An Oracle White Paper October 2010

JavaServer Faces 2.0 Overview and Adoption Roadmap in Oracle ADF Faces and Oracle JDeveloper 11g

In	troduction	. 5
JS	SF 2.0 new features	. 5
	A new way to define pages: The Facelets VDL	. 6
	Templates	. 6
	Composite components	. 6
	Resource handling	. 7
	Client behavior	. 7
	Ajax integration	. 7
	GET requests and bookmarking support	. 8
	System events	. 8
	JSF annotations	. 9
	faces-config ordering	10
	New scopes	10
	New implicit EL objects	10
	Navigation	10
	Project stage configuration	11
Comparison of JSF 2.0 with ADF Faces		11
	Ajax	11
	Partial state saving	12
	Facelets	12
	Behavior	12

Resource handling	13	
View scope	13	
Composite components	13	
JSF page bookmarking	13	
Implicit navigation	14	
Annotations	14	
Error handling	14	
Validation	14	
Naming container separator	14	
Project stage	15	
ADF Faces JSF 2.0 adoption: Q & A		
Oracle JDeveloper release specific questions	15	
ADF Faces specific questions		
ADF binding-specific questions	20	
ADF Controller specific questions	20	
Conclusion		

4

Disclaimer

The following is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle.

Introduction

JavaServer Faces 2.0 (JSF) is the new standard web user interface technology in Java Enterprise Edition (Java EE) 6. It is an update to version 1.2 of the same technology.

This revision to the JSF standard introduces two key themes for developers of rich internet applications (RIAs). The first of these is the native integration of Asynchronous JavaScript with XML (Ajax) into the JavaServer Faces request lifecycle and the second is the change of the default view declaration language from Java Server Pages (JSP) to Facelets. Additionally, other new features include the support for building composite components, additional scopes and enhancements in navigation, error handling and annotation-based managed bean configuration.

In the next release of Oracle JDeveloper 11*g* (11.1.2) Oracle intends to introduce support for JSF 2.0, both in the development environment and in the ADF Faces JSF component set. This whitepaper briefly introduces the JavaServer Faces 2.0 standard, helping developers to understand both, the differences between the baseline JSF 1.2 and JSF 2.0 and the functional overlap that exists between some parts of ADF Faces and version 2.0 of the standard.

This whitepaper outlines the roadmap for the planned adoption and integration of the ADF platform with JavaServer Faces 2.0. The target release for this initial JSF 2.0 integration is Oracle JDeveoper version 11.1.2, currently scheduled for calendar year 2011.

JSF 2.0 new features

The JSF 2.0 standard, which is a part of the larger Java Enterprise Edition 6 specification, was developed by the JSR-314 expert group (EG) within the Java community process (JCP) under the leadership of SUN. The EG was comprised of members from all of the key players in the JSF community, including leading developers of the Oracle ADF Faces and Apache Trinidad frameworks.

Many of the new features in JSF 2.0 have an equivalent implementation in one or more existing JSF frameworks or component sets which have pushed beyond the limitations of the JSF 1.2 specification. With this in mind, it is fair to say that JSF 2.0 is a standardization of the best of

these extended features. In addition it has not been afraid to reach out into the world beyond JSF for modern ideas, for example the new flash memory scope, an idea borrowed from the Ruby on Rails framework.

A new way to define pages: The Facelets VDL

New in JSF 2.0, the view declaration language (VDL) API defines the contract between the JSF runtime and the view technology used to host the JSF component tags. Before version 2.0 of JSF, Java Server Pages (JSP) was used as the default way to produce the JSF component tree at runtime. Unfortunately, the JSP request lifecycle, a simple parse-compile-render process, is not integrated with the more complex JSF request lifecycle. Commonly, because of this lifecycle mismatch, developers who used HTML and JSTL tags in the context of JSF views defined within JSP had to deal with unpredictable results in page rendering.

JSF 2.0 introduces Facelets as the new default view declaration language. The introduction of Facelets addresses the shortcomings of JSP and improves the developer page design and reuse experience. Unlike JSP documents, which are compiled into an intermediate Servlet at runtime, Facelets don't impose this unnecessary overhead and build the JSF component tree directly. This leads to far better performance in the component tree creation and page rendering processes.

