in which only a single argument is passed
*/
public Object invokeMethodExpression(
                                String methodExpression,
                                Object event,
                                Class eventClass){
  //call method shown below
  return invokeMethodExpression(methodExpression,
                                new Object[]{event},
                                new Class[]{eventClass});
}
```

So please put a note on the first method name that the name has changed as shown above.

Page 311	What You Should Know About the Data That Is Exported to Excel

In this section of the book we provide a hint of how to add an Excel fomular to the exported table cell data so that it gets propery formatted when opened in Excel.

This hint stopped working in the latest release of Oracle JDeveloper, which is JDeveloper 11g R1 PS2 (JDEVADF_11.1.1.3.PS2_GENERIC_100408.2356.5660) because of a bug fix that prevents the export of hidden output text content.

We assume that it requires a new enhancement request to properly implement the option to add excel formula (and some users unfortunately already started suffering from the side effect this bug fix has - including this book).

Chapter 15:

Page 483	Typo "exiting" instead of "existing"

Luc Bors from Amis in the Netherlands found this interesting typo in the book: “A standard JSF component that is built from exiting ADF Faces Components …” . This of course should be "A standard JSF component that is built from existing ADF Faces Components ..."

Chapter 19:

Page 601	Registering the adf-js-partition.xsd Schema

The adf-js-partitions.xsd schema has been moved to <wls_jdev_install>\oracle_common\modules\oracle.adf.view_11.1.1\adf-richclient-api-11.jar

Page 601	Creating te adf-js-partitions.xml file

The custom adf-js-partitions.xml file structure has a typo and wrong xml tag. The correct XML is shown below

```
<?xml version="1.0" encoding="utf-8" ?>
<partitions
   xmlns="http://xmlns.oracle.com/adf/faces/partition">
 <partition>
 <partition-name> ... </partition-name>
 <feature></feature>
 ...
 </partition>

 <partition>
 <partition-name> ... </partition-name>
 <feature>...</feature>
 ...
 </partition>
</partitions>
```

RELATED DOCOMENTATION

☒	Oracle Fusion Developer Guide – McGraw Hill Oracle Press, Frank Nimphius, Lynn Munsinger http://www.mhprofessional.com/product.php?cat=112&isbn=0071622543
☒	
☒	