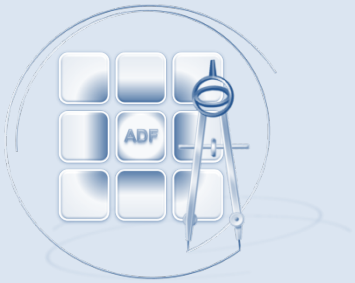
ORACLE

ADF Architecture Square

ADF Code Guidelines v2.00 – 02/01/2014



twitter.com/adfArchSquare

Abstract

This document sets out ADF code guidelines, practices and conventions for applications built using ADF Business Components and ADF Faces Rich Client (release 11g and greater). The guidelines are summarized from a number of Oracle documents and other 3rd party collateral, with the goal of giving developers and development teams a short circuit on producing their own best practices collateral.

The document is not a final production, but a living document that will be extended to cover new information as discovered or as the ADF framework changes.

Readers are encouraged to discuss the guidelines on the ADF EMG (http://bit.ly/JOr1wW) and provide constructive feedback to Chris Muir via the ADF EMG Issue Tracker (http://bit.ly/Qj2JAw).

Author: Chris Muir
Date: 02/01/2014

Goals

"Guidelines defined: Noun: A general rule, principle, or piece of advice."

The goals of this document are:

- Ease development and maintenance of ADF projects by providing a single document for ADF code guidelines
- Be flexible enough to adapt to change
- Easy to refer to
- Extensible

In producing this document Oracle has taken examples from a selection of projects and a wide range of documents to create a centralized set of guidelines. However not every single project and use case will have been identified and the document may be incomplete, or incorrect for all use cases, or simply too formal for your circumstances. As such the guidelines should not be seen as a straight jacket nor a set of best practices you should blindly follow in all cases, but rather as the definition of "guidelines" above infers, a set of general rules or advice that you can diverge from if you see necessary, which of course you should also document why and when this has occurred.

Oracle encourages you to use this document as a set of guidelines that you extend from for your own projects, note which guidelines are relevant to your projects and those that are not, and if you find the document unsatisfactory for whatever reason share this back with the original authors and the wider ADF community via the ADF EMG issue tracker http://bit.ly/Qj2JAw.

Oracle also recommends you don't start flipping features on or off based on the guidelines within this document without an understanding of what they do and their impact, particular performance and production related features. From the Oracle database community it is well known that you should always have a set of metrics before tuning the database so you can compare the results. The same applies to ADF and Java development.

In terms of the "currency" of the guidelines in this document, note this document was written and published when JDeveloper 11.1.1.6.0 and 11.1.2.2.0 were the latest available releases of JDeveloper in the 11gR1 and 11gR2 branches. Obviously in the future new features may be included in later releases that may make certain guidelines obsolete or invalid. Always attempt to seek the latest version of this document, and, if you detect a situation where the document is invalid please attempt to contact the original authors to share your concerns via the ADF EMG issue tracker http://bit.ly/Qj2JAw.

Document Structure

This document is structured by major topic area, starting with General guidelines, then project specific guidelines considering ADF Business Component then ViewController projects, followed by WebLogic Server. Each area is broken down into an alphabetical list of minor topics for each major topic area, starting with its own General area. Broadly speaking the document attempts to not repeat itself, moving repeated minor topics into a General area specified once. Feel free to leap in and read any topic in this

document, but realize that your reading needs to also consider the broader General areas which may also have information to the relevant topics at hand.

Future Extensions

This document is designed to be a living document that will be revised and updated as new guidelines come to light, as well as new subject areas are introduced and covered. The following topics are believed not to be adequately covered in the existing document, or not covered at all by this document, but hopefully will be included at some point in the future:

- Accessibility
- ADF Business Components - Application Module Pooling
- ADF Business Components - Database Issues
- View Controller - Internationalization and Bi-Directional Support
- Maven

Note this is not a definitive list of missing topics.

Out of Scope

The following topics are not covered by this document and will be potentially discussed in separate documentation to be released at a future data:

- ADF Faces RC Components
- Naming Conventions
- Application Layout Conventions

Document Annotations

If you choose to adopt the guidelines from this document each guideline has been annotated with the convention [ADFcg1-12345] so you may refer back to them from your own documentation and development code. The parts of the annotation include:

- ADFcg - Stands for "Application Development Framework Code Guidelines", essentially this document.
- ADFcg number prefix – This document uses a major-point-minor version numbering scheme such as v1.00 or v4.03. The number before the decimal point is the major version number, the number after the minor version number within the major version.

Major revisions to this document such as the introduction of guidelines will result in a new major version number. Minor revisions to this document such as the correction of spelling, grammar or similar will result in a minor revision number updates only.

The number directly after the ADFcg annotation refers to which major version number of this document the recommendation was introduced, or, if the recommendation was modified from the last version, the version number the recommendation was last modified. As such [ADFcg3-12345] is a guideline either introduced or last modified in v3 of this document.

- The remaining number after the hyphen is a unique number across all versions of this document. As such if you see guideline numbers [ADFcg1-12345] and [ADFcg1-12346] these are two distinct guidelines, however if you see numbers [ADFcg1-12345] and [ADFcg3-12345] these two guidelines are one and the same, the later number being a revision of the 12345 guideline at version 3.

  Over time it is expected there will be multiple releases of this document with new recommendations added, existing recommendations modified and obsolete recommendations removed. The following conventions will be followed for each guideline:

- Each guideline annotation (that is [ADFcg1-12345]) will be unique across all releases of the documentation. A guideline annotation will never be reused.
- If a guideline's text is significantly modified (beyond spelling mistakes and grammar fixes) between versions of this document the ADFcg number prefix will be updated to the current major version of this document. For example the recommendation ADFcg1-06000 introduced in release 1 of this document and updated in release 6 of this document will become ADFcg6-06000. If the recommendation is not subsequently updated in release 7, the recommendation annotation will remain at ADFcg6-06000.
- If a guideline becomes obsolete, the text will be removed except for the recommendation annotation marked as obsolete.

  Beyond the annotations used above, for some guidelines you'll note the guideline has a footnote that refers to an external document. Rather than duplicating the complete text from the external document, it's left up to the reader to pursue and read the links.

Change Log

The following changes have been made in this v2.00 of the guidelines:

Rule	Change	ADF EMG Issue #
n/a	Introduced this change log section.	n/a
ADFcg2-02066	New rule to advise developers to be careful when calling applyViewCriteria and setApplyViewCriteriaName not to override existing criteria.	ADFEMG-199
ADFcg2-01033	New rule to advise developers to check adf-config.xml file.	ADFEMG-198
ADFcg2-02067	New rule considering the effect of calling setAttribute() on a VO with LOVs	ADFEMG-197
ADFcg2-02068	New rule detailing not to reuse programmatic view criteria between view objects	ADFEMG-196
ADFcg2-03100	New rule dictating when to optimize consecutive method calls	ADFEMG-195
n/a	Minor fix: Fixed blurb to service interface rules.	n/a
ADFcg2-02069	New rule describing the optimization of the ADF BC ps_txn database table	ADFEMG-153
ADFcg2-02070	New rule recommending using a separate schema for jbo.server.internal_connection	ADFEMG-152
ADFcg2-02029	Rule has been updated to a mandatory requirement	ADFEMG-151
ADFcg2-04002	New rule recommending against using	ADFEMG-145

	getVendorConnection()	
ADFcg2-03101	New rule stating limitations around applicationScope beans in a cluster	ADFEMG-110
n/a	Updated all footnotes to the latest documentation set where available	n/a

General Code Guidelines

The following code guidelines apply regardless of the project type.

- [ADFcg1-00000] - **All guidelines successfully followed** - this document is full of negatives, don't do this, don't do that. It's nice to start with a positive don't you think? If you comply with all the guidelines in this document you can apply this "success" code.