When dealing with Facelets based JSF pages, the most obvious change for JSF developers is the file extension, which, by default, is ".jsf" rather than the ".jsp" or ".jspx" that they will have been used to. The content of a Facelets document is a contribution of HTML markup defining layout, JSF component references, Expression Language and a set of Facelets tags to include external content, reference templates, and to define parameters.

Templates

Facelets in JSF 2.0 provide a templating facility using named areas, similar to a named facet in JSF, identified by the ui:insert tag. Consuming pages use the ui:composition tag to reference the template Facelets source files.

Composite components

A composite component in JSF 2.0 is a Facelet file that is saved in an application specific resource directory. The composite component definition is composed of HTML markup and JSF components. Consuming pages include a namespace reference that points to the folder hosting the composite component file with the component filename being used as the component tag name. The interface definition of a composite component defines the attributes, facets and event listeners of the component for the consumer to discover.

This file-based approach to creating custom composite components is much simpler than the very technical task of building JSF components with the previous version of the specification.

Those familiar with the ADF Faces declarative components mechanism will immediately see the similarities.

Resource handling

JSF 2.0 finally provides a mechanism for component developers to reference resources such as images, CSS or JavaScript outside of the JavaServer Faces request lifecycle triggered by the Faces Servlet. Prior to standardization in JSF 2.0 this particular task was managed by a variety of ways by different component frameworks.

In JSF 2.0, resource handling is implemented through a specific packaging structure, which specifies that the external resources used by a component may be located either in the web projects root/resources folder or, on the class path in the META-INF/resources folder of a Java Archive (JAR) library.

Client behavior

Behavior tags in JavaServer Faces 1.2 were added to UI components to perform server side validation and value conversion. JSF 2.0 introduces the idea of client behaviors as a means to execute client side scripts declaratively for a UI component. A client behavior implements the ClientBehavior interface. It is registered to JSF using the FacesBehavior annotation or by using an entry in faces-config.xml. Using behaviors, the component or its renderer ensures that the associated JavaScript gets added to the page output and executed when a specific event occurs on the client. An example of a predefined client behavior tag provided by the JSF reference implementation is the f:ajax tag which is used to issue an Ajax request.

Ajax integration

Much like resource management, Ajax in JavaServer Faces is not new, but thus far has been implemented differently by each JSF component provider. These different implementations led to many conflicts when component sets were mixed into the same web user interface. The difficulty was in the different ways in which the browser DOM was used and manipulated in response to an Ajax request by the various component frameworks.

In JSF 2.0, jsf.ajax package provides a new JavaScript API that developers can use to fire Ajax requests in response to client-side component events. When using the Ajax API, developers provide arguments defining the event source, the action to execute, components to refresh in response to successful execution, a function to handle callbacks and an error handler. This Ajax API is, however, a low level API primarily suited to the needs of component developers. To simplify this Ajax API and to make it available for declarative use by component consumers, JSF 2.0 provides the f:ajax client behavior tag that was mentioned earlier. Developers use this tag to declaratively specify the client function to invoke when the event occurs, the error handler and the components to refresh.

GET requests and bookmarking support

The default navigation in JSF is conducted through postbacks (HTTP POST), which function without adding the request parameters to the request URL. However, the use of postbacks has always presented a challenge for developers who wanted to make their pages bookmarkable. In this scenario, it is useful to capture the request parameters within the bookmarked URL so that the state for the page can be re-constructed. As a result, prior to JSF 2.0 this bookmarking use-case within JSF applications was difficult to achieve, and in many cases not possible at all.

JSF 2.0 adds support for GET requests in addition to POST. For GET requests to function, developers need to add a new tag to the page that maps an incoming URL request parameter to a managed bean method. This managed bean method can then be used to re-create the desired state for the bookmarked page. The new tags provided for this purpose are f:metadata and f:viewParam.

To complement this mechanism for restoring state on a GET request, JSF 2.0 also allows developers to issue page navigation events which use GET rather than post from the *h:button* and h:link components. Request parameter names and values are added to the GET URL using the existing f:param child tag. In this way, a URL can be constructed containing all of the required state information.

PreRenderViewEvent

In a related vein, using the f:event tag enclosed by the f:metadata tag on a page, developers can listen and respond to the PreRenderViewEvent for the page that fires after the request parameters are applied but before the requested view renders. Developers may define an event listener in a managed bean to prepare the page context before the view is served. It is also possible to use the information of a request parameter to programmatically navigate away from the requested view.

System events

In addition to component and phase events that the framework broadcasts in response to user UI interaction and request lifecycle handling, JSF 2.0 also provides a series of system events that can be handled on the component or application level.

JSF page authors can register listeners for these system events in Java from a managed bean, or declaratively using the new f:event tag, that can be added as a child of a parent UI component tag. The default system events that developers can listen for are

8

- preRenderComponent
- postAddToView
- preValidate

• postValidate

Although these system events appear to be similar to the existing lifecycle events supported by JSF 1.2, in reality they provide a much more fine-grained access for the developer than do the existing phase events.

A managed bean method used to handle these events must have a signature that accepts a single argument of type ComponentSystemEvent (essentially the same API used for action listeners and value change listeners.)

JSF annotations

One of the core themes introduced across the whole of the Java Enterprise Edition specification is the reduction in the number of metadata configuration files used. This is primarily managed through the use of Java annotations. JSF 2.0 has adopted this principle allowing developers to use annotations both when developing custom JSF components and when building JSF applications. For example, managed beans no longer need to be defined exclusively in the faces-config.xml file and instead it is sufficient to simply add an annotation to the desired Java code.

To define a managed bean, developers add the @ManagedBean annotation and refine that using one of the following scope annotations: @RequestScope, @ViewScope, @SessionScope or @ApplicationScope. If no managed bean name is provided explicitly in the annotation then the name of the Java class in standard camel case is used. Managed properties can be defined in a similar way.

In addition to the JSF 2.0 specific annotations, servlet lifecycle annotations such as @PostConstruct and @PreDestroy are supported as well.

Further annotations exist to simplify the life of component developers in the same way, removing the need for metadata artifacts in the component definition. The following annotations are available for this purpose:

- @FacesComponent
- @FacesRenderer
- @FacesConverter
- @FacesValidator
- @FacesBehavior

Note: The use of annotations can be disabled for a specific JSF application if the *faces-config* element in the faces-config.xml file has the *metadata-complete* attribute set to true.

faces-config ordering

Releases of JSF prior to 2.0 have no mechanism to control the loading order of facesconfig.xml files from JAR files in the classpath. As a result, there was no way for developers to control the order in which phase listeners or view handlers fired and loaded. In JSF 2.0, this restriction has been redressed, making it possible to coordinate the loading over multiple configuration files.

New scopes

JSF 2.0 adds two new scopes – flash and view – to the existing application, session, and request scopes. These new scopes provide better support for Ajax page refresh and partial submits.

The lifetime of objects in view scope lasts from the load of a view until its dismissal by the user on navigation to another view.

To reference an object in view scope, developers use an EL expression that is prefixed with the "viewScope" identifier, or in Java, by calling getViewMap() on the current UIViewRoot object. View scope can be thought of as sitting between request and session scope in terms of the object lifetime.

Flash scope allows developers to pass objects from one view to the next when transitioning. Unlike view scope, the flash scope survives redirects and GET requests and is only cleared once the new view renders. The flash scope is accessible from EL, using the "flash" prefix, and from Java, where it is exposed on the JSF ExternalContext object.

New implicit EL objects

JavaServer Faces 2.0 introduces new implicit objects that are accessible from Expression Language. Two of these have been shown in relation to the new memory scopes already. The new objects are

- component Object that gives access to the current UIComponent instance
- cc Object to access the current custom composite component
- flash Object that represents the new flash scope
- viewScope Object that references the new view scope
- resource Object to access the resource handler.

Navigation

Navigation has had one of the largest makeovers of all the areas within the JSF framework. In releases of JSF prior to 2.0, navigation was purely based on the outcome of an action, which

would be used to identify the appropriate navigation case definition within the applications faces-config.xml file.