ADF Application XML Files

The following general conventions apply to ADF Application XML files types:

- [ADFcg1-01000] – **Ensure all XML files are valid** - All XML files used by JDeveloper applications should be complete without any structural errors in the XML.
- [ADFcg3-01001] - **Use declarative editors for modifying XML over source code** - In many cases JDeveloper provides declarative editors for modifying XML files over working with the raw source code. The declarative editors in many cases have additional logic and validation for modifying the underlying settings stored in the XML files which programmers may not be aware of and as such programmers may introduce unexpected issues.

adf-config.xml

The following guidelines apply to the adf-config.xml file:

- [ADFcg2-01033] - **Avoid invalid adf-config.xml files** - Arguably all XML files should be valid but this guideline requires special guidance as learned from internal Oracle teams. They have noted a badly formed adf-config.xml file, potentially with duplicate entries, redundant tags, or invalid namespaces can result in situations that are very hard to debug & diagnose. ADF developers are recommended to carefully check their adf-config.xml file as a result.

adf-settings.xml

There are currently no general guidelines for the adf-settings.xml file.

trinidad-config.xml

The following guidelines apply to the trinidad.xml file:

- [ADFcg1-01002] – **Consider animation-enabled=false for browser performance** - Consider faster browser performance by turning ADF Faces RC animations off. At minimum give each user the ability to determine this themselves.

web.xml

The following guidelines apply to the web.xml file:

- [ADFcg1-01003] – **Add a default activity ID to the welcome-file** - always add a default activity ID from adfc-config.xml file to your web.xml <welcome-file> entry. This becomes the activity the application navigates to when a timeout occurs in a popup or the user opens a new window with Ctrl-n in their browser while in a bounded task flow which isn't a valid starting point for the new window. Note

the entry should be the view activity's JSF viewId from the unbounded task flow, not the relating JSPX page, path and extension.

- [ADFcg1-01004] - **web.xml should not define both the Servlet Filter and the Binding Filter** - For standard ADF applications, only the binding filter is required.

weblogic.xml

The following guidelines apply to the weblogic.xml file:

- [ADFcg1-01005] - **url.rewriting.enabled should be false** - If enabled, the feature would cause security vulnerability because the session identifier would be exposed in certain scenarios.

Deployment

The following general guidelines apply to deployment:

- [ADFcg1-01006] – **Use ADF Libraries appropriately for reusable components** - ADF Libraries provide the best mechanism to ensure the discoverability of metadata based artifacts within a standard JAR format.
- [ADFcg1-01007] - **Procedures exist in the build process to reset development mode flags** - There are many flags in the web.xml file which enable debugging and diagnostic features to aid the developer. However, many of these features are detrimental from the perspective of performance (and security). For more information see Tables 8-1 / 8-2 in the Oracle Fusion Middleware Performance and Tuning Guide.
- [ADFcg1-01008] – **Use ojdeploy for Packaging** - For applications leveraging ADF Security or MDS, this is an important step to ensure that all of the correct metadata is included in the archive.
- [ADFcg1-01009] – **Strip unwanted artifacts (e.g. tests) from Packaging** - Deployable and shipped artifacts should not include test classes, source code or temporary files.
- [ADFcg1-01010] – **Ensure connection information is not deployed with the application** - Connections should be configured by the consumer of the application – usually though the Java EE data source mechanism.
- [ADFcg1-01011] - **Packaged EAR file should include information bearing Manifiest file** - For support purposes

Deployment to Development and Testing Systems

The following guidelines apply to deploying your applications to development or testing environments:

- [ADFcg1-01012] – **Consider web.xml settings for development and testing** – there are a number of web.xml settings that can be set to assist the process of debugging during development and testing:

  oracle.adf.view.rich.ASSERT_ENABLED=true[1]

  org.apache.myfaces.trinidad.DEBUG_JAVASCRIPT=true[2]

  org.apache.myfaces.trinidad.DISABLE_CONTENT_COMPRESSION=true[3]

  oracle.adf.view.rich.LOGGER_LEVEL=true[4]

  Be aware that you might not want to have these set for all testing systems such as user acceptance testing and performance and stress testing, where the perception of speed to your users is paramount.

Deployment to Production Systems

The following guidelines apply to deploying your application to a production environment:

- [ADFcg1-01013] – **Ensure web.xml settings for production** – ensure that the following context parameters are set for a production environment:

  oracle.adf.view.rich.automation.ENABLE=false[5] [default value]

  oracle.adf.view.rich.ASSERT_ENABLED=false[6] [default value]

  org.apache.myfaces.trinidad.CHECK_FILE_MODIFICATION=false[7] [default value]

  org.apache.myfaces.trinidad.COMPRESS_VIEW_STATES=true[8]

---

[1] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 – Section 8.2.1 Oracle ADF Faces Configuration and Profiling - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCIDBIG

[2] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 – Section 8.2.2 Tuning ADF Faces - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCBGHDI

[3] Oracle ADF Web Developer's Guide 12.1.2.0.0 - Section A.2.3.9 Compression for CSS Class Names - http://docs.oracle.com/middleware/1212/adf/ADFUI/ap_config.htm#ADFUI579

[4] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 – Section 8.2.1 Oracle ADF Faces Configuration and Profiling - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCIDBIG

[5] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 – Section 8.2.2 Tuning ADF Faces - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCBGHDI

[6] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 – Section 8.2.1 Oracle ADF Faces Configuration and Profiling - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCIDBIG

[7] Oracle ADF Web Developer's Guide 12.1.2.0.0 - Section A.2.3.2 Debugging - http://docs.oracle.com/middleware/1212/adf/ADFUI/ap_config.htm#ADFUI579

[8] Oracle ADF Web Developer's Guide 12.1.2.0.0 - Section A.2.3.1 State Saving - http://docs.oracle.com/middleware/1212/adf/ADFUI/ap_config.htm#ADFUI579

org.apache.myfaces.trinidad.DEBUG_JAVASCRIPT=false[9]

org.apache.myfaces.trinidad.DISABLE_CONTENT_COMPRESSION=false[10] [default value]

oracle.adf.view.rich.libraryPartitioning.DISABLED=false[11] [default value]

oracle.adf.view.rich.LOGGER_LEVEL=false[12] [default value]

javax.faces.STATE_SAVING_METHOD=client[13]

- [ADFcg1-01014] – **Consider oracle.adf.view.rich.versionString.HIDDEN=false for production** - set the web.xml oracle.adf.view.rich.versionString.HIDDEN context parameter to true to remove the ADF Faces version information on each page to reduce the size of the page and also remove an attack vector by unnecessarily publishing the version of ADF.
- [ADFcg1-01015] – **Consider setting oracle.adf.view.rich.security.FRAME_BUSTING for production** – consider the web.xml oracle.adf.view.rich.security.FRAME_BUSTING context parameters options to prevent clickjacking, which occurs when a malicious web site pulls a page originating from another domain into a frame and overlays it with a counterfeit page.14

Audit

The following guidelines apply to the JDeveloper Audit facility:

- [ADFcg1-01016] – **Use the JDeveloper Audit facility on your workspace** - Always run 'Audit' on the workspace to identify violations of framework rules against your source code. There should be no Audit errors and warnings unless the developer overrides them for a a legitimate reason. Such overrides should be documented.

Java

The following guidelines apply to the Java code within your application:

---

[9] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 – Section 8.2.2 Tuning ADF Faces - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCBGHDI

[10] Oracle ADF Web Developer's Guide 12.1.2.0.0 - Section A.2.3.9 Compression for CSS Class Names - http://docs.oracle.com/middleware/1212/adf/ADFUI/ap_config.htm#ADFUI579

[11] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 – Section 8.2.2 Tuning ADF Faces - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCBGHDI

[12] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 – Section 8.2.1 Oracle ADF Faces Configuration and Profiling - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCIDBIG

[13] Oracle ADF Web Developer's Guide 12.1.2.0.0 - Section A.2.2 What You May Need to Know About Required Elements in web.xml - http://docs.oracle.com/middleware/1212/adf/ADFUI/ap_config.htm#ADFUI578

[14] Oracle ADF Web Developer's Guide 12.1.2.0.0 – Section A.2.3.19 Framebusting http://docs.oracle.com/middleware/1212/adf/ADFUI/ap_config.htm#ADFUI579

- [ADFcg1-01017] - **Don't use internal ADF packages** - Do not use any of the following ADF packages within your Java code as Oracle does not guarantee to change the implementation in the future:
    - oracle.adfinternal
    - oracle.adf.controller.internal
    - org.apache.myfaces.trinidadinternal
    - oracle.webcenter.internal
    - oracle.webcenter.*.internal
- [ADFcg1-01018] - **Don't use System.exit()** - Executing System.exit() in a running ADF application will force the JVM container to quit. Do not use this.
- [ADFcg1-01019] - **Don't hardcode human readable text that will be displayed in the UI** - Do not hardcode human readable text in your Java code or web pages that will be displayed to the user. Rather store any hardcoded text in a resource bundle and refer to the key value-pair instead.
- [ADFcg1-01020] – **Ensure catch blocks do not swallow errors** - Stubbed try/catch blocks that do not re-throw or report exceptions by logging may be concealing bugs. Rather ensure all catch blocks rethrow exceptions so they percolate through the ADF controller or ADF model exception handlers.
- [ADFcg1-01021] - **Include JavaDoc** - provide an appropriate level of documentation through out your code.

Logging

The following guidelines apply to logging within your ADF application:

- [ADFcg1-01022] - **Use the ADFLogger**- Make use of ADFLogger in your ADF applications.
- [ADFcg1-01023] - **Guard log conditions** - The ADF logger allows you to write conditions that test the current log level and therefore only embark on expensive log message creation
- [ADFcg1-01024] - **Log critical errors at SEVERE level** - Most production application servers will restrict logging output to SEVERE only during routine operations. Consider logging errors at SEVERE level. However also consider raising bugs against such SEVERE errors to investigate and fix the reason they occurred in the first place (ie. Logging the error is not enough).
- [ADFcg1-01025] - **Consider logging granularity** - Don't write all log messages at the same logging level Consider what information will be of use for respectively; reporting,error details, live analysis by support and specialized analysis by the development team.
- [ADFcg1-01026] – **Consider user level logging in production** – Enterprise manager now supports user level tracing which allows you to lower the log level for a single session. Traditionally turning on logging on production systems would result in a flood of information for all sessions making the reason to write logging in the first place less than useful. However with the ability to turn logging on for a specific user this makes the implementation of logging more useful.
- [ADFcg1-01027] – **Do not include redundant logging** - Do not over-log. Only emit a log level when it can convey useful information about parameters or results. Logging is not a code coverage tool, and any logging added as debugging aids should be stripped out once the particular problem is resolved.

- [ADFcg1-01028] - **Don't use System.out.* or System.err.*** - Do not use functions like System.out.println.for debugging and do not leave this unnecessarily in your code.

Maven

The following guidelines apply to Maven:

- [ADFcg1-01029] - **Don't override ADF default directories with Maven directories** - For ADF applications based on the Fusion template that also uses Maven, do not attempt to override the default directory locations ADF uses for files such as the /adfmsrc directory for page definitions under the ViewController project.

Source Code

The following guidelines apply to all types of source code used by your ADF applications, including Java, XML, JSPX files etc:

- [ADFcg1-01030] – **Always reformat code** - All code should be reformatted before check in to assist readability.
- [ADFcg1-01031] - **Remove commented out code** - Do not leave commented out code in your source code upon check in.

Source Control

The following guidelines apply to source control for your ADF applications:

- [ADFcg1-01032] - **Ensure compiled and temporary files are not checked into source control** - Ensure files such as those in (WEB-INF/temp, /classes, /deploy) are not checked into source control. By default JDeveloper manages the files to ignore, so let JDeveloper manage the interactions with the source control system.

ADF Business Components

The following guidelines apply to ADF Business Components:

General

The following general guidelines apply to ADF Business Components:

- [ADFcg1-02000] – **Define a jbo.debugoutput run option** - Within the ADF Business Component model project properties define a new Run/Debug/Profile option entitled JboDebugOutputConsole where the Launch Setting → Java options is set to -Djbo.debugoutput=console. Only use this setting for none production systems, it is not supported in production ADF applications.
- [ADFcg1-02001] – **Ensure jbo.debugoutput disabled for production** - Ensure the Java command line -Djbo.debugoutput=console option is not enabled for production ADF systems.
- [ADFcg1-02002] – **Implement an ADF BC extension framework** - Implement an ADF Business Component extensions framework[15]

The following guideline has two camps of thought inside Oracle, those who think ADF BC classes should be implemented, and those who think they shouldn't. Rather than leave the guideline out completely we'll include it here for you to understand the polar arguments and to make a choice yourself:

[ADFcg1-02003a] - **Generate type-safe ADF BC classes** - To assist typesafe programming and monitoring of your ADF Business Components code at runtime via heap monitors, for each entity object, view object and application module do generate the associated EntityImpl, ViewObjectImpl, ViewRowImpl and [AppModuleImpl classes.

vs

[ADFcg1-02003b] – **Do not generate unnecessary ADF BC classes** – Do not generate custom framework extensions for entity objects, view objects and application modules such as custom EntityImpl, EntityCache/CollectionImpl, EntityDefImpl, ViewObjectImpl, ViewRowImpl, EntityDefImpl, AppModuleImpl or AppModuleDefImpl classes unless you specifically need to add a custom code. The unnecessary inclusion of this additional code creates a design time maintenance issue and a runtime overhead.

Note the second version of the rule extends to further classes than the first rule, as the second rule addresses adding custom code which applies to a larger set of classes than those required for type-safe operations.

[15] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 16.2 Creating ADF Business Components Extension Classe - http://docs.oracle.com/middleware/1212/adf/ADFFD/bcadvgen.htm#ADFFD1026

- [ADFcg1-02004] - **Extend custom exceptions with JboException or subclasses** – Generally speaking custom Java exceptions in code should be avoided and applications should rely on declarative validation instead. However if an application does write any programmatic exceptions all custom exceptions written in the ADF BC model layer should extend JboException or a valid subclass such as AttrValException that when raised the error is handled gracefully by the UI layer.
- [ADFcg1-02005] - **Do not import view or controller classes** - The model project should not import or use classes from the view or controller layers such as oracle.adf.controller, oracle.adf.view or org.apache.myfaces.trinidad otherwise they are in violation of the MVC pattern.
- [ADFcg1-02006] - **Don't hardcode human readable text** - Do not hardcode human readable text in your ADF Business Component Java code. Rather store any hardcoded text in a resource bundle and refer to the key value-pair instead.
- [ADFcg1-02007] - **Don't introduce HTTP session dependencies into ADF BC, rely of ADFContext where possible** - ADF BC code should maintain separation and not assume that it is being called from within the context of an ADF Web Application. The HTTP session and other UI layer states should not be accessed directly if you need to use the ADFContext object to access these instead.
- [ADFcg1-02008] - **State required by the service layer should be stored in the service layer** - A common anti-pattern is the storage of information such as user information (e..g preferences, identity etc.) which will ultimately be used to configure the results of queries being stored in the UI layer state. Such state should be pushed into the service layer. Here it can be both pulled into the UI layer if needed, but can also participate in service side queries and operations. Transient VO attributes, programmatic VOs, the UserData map or similar mechanisms can all be used for this.
- [ADFcg1-02009] - **Ensure User Data Hash variables are passivation safe** - Over a passivation-activation cycle variables stored within the ADF BC Session User Data Hash are not passivation safe. As such any variables may be lost. To circumvent this override the passivateState and activateState methods in your application module to store and retrieve the User Data Hash variables over a passivation-activation cycle.
- [ADFcg1-02010] – **Don't define unused declarative ADF BC objects** – By definition a general good programming practice anyway, don't define unused objects in our ADF Business Component projects as they clutter the applications, bloat your code, make it harder to read and debug, and waste space. This included unused nested application modules, application module exposed view object instances,

Application Modules

There are currently no general guidelines for application modules.