The new implicit navigation feature added in JSF 2.0 allows developers to perform navigation directly to a view without needing a navigation case in the configuration file. Implicit navigation is attempted when no matching navigation rule is found for an action outcome.

If neither, explicit navigation using defined navigation cases in faces-config.xml and implicit navigation succeed, then, as in previous versions, the viewId is not changed and the source page is redisplayed.

Another new feature added to explicit navigation is conditional navigation. For this, the expert group added a new if element to the faces-config.xml vocabulary that allows developers to define a condition as part of the navigation case. This condition uses expression language to produce a boolean value which governs the ultimate navigation. In previous releases of JSF, a managed bean had to be used to evaluate conditions and return the appropriate navigation case string as the outcome.

The last enhancement to navigation in JSF 2.0 is preemptive navigation, which allows developers to programmatically determine the navigation target for a navigation case defined in the faces-config.xml file.

Project stage configuration

Finally, JSF 2.0 introduces javax.faces.application.ProjectStage as a new context initialization parameter. This parameter sets up the environment for the JSF application to execute in and has valid values of: Development, UnitTest, SystemTest, or Production.

This parameter, accessible through Java using the Application.getProjectStage() method, allows JSF component and application developers to configure their code to behave differently based on ProjectStage settings. For example, an application running in a development environment may display more verbose error messages than when running in a production environment.

Comparison of JSF 2.0 with ADF Faces

Several of the new JavaServer Faces 2.0 features have parallel functionality in Oracle ADF Faces.

Ajax

JSF 2.0 provides support for Ajax on various levels in its architecture, including a JavaScript API - jsf.ajax.request, declarative support in Facelets through the f:ajax tag, as well as various server-side APIs, such as PartialViewContext.

JSF 2.0 allows a partial response to be sent to the client instead of a full-page refresh. The equivalent functionality in ADF Faces is provided through partial page rendering features. ADF Faces and the new JSF standard differ in the following respects:

- Oracle ADF uses the ADF Faces component framework to dynamically register components for partial page refresh in response to a server side data change on the model layer. This auto-partial page refresh behavior is not available with JSF 2.0
- The Mojarra JSF 2.0 Reference Implementation (RI) has no file upload component and no capability to handle multipart/form-data requests, as it does not support iframebased Ajax requests. ADF Faces provides this functionality and will continue to do so in its own implementation until JSF provides a standard implementation.
- ADF Faces partial request calls are integrated with client side validation.

Moving forward, the plan for ADF Faces is to support both, the JSF 2.0 native Ajax APIs and the ADF Faces partial page refresh implementation.

Partial state saving

To reduce the overhead of state saving and management, JSF 2.0 introduces partial state saving as new functionality. In an effort to provide continued backwards compatibility, ADF Faces will utilize its existing state saving implementation while the new JSF 2.0 implementation matures.

Facelets

ADF Faces has supported Facelets since JSF 1.2. This support will be updated to align with the new standard Facelets APIs. In addition, MDS support, which previously existed only for JSP documents, is targeted to become available for Facelets as well.

ADF Faces will continue to support both JSP and Facelets. However, since many JSF 2.0 features are not available for the JSP, Facelets will become the recommended View Declaration Language.

Behavior

The ADF Faces client behavior functionality has been extended in Oracle JDeveloper 11g R2 to ensure that JSF 2.0 client behaviors can be used. Future releases will target aligning the ADF Faces internal implementation with the new JSF ClientBehavior API.

Compared to JSF RI, the implementation of ADF Faces client behavior differs in that events are raised at the component level and not the DOM. The event is broadcast to interested listeners and contains the ADF Faces specific component event and the native DOM event for JSF 2.0 behaviors to respond to. ADF Faces will continue supporting component level JavaScript events while extending support for JSF 2.0 behavior tags as well.

Resource handling

To handle resource loading efficiently, using the af:resource tag, ADF Faces adds styles and script sources to the page header. For its own resource handling, ADF Faces continues using the Trinidad ResourceServlet and the af:resource tag. Application developers can also take advantage of the new JSF resource loading mechanism within their ADF Faces applications as well.