AppModuleDefImpl

There are currently no general guidelines for the AppModuleDefImpl.

AppModuleImpl

The following guidelines apply to AppModuleImpl class that can be generated for your application modules:

- [ADFcg1-02011] – **Use view object typesafe getters** - By default the AppModuleImpl contains getter methods to return the associated data model view objects exposed through the application module editor.

Ensure for the relating view objects that you've first generated the associated ViewObjectImpl class and that the getter method in the AppModuleImpl returns the typesafe class (e.g CustomerViewImpl) rather than the parent ViewObjectImpl.

DBTransactionImpl

The following guidelines apply to the DBTransactionImpl that can be accessed via the ApplicationModuleImpl:

- [ADFcg1-02012] - **Close callableStatement and preparedStatements** - Via the ApplicationModuleImpl you can retrieve a DBTransactionImpl to call createCallableStatement or createPreparedStatements. These both return JDBC objects which require you to close the statements once complete. This requires a try/catch/final block to ensure the JDBC connection is closed at the end of its use.
- [ADFcg1-02013] - **Carefully consider using transactions within overridden ADF BC code** - Besides custom methods, generally speaking ADF BC manages user transactions on the developers behalf. Any usage of get(DB)Transaction() .commit() or rollback() should be considered very carefully based on the possible side effects it may have to open transactions within the AM scope.

Client Interface Methods

The following guidelines apply to application module client interface methods:

- [ADFcg1-02014] – **Use the application module client interface where appropriate** - Reserve client interface methods at the application module level for methods that work across view objects and their rows.

Configuration

The following guidelines apply to application module configuration:

- [ADFcg1-02015] – **Use optimistic locking** - Only use optimistic locking (the default) for web applications, rely on the global setting in the adf-config.xml file to set this.[16]
- [ADFcg1-02016] – **Use application module JDBC data source only** - Only use JDBC data sources for application module configurations, do not use JDBC URLs.[17]
- [ADFcg1-02017] – **Develop and test with Enable Application Module pooling off** – During development and testing test with Enable Application Module pooling off, though remember for user acceptance testing and load and stress testing where performance is critical turn this option back on as it affects performance. Even if you're no longer developing or testing your system, consider periodically

[16] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 49.11.1 How to Confirm That Applications Use Optimistic Locking - http://docs.oracle.com/middleware/1212/adf/ADFFD/bcstatemgmt.htm#ADFFD1330

[17] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 13.3.1 How to Use a JDBC Data Source Connection Type - http://docs.oracle.com/middleware/1212/adf/ADFFD/bcservices.htm#sthref368

complete a round of testing with jbo.ampool.doampooling turned off via your Continuous Integration server to ensure your application remains activation safe.[18]

- [ADFcg1-02018] - **Run with Enable Application Module pooling on for production** - in a production environment jbo.ampool.doampooling must be turned on to be supported by Oracle.[19]

Nested Application Modules

There are currently no defined guidelines for nested application modules.

ps_txn and ps_txn_seq

The following guidelines apply to the database ps_txn table and ps_txn_seq sequence:

- [ADFcg2-02069] – **Optimize the ps_txn table definition** - When creating the ps_txn table, use securefiles to store LOB data for the content column, and create a primary column index on the ps_txn table as global, partitioned reverse key index. The securefile configuration delivers superior performance over the basicfile configuration when working with LOB data. The reverse key index helps by reducing contention that can happen when the rate of inserts is high.[20]
- [ADFcg2-02070] – **Consider using an alternative schema for jbo.server.internal_connection** - Since the framework creates temporary tables such as px_txn in the database, the implication of not setting a value for the jbo.server.internal_connection is that the current database user must have CREATE TABLE, CREATE INDEX and CREATE SEQUENCE privileges. Since this is often not desirable, it is recommended to supply an appropriate value for the jbo.server.internal_connection property, providing the credentials for a separate database schema where the object may be created.[21]

Service Interface

The following guidelines apply to application module service interfaces:

- [ADFcg1-02019] - Avoid circular dependencies in view objects when defining your service interface - When creating service interfaces ensure no circular dependencies exist between view objects. For example using the HR schema, the department view typically is referenced by the employee view in the departmentId attribute while the department managerId does reference the employee view. These circular dependencies should be eliminated.
- [ADFcg1-02020] - **Don't mix ADF BC web model projects with service interface projects** - Build Service interfaces in their own model project and ensure code reuse by shared ADF libraries

---

[18] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 49.2.2.3.1 About Managed Release Level http://docs.oracle.com/middleware/1212/adf/ADFFD/bcstatemgmt.htm#sthref940

[19] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 49.2.2.3.1 About Managed Release Level http://docs.oracle.com/middleware/1212/adf/ADFFD/bcstatemgmt.htm#sthref940

[20] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 – Section 49.5.2.2 Controlling the Schema Where the State Management Table Resides http://bit.ly/adfdevguide1212sect49522

[21] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 – Section 49.5.2.2 Controlling the Schema Where the State Management Table Resides http://bit.ly/adfdevguide1212sect49522

Shared Application Modules

There are no guidelines yet defined for shared application modules.

Testing

The following guidelines apply to application module testing.

- [ADFcg1-02021] – **Use the ADF Business Component tester** - The IDE comes with a testing tool for testing your ADF Business Components objects. It's recommend you test your ADF Business Components and any custom functionality you add separate from the ViewController. This will help you isolate if problems reside in your Model layer or your ViewController.
- [ADFcg1-02022] – **Consider testing for HA even if you won't use it** – While developing and testing systems, even if the system is not designed to be used in a high availability environment, consider testing with the HA options on (but turn them off during user acceptance testing, load and stress testing and production systems). This is because it's much easier to build and test the system with the HA options turned on, then to retrospectively turn them on if you decide your application needs to move to a HA environment. Consider Oracle Support Note 1468116.1 for what to set for a high available system.

Tuning

There are no guidelines yet defined for application module tuning.

Domains

There are no guidelines yet defined for domains.

Entity Associations

There are no general guidelines yet defined for entity associations.

Entity Objects

The following section outlines general guidelines that apply to entity objects:

- [ADFcg1-02023] – **Implement entity object sequences consistently** - There are more than 1 way to populate entity object attributes, including with database sequences, via the DBSequence domain type and database triggers, overriding the EntityImpl create() method, or using Groovy expressions. For your overall application decide the technique to use, apply it consistently across all entity objects and apply it as soon as the entity object is created.
- [ADFcg1-02024] – **Do not access an application module from an entity object** - entity objects should not access application modules as this makes the entity object only useable within that application module.
- [ADFcg1-02025] – **Define at least one entity object key attribute per entity object** - An entity object must has at least one attribute defined as the key.
- [ADFcg1-02026] – **Use a surrogate sequence generated primary key over natural keys** – While ADF BC applications can be built using natural keys on tables, generally speaking you will have more problems implementing this approach and dealing with validation. Instead use surrogate sequence generated primary keys. If you have a legacy data model with natural keys, consider creating a surrogate sequence generated primary key and changing the original natural primary key to a unique key instead.

- [ADFcg1-02027] - **Don't use entity object key ROWID** - By default if ADF BC doesn't find a primary key at design time for an entity object's relating database table, it will generate an additional attribute mapping to ROWID. This will continue to work unless an Oracle database Real Application Cluster (RAC) is used where across database partitions different ROWIDs can be returned for the same row. Therefore attempt to avoid using the ROWID attribute and instead select a combination of existing attributes as the primary key.

View Accessors

There are no guidelines yet defined for entity object view accessors.

Alternate Keys

There are no guidelines yet defined for entity object alternate keys.

Attributes

The following guidelines apply to entity object attributes:

- [ADFcg1-02028] – **Use entity object history columns where appropriate** - When creating an entity object based on a table with columns such as created_by, created_date, updated_by and updated_date, set the relating attribute history column properties. Be mindful for existing legacy systems with database triggers these may be populated for the entity, as such don't use these options but rather the entity attribute Refresh After Insert an Refresh After Update properties.
- [ADFcg2-02029] – **Use entity object attribute change indicators** - When an entity object is updated concurrently by more than one user and one of those users loses affinity with their application module during the update, it is possible to experience data corruption if a change indicator has not been defined for the entity object. To avoid this type of scenario, define a change indicator attribute for all entity objects.[22]
- [ADFcg1-02030] – **Define entity object attribute labels for UI displayed attributes** - For all entity object attributes that are displayed in the UI set their default label.
- [ADFcg1-02031] - **Define entity object attribute tooltips for UI displayed attributes where an explanation of the allowable values is required** - For all entity object attributes that are displayed in the UI, where the attribute has a set of allowable value that isn't obvious, include a one sentence tooltip to describe this.
- [ADFcg1-02032] – **Ensure entity object UI Hints and messages are internationalized** - Assuming that the application will be translated all entity object field labels and other relevant UI hints and messages need to be managed via message bundles.

Business Logic Groups

There are no guidelines yet defined for entity object business logic groups.

---

[22] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 – Section 49.2.3 What You May Need to Know About Data Consistency and Application Module Affinity http://bit.ly/adfdevguide1212sect4923

Business Rules/Validations

The following guidelines apply to entity object business rules:

- [ADFcg1-02033] – **Avoid entity object attribute validation in accessors** - do not write code to validate an attribute in its setter accessor within the associated EntityImpl class for the entity object. Rather create a declarative method validator for the attribute. This comes with the added advantage that all validations are visible in the entity object's business rules page and have all the features available to the declarative validators, as well as making the method reusable by other similar attributes.[23]
- [ADFcg1-02034] – **Avoid using the entity object validateEntity() method for validation** - do not override an entity object's EntityImpl validateEntity() method. Rather code any business logic as a declarative method validator for the entity object itself. This comes with the added advantage all validations are visible in the entity object's business rules page and have all the features available to the declarative validators.[24]
- [ADFcg1-02035] - **Create error messages for entity object mandatory and precision validators** - The default error messages for the pregenerated mandatory and precision validators can be overridden with clearer messages. These are listed under the business rules of each individual entity object attribute.

EntityCache/CollectionImpl

There are currently no guidelines that apply to the EntityCache/CollectionImpl.

EntityDefImpl

- There are currently no guidelines that apply to the EntityDefImpl.

EntityImpl

The following section outlines general guidelines that apply to the EntityImpl:

- [ADFcg1-02036] – **Do not call postChanges unless you can guarantee a commit or rollback concludes the HTTP request** – Typically the EntityImpl postChanges method is called by the ADF BC lifecycle upon a commit operation. However you can also programmatically call this method. As the postChanges method changes the state of the associated EntityImpl which then assumes the changes have been committed to the database, this can leave your entity object in a dirty state. As such only call the postChanges method directly if you know that at the end of the HTTP request you will commit or rollback the transaction.

Security

There are no guidelines yet defined for entity object security.

---

[23] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section D.3.4 EntityImpl Class - http://docs.oracle.com/middleware/1212/adf/ADFFD/appendix_mostcommon.htm#ADFFD1472

[24] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section D.3.4 EntityImpl Class - http://docs.oracle.com/middleware/1212/adf/ADFFD/appendix_mostcommon.htm#ADFFD1472

Tuning

There are no guidelines yet defined for entity object tuning.

View Accessors

The following guidelines apply to entity object view accessors:

- [ADFcg-02037] – **Implement entity object accessor tuning** - Ensure to set the view object exposed through the view accessor tuning options appropriate to how the view object will be used.

Groovy

There are no guidelines yet defined for Groovy.

Passivation and Activation

The following guidelines apply to ADF BC passivation and Activation:

- [ADFcg1-02038] – **Ensure custom AppModuleImpl and ViewObjectImpl instance variables are passivation safe** – Over a passivation-activation cycle a new instance of custom AppModuleImpl, and/or ViewObjectImpl classes may be used by the application module pool on the session's behalf. As such any instance variables defined in these custom classes may be lost. To circumvent this override the passivateState and activateState methods to successfully store and retrieve these values of a passivation-activation cycle.
- [ADFcg1-02039] – **Ensure customer ViewRowImpl and EntityImpl instance variables passivation safe** - Over a passivation-activation cycle a new instance of custom ViewRowImpl and/or EntityImpl may be used by the application module pool on the session's behalf. As such any instance variables defined in these custom classes may be lost. To circumvent this store the values as transient attributes on the view object or entity object instead.
- [ADFcg1-02040] - Passivate non-calculated transient attributes - For a view object containing any non-calculated transient attributes select both the "Passivate State" and "Include All Transient Values" checkboxes to ensure that under high load transient attribute value survives a passivate/activation cycle.

Property Sets

There are no guidelines yet defined for property sets.

View Links

There are no guidelines yet defined for view links.

View Objects

The following general guidelines apply to view objects:

- [ADFcg1-02041] – **Avoid read only view objects for speed benefits alone** - Older ADF manuals recommended using read only view objects as there was a minor speed benefit. Since JDeveloper 11g this is no longer true as the caching mechanism has been improved. Today read only view objects should be reserved for complex queries such as those with set operations (e.g. UNION, MINUS, INTERSECT) and connect by queries. All other view objects should be based on entity objects. If the view object

needs to be read only, unselect the "Updateable" property when the entity object is mapped to view objects.

- [ADFcg1-02042] **- Use declarative SQL view objects where possible** - Declarative SQL based view objects selectively prune the attributes included in the underlying query based on the attributes accessed via the binding and UI layers. This results in faster turn around with the database and less middle tier memory consumed.
- [ADFcg1-02043] – **Rarely use view object expert mode** - Rarely use expert mode for view objects. If the query you wish to execute can be implemented by modifying other properties of the view object over expert mode this should be your preference. For example SQL functions on attributes can be applied at the attribute level.
- [ADFcg1-02044] – **Define at least one view object key attribute** - A view object must has at least one attribute defined as the key. This is particularly important for view objects that are marked for passivation as the key is required to restore the current row on activation.
- [ADFcg1-02045] - **View objects should include all key attributes from underlying entity objects** - If a view object is based on one or more entity objects, then view object should include all the key attributes from each entity object.
- [ADFcg1-02046] – **Avoid unnecessary view object duplication** - In many cases very similar view object instances can be created from the same base definition with the correct use of applied view criteria when the instance is exposed via the application module.

Alternate Keys

There are no guidelines yet defined for view object alternate keys.

Attributes

The following guidelines apply to view object attributes:

- [ADFcg1-02047] – **Ensure view object UI Hints and messages internationalized** - Assuming that the application will be translated all view object field labels and other relevant UI hints and messages need to be managed via message bundles.