View scope

JSF 2.0 provides a new memory scope, view scope, which holds its values for the duration that a specific view is present. ADF Controller also provides a scope of the same name.

The difference between the two view scopes is that JSF 2.0 stores information in a map on the UIViewRoot, which is emptied upon page refresh or a redirect to the view.

The ADF Controller view scope is refreshed when users navigate to a new view, abandoning the currently displayed view. Thus data stored in the ADF Controller's view scope survives refreshes and redirects to the same view.

Utilizing JSF 2.0, ADF Faces supports the new view scope defined on the UIViewRoot. To preserve the longer duration of the existing ADFc view scope, all requests to the JSF 2.0 view scopes will be transparently delegated to the existing ViewScope provider.

Composite components

JSF 2.0 composite components are comparable to ADF Faces declarative components and page templates. However, unlike JSF 2.0 composite components, ADF Faces components are capable of having their own backing bean scope. In addition, JSF 2.0 composite components are limited to the Facelets VDL, whereas the ADF faces declarative components support both the Facelets and JSP VDL. Because of the advantages offered by declarative components, ADF Faces will continue to support them while adopting support for composite components as well.

JSF page bookmarking

JSF 2.0 provides support for GET request handling and parameter mapping, allowing developers to build JSF pages that know how to recreate dynamic state when referenced from a bookmark.

Unbounded task flows in Oracle ADF Controller also provides bookmarking support, which, compared to how JavaServer Faces 2.0 implements bookmarking, does not require parameter mappings to be defined page sources or navigation to be performed with specific GET request enabled UI components. ADF Controller continues to support bookmarking when adopting JavaServer Faces 2.0.

Implicit navigation

Implicit navigation in JavaServer Faces 2.0, which does not require a navigation case to be defined if a view id with the given action name exists, is planned to be supported in ADF Faces but may not be available in Oracle JDeveloper 11g 11.1.2.

One of the motivating factors behind development of ADF Controller is the ability to route navigation to non-viewable targets like method call activities or router decisions. JavaServer Faces 2.0 continues to only support navigation between views, which means that pre-emptive navigation cannot be used to navigate to non-view activities like methods.

Annotations

JSF 2.0 supports annotations for defining managed beans instead of configuring them in the faces-config.xml file. When utilizing ADF Controller, managed beans may be configured for use in bounded task flows, which requires registration in the configuration file. Based on this consideration, when using ADF Controller for navigating in JavaServer Faces 2.0 projects, bean definition will continue to be required in the configuration file.

Error handling

JSF 2.0 introduces the new ExceptionHandler API to centrally handle unexpected exceptions occurring during the request lifecycle. ADF Faces provides a similar service that continues to be supported. Exceptions that are not handled by the ADF Faces exception handler will be delegated to the JSF2 exception handler for processing.

Validation

JSF 2 introduces the javax.faces.VALIDATE_EMPTY_FIELDS context parameter to allow empty fields to be validated when bean validation is available. This validation has been implemented and confirmed for ADF Faces as per JavaServer Faces 2.0 requirements.

Another key feature of validation in JSF 2.0 is the integration with JSR-303 (Bean Validation) using the f:validateBean tag. The impact of bean validation integration utilizing the f:validateBean tag with ADF Faces is under review and should be a supported feature shortly.

Naming container separator

Naming containers in JSF are like namespaces that ensure unique naming for their child components to prevent naming conflicts within the scope of a view. In JSF 2.0, the colon character that is used by default to distinguish the naming container id from the component id can be configured via the javax.faces.SEPARATOR_CHAR context parameter. ADF Faces plans to resolve this requirement in a subsequent release of its JSF 2.0 support.

Project stage

The JSF 2.0 project stage feature will be supported in ADF Faces. Developers use the project stage to define ADF Faces context settings for a specific runtime environment. For example, during development, developers may enable debugging and assertions for the application to test and also may disable content compression. In production mode however, these settings are not recommended and a warning is printed.

ADF Faces JSF 2.0 adoption: Q & A

The following addresses development questions that developers may have when moving existing ADF Faces applications from JavaServer Faces 1.2 to JavaServer Faces 2.0.

Oracle JDeveloper release specific questions

Which release of Oracle JDeveloper supports JavaServer Faces 2.0?