Client Interface and Client Row Interface Methods

The following guidelines apply to application module client interface methods:

- [ADFcg1-02048] – **Use view object client interface where appropriate** - Reserve client interface methods at the view object level for methods that work across rows in the defined view object.
- [ADFcg1-02049] – **Use view object client row interface where appropriate** - Reserve client row interface methods at the view object level for methods that work on one row at a time defined in the same view object.

List of Values

The following guidelines apply to list of values with view objects:

- [ADFcg1-02050] - **Don't define circular list of value dependencies** - View objects allow you to define dependent/chained list of values. Be careful not to define a circular dependency as this can result in an infinite loop.
- [ADFcg2-02067] - **Avoid programmatically calling excessive setAttribute calls on LOV attributes, use setAttributeValues() instead** - Programmatically it is possible for an application to make multiple setAttribute() calls on the same view object row. If the row has associated list of values such that one or more of the setAttribute() calls will impact the LOVs (ie. force the LOV accessors to refresh and/or LOVs to derive values), this can have a runtime impact, causing the LOV logic to run multiple times. In addition depending on the order of the setAttribute() calls, you can get different behaviours based on when LOV runs when. To get a consistent and performant result, it is recommended the disparate setAttribute() calls be bundled into one setAttributeValues() call. This will ensure the LOV logic runs once for all of these attributes and the LOVs run in the order they are listed in the VO xml.

Programmatic

The following guidelines apply to programmatically working with view objects:

- [ADFcg1-02051] **- Use createRowSetIterator() for view object programmatic iteration** - if in code you need to programmatically iterate over the rows provided by a view object, use the createRowSetIterator() and closeRowSetIterator() methods to manage a separate RowSetIterator to the default provided by the view object, as the default may be bound to the user interface.[25]
- [ADFcg1-02052] – **Use database filtering first before view object RowMatch** - RowMatch performs filtering of records in memory and as a result must query all records to the midtier first. To speed the performance of RowMatch use database-level filtering to retrieve the smallest-possible rowset first, and then using RowMatch as appropriate to subset that list in memory.[26]

Queries

The following guidelines apply to view object queries:

- [ADFcg1-02053] – **Use view criteria over editing view object where clauses** - Rather than customizing a view objects where clause which makes the use of the view object less flexible, use a view criteria instead.
- [ADFcg1-02054] – **Never use view object select *** - Never use SELECT * for a view object query. Always specify all columns in the query.
- [ADFcg1-02055] – **Avoid hardcoding view object queries with literals** - Don't hardcode view object where clauses or view criteria with literals. Always use bind variables.

---

[25] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 13.7.6 What You May Need to Know About Programmatic Row Set Iteration - http://docs.oracle.com/middleware/1212/adf/ADFFD/bcservices.htm#sthref386

[26] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 10.3.4.3 How a RowMatch Affects Rows Fetched from the Database - http://docs.oracle.com/middleware/1212/adf/ADFFD/bcqueryprog.htm#ADFFD1228

- [ADFcg1-02056] – **Use view object bind variables over dynamic SQL** - Bind variables also make the queries of view objects less susceptible to SQL injection attacks.
- [ADFcg1-02057] – **Do not allow unfiltered user input for query creation or modification or JDBC prepared statements** - Creating or modifying view objects at runtime based on values input from the user (either though the UI or some other mechanism) provides a potential risk point for SQL Injection as well as performance issues, including DOS attacks

Tuning

The following guidelines apply to view object tuning:

- [ADFcg1-02058] – **Consider view object fetch size and access mode** - Consider the view object tuning options for how the view object will be used. If the view object is used to display a page of 10 records for example, set the Batch/Fetch size of the view object to 10 + 1 too.[27] Also consider how the user will progress through the data and make use of the appropriate access mode to restrict memory usage.
- [ADFcg1-02059] - **View object tuning location** - Tuning options for each view object can be set either within the view object or when the view object is exposed through the application module or used as an accessor for an entity object or view object. It's recommend you always override the tuning options at the view object defining the best "general" tuning options, then override the tuning options at the application module or accessors level for their specific use case.
- [ADFcg1-02060] – **Check view object SQL performance** - Ensure to check the performance in the database of the underlying view object queries, especially critical for search forms. If need be seek guidance from your DBA on creating the appropriate database indexes for the search criteria.

ViewDefImpl

The following section outlines general guidelines that apply to the ViewDefImpl:

- [ADFcg1-02061] – **Do not generate a ViewDefImpl class** - Do not generate the ViewDefImpl class for your view object unless needed.

ViewObjectImpl

The following section outlines general guidelines that apply to the ViewObjectImpl:

- [ADFcg1-02062] - **Use type-safe bind variable getter/setter accessors** - If accessing bind variable for the view object, always use the type-safe getter and setter methods provided by the ViewObjectImpl over the generic getAttribute and setAttribute methods.

ViewRowImpl

[27] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 8.3.12 What You May Need to Know About Optimizing View Object Runtime Performance - http://docs.oracle.com/middleware/1212/adf/ADFFD/bcqueryresults.htm#ADFFD19565

The following section outlines general guidelines that apply to the ViewRowImpl:

- [ADFcg1-02063] **- Use type-safe view object getter/setter accessors** - If accessing any attribute for the view object, always use the type-safe getter and setter methods provided by the ViewRowImpl over the generic getAttribute and setAttribute methods.

View Accessors

The following section outlines guidelines for view object view accessors:

- [ADFcg-02064] – **Ensure to use view object accessor tuning** - ensure to set the view object exposed through the view accessor tuning options appropriate to how the view object will be used.

View Criteria

The following guidelines apply to view object view criteria:

- [ADFcg1-02065] – **Avoid case insensitive view criteria if not needed** - For view objects using view criteria, ensure to uncheck the case insensitive option if it not needed as it will make database queries slower unless an Oracle RDBMS function based index is available.
- [ADFcg2-02066] - **Always conduct a security code review on calls to applyViewCriteria, setApplyViewCriteria** - Calling any of the following methods in ViewObjectImpl or ViewCriteriaManager will clear the list of applied view criteria on a view object.
    1. applyViewCriteria(ViewCriteria)[28]
    2. applyViewCriteria(ViewCriteria, boolean)[29] with false in 2nd parameter value
    3. setApplyViewCriteriaName(String)[30]
    4. setApplyViewCriteriaName(String, boolean)[31] with false in 2nd parameter value
    5. setApplyViewCriteriaNames(String[])[32]

  This can have an impact if the already applied view criteria are required for data security requirements. Therefore it is important teams during their code review process investigate any calls to these methods and ensure they are still meeting any security requirements.
- [ADFcg2-02068] - **Do not reuse programmatic view criteria between view objects** - when programmatically creating a view criteria and applying it to a view object instance, the criteria keeps an internal pointer to the view object instance and therefore it cannot be used by another view object instance, even if it is of the same type. Therefore the following code is illegal:

[28] See applyViewCriteria(ViewCriteria) JavaDoc: http://bit.ly/1ce6JvS
[29] See applyViewCriteria(ViewCriteria, Boolean) JavaDoc: http://bit.ly/1ceyO2Z
[30] See setApplyViewCriteriaName(String) JavaDoc: http://bit.ly/1dhjmHN
[31] See setApplyViewCriteriaName(String, Boolean) JavaDoc: http://bit.ly/1eN4zbd
[32] See setApplyViewCriteriaNames(String[]) JavaDoc: http://bit.ly/Kd72yc

```
vc1 = vo1.createViewCriteria();
vo2.applyViewCriteria(vc1);
```

Instead replace it with code like this:

```
vc1 = vo1.createViewCriteria();
vc2 = vo2.createViewCriteria();
vc2.copyFrom(vc1);
vo2.applyViewCriteria(vc2);
```

ViewController

The following guidelines apply to objects in the ViewController project:

JavaServer Faces Constructs

The following sections relate to JavaServer Faces constructs such as the faces-confix.xml file, beans and more.