Oracle JDeveloper 11g R2 (11.1.2) is the first JDeveloper release that supports JSF 2.0.

How do I develop JSF 1.2-based applications in JDeveloper 11g R2?

Application developers who want to use the new release of Oracle JDeveloper to maintain or further develop JSF 1.2-based applications need to migrate their project dependencies to JSF 2.0. The latest version of Oracle JDeveloper 11g no longer supports JavaServer Faces 1.2. Developers who do not want to migrate to JSF 2.0 should stay with Oracle JDeveloper 11.1.1.x.

How can I migrate existing ADF Faces applications to JSF 2.0?

JDeveloper will automatically migrate existing JSF 1.2-based applications to JSF 2.0 when the application is first opened. At this point, JDeveloper displays an alert giving the user the option to prevent the migration. Developers that decline migration will not be able to open the application in Oracle JDeveloper 11g R2 and have to continue with Oracle JDeveloper 11g R1. Accepting the migration dialog will change the projects library dependencies to JSF 2.0. No changes are applied to the application code.

If an ADF Faces application uses JavaServer Faces 1.0 and ADF Faces 10.1.3, then the migration path is to first upgrade to 11g, which is done using Oracle JDeveloper 11g R1.

ADF Faces specific questions

What is the benefit of using ADF Faces rather than JSF 2.0 RI components with JavaServer Faces 2.0?

Oracle ADF Faces is more than simply a JSF component set. It is a view layer development framework built on top of the JSF standard APIs. ADF Faces provides more than 150 components, including components that don't exist in the JSF 2.0 reference implementation, like data visualization components, graphical maps and regions.

Along with the extra components, ADF Faces provides services like drag and drop, lightweight dialog handling, active data streaming and integration with the JSR-227 based ADF binding framework, as well as a complete JavaScript component client architecture that is not available in the JSF standard.

Will ADF Faces applications integrate with JSF 2.0?

ADF Faces applications whether migrated to JSF 2.0 or developed new using Oracle JDeveloper 11g R2, integrate with JavaServer Faces 2.0 and therefore can use the new standard functionality. To remain backward compatible and to further support functionality not available in JSF 2.0, existing advanced ADF Faces functionality continues to be supported.

Are migrated applications expected to use Facelets?

Existing ADF Faces applications do not need to change their VDL to Facelets and may continue using JSP documents.

"Any JavaServer Faces implementations that claims compliance with this specification must include a complete JavaServer Pages implementation, and expose this implementation to the runtime of any JSF application. JSF applications, however, need not use JSP as their View Declaration Language (VDL)."

However, new features added to the JSF 2.0 standard require Facelets as the View Declaration Language. If JSP documents are used with JSF today, Oracle recommends moving to Facelets as soon as possible.

Facelets is a replacement for JSP that was designed from the outset with JSF in mind. New features introduced in version 2 and later are only exposed to page authors using Facelets. JSP is retained for backwards compatibility.

JavaServer™ Faces Specification 2.0, Chapter 10

JavaServer™ Faces Specification 2.0, Chapter 9

Customers using Facelets in JavaServer Faces 1.2 should be aware that the Facelets version standardized in JavaServer Faces 2.0 is not fully backwards compatible with the previous open source version. This also is explicitly stated in the JSF 2.0 specification.

Existing ADF Faces applications that use JSP as the VDL are not required to migrate to Facelets, though they would benefit from the Facelets advantages:

- Cleaner integration with JSF Facelets was designed from the ground up to target the JSF use case. On the other hand, JSF support was somewhat retrofitted into the existing JSP architecture. As such, Facelets provides cleaner integration with JSF.
- No compilation required Facelet views are parsed directly by the Facelets engine, avoiding the need to translate the page definition into Java.
- Improved page templates Facelets provide integrated page templating.
- Improved error reporting Runtime errors, such as EL evaluation exceptions, can be hard to trace because the JSP engine does not always provide clear guidance of where in the document the problem is located. Facelets provide more detailed information, including the line number of the failed metadata definition on the page, making it easier to debug.
- Better performance Facelets don't need to be compiled into Java code at runtime, which adds to better performance.