Beans

The following guidelines apply to beans:

- [ADFcg1-03000] – **Limit custom code in bean accessors** - As the JSF lifecycle may call the getter and setter method in your beans multiple times, keep the code as small as possible to avoid unnecessary processing.
- [ADFcg1-03001] – **Use beans only defined in the task flow for reuse** - To make them reusable, task flows should not reference managed beans that weren't defined within the task flow.
- [ADFcg1-03002] – **Avoid applicationScope and sessionScope variables** - Use applicationScope and sessionScoped variables with care as they create a tight coupling and unintended dependencies within your code.[33]
- [ADFcg1-03003] **- Use unbounded task flow scope pageFlowScope variables over sessionScope to support multi browser tabs** - ADF supports users opening multiple browser tabs onto your application. From the user's perspective these are two separate connections to the application, and as such the state for each should be separate. Variables stored in sessionScope will be shared across both browser tabs as only one instance of the sessionScope bean will be instantiated. Alternatively for pageFlowScope beans one is instantiated per browser tab.[34]
- [ADFcg1-03004] – **Define applicationScope and sessionScope beans in adfc-config.xml** - applicationScope and sessionScope managed beans should be defined in the adfc-config.xml file.
- [ADFcg1-03005] – **Define unbounded task flow beans in adfc-config.xml** - For any page or page fragment used by an application's unbounded task flow, any relating bean used by the page should also be defined in the adfc-config.xml.
- [ADFcg1-03006] – **Define bounded task flow bean in their own .xml file** - For any page or page fragment used by a bounded task flow, any relating managed bean used by the page should also be defined in the bounded task flow's .xml file.
- [ADFcg1-03007] – **Implement bean serialization for applicationScope, sessionScope, pageFlowScope and viewScope beans** - for all applicationScope, sessionScope and viewScope beans ensure they implement the serializable interface.

---

[33] Task Flow Design Fundamentals - Section Session Scoped Variables - http://bit.ly/adftaskflowfund

[34] Blog: Chris Muir - http://bit.ly/pageFlowScopeMultiTab

- [ADFcg1-03008] - **Avoid business logic in bean methods** - complex business logic that manipulates model layer data should not be placed in the ViewController layer. Consider placing such code in the model layer instead and making one programmatic call from the bean.
- [ADFcg1-03009] – **Store UI components in requestScope and backingBeanScope beans** - only store references to UI components in requestScope and backingBeanScope beans. Never store references to UI components in applicationScope, sessionScope, viewScope or pageFlowScope beans.[35]
- [ADFcg1-03010] – **Store UI component state in sessionScope, pageFlowScope or viewScope beans** - if you're not using ADF bindings to store and retrieve the values for an UI component, but rather are storing the state in managed beans, ensure the values are stored in viewScope, pageFlowScope or sessionScoped beans.
- [ADFcg1-03011] - **Backing beans for pages and page fragments should be declared as requestScope and backingBeanScope respectively** - Backing beans should only be holding references to UI objects and event handlers for specific components on a screen. UI component references in particular are not serializable and are not necessarily stable between requests. Therefore these beans should not live longer than those references.
- [ADFcg1-03012] - **Variables in applicationScope and sessionScope beans** - Be mindful of the variables you store in applicationScope and sessionScope as they may be stored there for longer than necessary.[36]
- [ADFcg1-03013] - **Only create backing beans when actually needed, do not bind unnecessary component references into backing beans** - You only need a Backing Bean when you actually need somewhere to hold event handler code. If you have no such code requirements, don't bother to create the bean.

  A careless selection in the design time can lead to references to all UI components within a view being created as java objects in the associated backing bean. This level of association is never needed. Only create UIComponent references when they are actually needed for things such as selection management (tables and trees) or programmatic PPR operations.
- [ADFcg1-03014] - **Define all managed beans explicitly in XML rather than through a mix of XML and annotations** - As of JSF2 we can use annotations in code to define classes as managed beans in the standard JSF scopes. However, such annotations do not exist for the ADF specific scopes and mixing both techniques may make it harder to track down the definition when maintaining the code.
- [ADFcg1-03015] - **Don't hardcode human readable text** - Do not hardcode human readable text in your beans. Rather store any hardcoded text in a resource bundle and refer to the key value-pair instead.

---

[35] Blog: Blake Sullivan - http://bit.ly/beansuicomponents

[36] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 27.3.1 What You May Need to Know About Object Scopes and Task Flows - http://docs.oracle.com/middleware/1212/adf/ADFFD/adf_lifecycle.htm#ADFFD19606

- [ADFcg1-03016] - **Utilize common utility class or Backing Bean superclass for common functions** - Utility Methods are often reused within multiple backing beans and a common provider class can save a lot of time and coding mistakes
- [ADFcg2-03101] – **Don't rely on state stored in applicationScope beans in a cluster** - JSF Application Scope beans are backed by a Servlet Context which can only have one instance per cluster node and there is no synchronization across nodes. The implication is for applicationScope managed beans, that for a single design time configured bean, there will be at runtime as many instances of the bean as you have nodes in the cluster. Each instance of your application running on each node therefore when accessing the applicationScope bean is indeed getting separate values. This makes using the applicationScope beans inadequate for carrying application "wide" values to be shared across all nodes. Rather the bean is probably best for carrying constants or values to reflect the state of a single node (such as number of users on a single node).[37]

Expression language (EL)

The following guidelines apply to expression language (EL):

- [ADFcg1-03017] – **Avoid hardcoding EL literals** - Avoid hardcoding literals in EL expressions. Instead place the hardcoding in a managed bean and refer to it through a boolean accessor instead. For example the EL expression #{bindings.Status.inputValue == 'Updateable'} could be replaced with #{editBookings.updateable}.
- [ADFcg1-03018] – **Avoid multi-expression EL boolean conditions** - As custom EL expressions tend to be repeated down a page or page fragment, any change to the expression logic must be changed in multiple places. To reduce the cost of change restrict EL boolean conditions to two at most. If more than two place the expression in a bean boolean accessor instead.
- [ADFcg1-03019] – **Avoid repeating EL conditions** - If there is a common EL condition used through out a page consider placing the logic in a bean boolean accessor instead.
- [ADFcg1-03020] - **Do not prefix EL references to managed beans in request, session or application scope with the *Scope prefix** - Although it seems to be a good thing to be explicit about the scope that you expect such an object to be in, doing so will bypass the JSF creation routines if the bean does not already exist. This is not an issue for the ADF specific scopes.
- [ADFcg1-03021] - **Restrict the use of EL evaluation in code** - Evaluating expression language within (and only within) Java Code can be error prone and requires a context switch which is inherently slow. It is preferable to inject EL values as managed properties to a bean and therefore allow the design time to validate the reference, or if the values must be determined at the point the code is evaluated, access the values via Java context objects or other Java constructs.

faces-config.xml

The following guidelines apply to the faces-config.xml file:

[37]Oracle ADF Fusion Developer's Guide 12.1.2.0.0 – Section 49.2.4 What You May Need to Know About Using Application Scoped Managed Beans in a Clustered Environment http://bit.ly/adfdevguide1212sect4924

- [ADFcg1-03022] - **Do not define beans in the faces-config.xml file** - for ADF Faces projects managed beans should no longer be configured in the faces-config.xml file.[38]
- [ADFcg1-03023] - **Do not define pages in the faces-config.xml file** - for ADF Faces projects pages should no longer be configured in the faces-config.xml file.