Are there arguments for not moving from JSP to Facelets documents?

Facelets and JSP documents handle JSF view definitions differently at runtime. JSP documents handle JSF page definitions as XML metadata that is fully parsed and processed by the server side JSP runtime engine. Facelets treat JSF pages as XHTML documents that are partially handled by the server side and the remainder processed by the browser.

The main argument to stay with JSP documents as the JSF page VDL is to wait for the JSF expert group to improve Facelets to become XML-aware so that only the content meant to be handled by the client is passed to the client.

Another reason to wait for the Facelets migration is if a project depends on libraries that don't yet have a Facelets equivalent.

Note: Automated JSP to Facelets migration as the VDL is out of scope for the Oracle JDeveloper 11g R2 release. Applications that are migrated to JSF 2.0 when opened in the new JDeveloper release will continue using JSP documents.

If I want to move to Facelets in the future, what do I need to consider today?

To prepare future migration to Facelets:

• CDATA blocks in JSP documents should be avoided – In early versions of Oracle JDeveloper 11g, the af:resource tag to hold or reference JavaScript and CSS sources did not exist. The best practice then was to use CDATA blocks in the af:document metaContainer facet to include those resources. When migrating to JSF 2.0, the recommendation is to change all references to JavaScript and CSS resources to using the af:resource tag.

- Scriptlets should not be used in JSF documents Though a rare use case, still developers use scriptlets within JSTL referenced on JSF pages. Facelets does not support scriptlets and any occurrence of scriptles in ADF Faces pages must be removed.
- Attention should be given to comments in JSP documents comments in JSP documents will be delivered to the Facelet pages if not striped out. Facelets has a context parameter to disable comments from being sent to the client.
- JSPX documents should be used Facelets requires valid XML documents, which is best to prepare for by using JSPX documents.

How long will Oracle support the use of JSPX documents with ADF Faces?

Oracle anticipates that developers will want to use new features in JSF 2.0. Many of the new features in JSF 2.0 require Facelets as the VDL. Until there is an acceptable migration path from JSP to Facelets, there are no plans for Oracle to discontinue support of the JSP VDL for ADF Faces.

However, the recommendation for developers is to adopt Facelets as soon as possible for their existing and new ADF Faces applications.

Will ADF Faces provide the same functionality and components for Facelets and JSPX?

For the initial release of ADF Faces on JSF 2.0 in Oracle JDeveloper 11g R2 this is the case. Some features in JSF 2.0, however, are available for Facelets only, which will certainly influence new ADF Faces functionality in the future.

Will ADF Faces page templates work with Facelets?

Yes, ADF Faces page templates can be used with Facelets.

Will ADF Faces declarative components work with Facelets?

The plan for declarative components is to allow developers to use them also with Facelets. For this, the declarative component needs to be opened in JDeveloper 11g R2 or later and then redeployed. However, Oracle does not provide a migration of existing declarative components to Facelets.

Why and when would I continue using declarative components?

The new composite component feature in JavaServer Faces 2.0 provides developers with the option to build custom component out of existing JSF components. However, this feature only works with Facelets as the VDL, not JSP.

ADF Faces declarative components work with JSP documents, as well as with Facelets for new declarative components created in JDeveloper 11g R2.

Use cases for ADF Faces declarative components include:

- The use of backing beans by the component where there may be multiple instances of the component on a page and the developer needs the ability to isolate each components bean.
- The developer is required to continue using JSP.

Can I use ADF Faces components to build JSF 2.0 composite components ?

Yes.

Can I use ADF Faces components within other third party JSF frameworks?

ADF Faces needs to own document and form, which means that the af:document and the af:form tags must be used. As such, placing ADF Faces components inside of arbitrary pages is not yet supported. Be aware that other component frameworks may have similar restrictions, so that even if ADF Faces af:document and af:form tags are used, the combination of the two component sets may not work or may not be supported.

Is it possible to mix and match JSF 2.0 API calls with ADF Faces API calls?

Yes. Developers who find the same functionality in both ADF Faces and JSF 2.0 are free to choose their preference. For the initial releases of ADF Faces JSF 2.0 support however the recommendation is to use ADF Faces APIs.

Is it posible to use jsf.ajax.request() calls in ADF Faces?

Yes. However, our recommendation is to use a declarative approach with the JSF 2.0 f:ajax tag or the existing ADF Faces partial submit and partial trigger implementation.

How do ADF Faces partial triggers integrate with JSF 2.0 Ajax requests ?

If the issuing component is an ADF Faces component, then partial triggers are evaluated and the referenced components are added to the list of partial targets. If the Ajax request is issued by a non ADF Faces JSF component then partial triggers are not resolved. Partial trigger resolution can be manually forced by a call to RequestContext.partialUpdateNotify() within a server side event handler.

Note: RequestContext is a Trinidad class that is wrapped by the AdfFacesContext class. Thus, it is also okay to call AdfFacesContext.partialUpdateNotify().

Does ADF Faces leverage the new error handling API in JSF 2.0?

If no exception handler is explicitly configured for ADF Faces applications, then the JSF default exception-handling mechanism is used. Even in the case of an ADF Faces configured exception handler, any exception that would propagate beyond the configured handler would be passed to the JSF 2.0 handler for processing if possible. ADF Faces will continue to support the existing oracle.adf.view.context.ExceptionHandler implementation however.

ADF binding-specific questions

Do I need to change ADF bound applications when adopting JSF 2.0 ?

The Oracle ADF binding layer is not impacted when moving ADF Faces from JSF 1.2 to JavaServer Faces 2.0. When opening an existing ADF application in Oracle JDeveloper 11g R2, the ADF binding libraries are automatically updated to include the latest version of the Faces control binding classes.

ADF Controller specific questions

Is it possible to define managed beans with annotations in ADF Controller?

Currently ADF Controller does not support annotated managed beans. All managed bean usages must be defined in the task flow metadata files or in the adfc-config.xml file.

Can I define managed beans in the JSF 2.0 view scope or flash scope?

ADF Controller, and thus task flows, doesn't currently support the JavaServer Faces 2.0 flash scope. If a managed bean is configured in viewScope, then the view scope of ADF Controller is used, which extends the scope object with the same name in JSF 2.0.

Will ADF Controller support JSF 2.0 preemptive navigation ?

The ADF Controller NavigationHandler implements ConfigurableNavigationHandler, which means that it can be used for pre-emptive navigation. However, if the matching navigation rule contains a non-viewable activity, like a method call activity, the navigation result cannot be determined pre-emptively and thus fails.

Conclusion

With Oracle JDeveloper 11g R2, version 11.1.2, ADF Faces now utilizes JavaServer Faces 2.0, which means that developers now have the option of integrating the new JSF 2.0 features in their development projects.

Integration of ADF Faces with JavaServer Faces 2.0 is an exciting process that has just begun with the release of Oracle JDeveloper 11g R2 and will continue to be significantly enhanced with the subsequent releases. The aim of ADF Faces is to continue to provide the unparalleled level of functionality and productivity to developers while moving to JSF 2.0. From an ADF Faces framework perspective, most of the changes in adopting JSF 2.0 are internal and should be transparent to developers.



JavaServer Faces 2.0 Overview and Adoption Roadmap in Oracle ADF Faces and Oracle JDeveloper 11g

December 2010 Author: Frank Nimphius Contributing Authors: Andy Schwartz, Duncan Mills, Shaun O'Brien

Oracle Corporation World Headquarters 500 Oracle Parkway Redwood Shores, CA 94065 U.S.A.

Worldwide Inquiries: Phone: +1.650.506.7000 Fax: +1.650.506.7200 oracle com



C | Oracle is committed to developing practices and products that help protect the environment

Copyright © 2009, Oracle and/or its affiliates. All rights reserved. This document is provided for information purposes only and the contents hereof are subject to change without notice. This document is not warranted to be error-free, nor subject to any other warranties or conditions, whether expressed orally or implied in law, including implied warranties and conditions of merchantability or fitness for a particular purpose. We specifically disclaim any liability with respect to this document and no contractual obligations are formed either directly or indirectly by this document. This document may not be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without our prior written permission.

Oracle is a registered trademark of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

0109