Bindings

The following general guidelines apply to bindings:

- [ADFcg1-03024] - **Only interact with the business model through the binding layer** - Accessing the underlying business model (e.g. an application module) via the dataprovider can have multiple negative effects such as causing memory and connection leaks, getting the wrong sessions data and breaking the auto PPR functionality.
- [ADFcg1-03025] – **Only interact with ADF BC custom methods through the client interface exposed methods** – ADF BC application module, view objects and view object rows provide the ability to expose custom methods to the view layer to be called. Make use of these via the binding layer rather than accessing the method by programmatically retrieving the ADF BC objects via the binding layer.
- [ADFc1-03026] - **In Method activities and managed beans bind data explicitly from the current binding container** - As in the use of #{data}, reaching out to other binding containers from the BindingContext via findBindingContainer() is not reliable. Required data/operations/methods should be defined explicitly in the correct pageDef context.

Binding Container

The following guidelines apply to the binding container:

- [ADFcg1-03027] - **Don't cache the BindingContainer -** Ensures that you don't cause a memory leak in the application by pinning state. Always grab the BindingContainer as you need it.
- [ADFcg1-03028] - **Don't reference #{data}** - The #{data} expression root is not guaranteed to provide access to the required data. Success will depend on the state of the cache. If data is needed in a particular view / method then create an explicit binding in that scope.

Binding Context

The following guidelines apply to the binding context:

- [ADFcg1-03029] - **Avoid accessing the binding context** - In most cases applications should expose model objects through the "bindings" which implies the BindingContainer. The need to access the binding context should be a rare requirement.

---

[38] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 26.4 Using a Managed Bean in a Fusion Web Application - http://docs.oracle.com/middleware/1212/adf/ADFFD/web_getstarted.htm#ADFFD1745

- [ADFcg1-03030] - **Use the correct method for obtaining the BindingContext** - Various legacy techniques for getting the BindingContext exist. Ensure that you are using the correct approach of the getCurrent() factory method.
- [ADFcg1-03031] - **Don't cache the BindingContext -** Ensures that you don't cause a memory leak in the application by pinning state. Always grab the BindingContext as you need it.

DataBindings.cpx

The following guidelines apply to the DataBindings.cpx file:

- [ADFcg1-03032] - **Ensure there are no errors in DataBindings.cpx file** - This would be indicative of an incomplete re-factoring exercise

LOVs

The following guidelines apply to LOV bindings:

- [ADFcg1-03033] - **Use ADF BC LOVs over binding layer LOVs** - The ADF binding layer allows you to construct LOVs for select components based on static lists and dynamic lists deriving their data from alternative iterators. This facility is a left over from earlier JDeveloper releases and if you're using ADF Business Components you should instead make use of the ADF BC declarative LOV features provided out of the box.

Task Flows

- The following guidelines apply to bindings used by task flows:
- [ADFcg1-03034] - **Use bound methods on task flows rather than invokeAction executables in the pageDef** - Although the use of invokeAction is allowed it is generally much clearer to use a method activity in a page flow to explicitly show when code is executed before a page or fragment is displayed. Also the invokeAction mechanism has no way to cleanly execute associated code once before the page renders, using the method activity provides you with the cleanest interface for implementing this logic. Exception: ADF Mobile AMX page bindings.

Components

The following guidelines apply to ADF Faces RC components:

- [ADFcg1-03035] - **Don't hold UIComponent references outside of backingBeanScope or requestScope beans** - UIComponent references are not guaranteed to be stable between requests and are not serializable. At best, holding such references will lead to memory leaks, at worst you may start to get errors in high transaction environments upon failover. Always store such references in either backingBeanScope beans for page fragments or requestScope beans for pages.
- [ADFcg1-03036] – **Consider command partialSubmit=true** - Command components such as af:commandButton by default when pressed cause a complete page refresh with the server. If not

navigating between pages, consider using partialSubmit=true which will result in only components with their partialTriggers set to be processed.

- [ADFcg1-03037] – **Avoid long component ID lengths** - Avoid long component ID names as this has a performance hit.[39]
- [ADFcg1-03038] - **Abbreviate "container" component ID lengths to 4 characters** - The number 4 here is arbitrary. However for container component, that is components that contain other components, as their ID gets repeated multiple times in all the child component IDs, the smaller the ID you use here the smaller and more efficient the page will be.
- [ADFcg1-03039] - **Avoid unnecessary use of clientComponent=true** - Setting clientComponent=true on any component results in a larger DOM structure on the browser which will have a small impact hit both terms of speed and memory usage. Only use clientComponent=true when necessary.[40]
- [ADFcg1-03040] - **Use rendered=false over visible=false if possible** - If you wish to hide a component use rendered=false over visible=false when possible as the first results in the server skipping the processing of the component and sending it over the wire to the browser, while the 2nd requires the server to process the component, send it over the wire, the browser to process it then hide it.[41]
- [ADFcg1-03041] – **Consider that visible=false presents a security risk for components** - As components with visible=false are still sent to the client then hidden, it's easy for hackers to see them in the browser DOM, manipulate them and if they're submittable components send them on the next request. Instead either use rendered=false so the component isn't sent to the browser in the first place, or back the submittable component by a validation rule to ensure it can't be submitted when the component is meant to be hidden.
- [ADFcg1-03042] - **Never rely on UI side validation** - Never solely rely on UI (browser) validation alone as this can be easily exploited by hackers. As example a selectOneChoice on the browser provides an allowable list of values the user can select from, but doesn't validate that the user hasn't submitted a different value not in the list. Always back UI side validation up with server side validation to catch such workarounds.
- [ADFcg1-03043] - **Keep naming container nesting to a minimum** - Very deeply nested naming container hierarchies in the page can be very hard to work with from the layout / maintenance point of view and also have a performance penalty as the component IDs are prefixed with the naming container ID, make the requests send to the server larger. . Aim to simplify your layouts if you can.

---

[39] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 - Section 8.2.3 Tuning ADF Faces Component Attributes - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCHGGJG

[40] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 - Section 8.2.3 Tuning ADF Faces Component Attributes - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCHGGJG

[41] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 - Section 8.2.3 Tuning ADF Faces Component Attributes - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCHGGJG

- [ADFcg1-03044] - **Only use ADF Faces, DVT, JSF Core and Trinidad HTML components** - In JSF 1.n there is limited co-existence capability between component sets particularly when it comes to the lifecycles of AJAX transactions. Use of other component sets should be by exception only.

Problematic Components

The following guidelines apply various JSF, JSP and ADF Faces components that should be used with caution:

- [ADFcg1-03045] - **Avoid using the af:inlineFrame tag** - Inline frames may introduce problems both with geometry management and performance (especially on mobile browsers).
- [ADFcg1-03046] - **Avoid using the f:verbatim tag** - The use of verbatim is often linked to other known issues such as the use of embedded HTML and generally should be avoided. For scenarios such as embedding applets or other embedded technologies, consider creating a custom component wrapper.
- [ADFcg1-03047] - **Do not use the jsp:include tag** - As well as tying the page to a particular page assembly technology, JSP Scriptlets do not fit in with any of the lifecycle aspects of ADF or JSF. Use method activities in ADF Task Flows or lifecycle annotations (in JSF 2.0) for these purposes. Pages with JSP includes will not be upgradable to use Facelets. Instead use the af:declarativeComponent tag.[42]
- [ADFcg1-03048] - **Do not use the jsp:scriptlet tag** - JSP Includes tie the page to a particular page assembly provider. Use ADF Taskflows or declarative components instead. Pages with JSP includes will not be upgradable to use Facelets.

Data Controls

The following general guidelines apply to data controls:

- [ADFcg1-03049] - **No use of PlaceHolder data control** - The placeholder data control is intended for use during prototyping and mockups only

Declarative Components

There are no currently defined rules for declarative components.

JavaScript

The following guidelines apply to JavaScript:

- [ADFcg1-03050] **- Avoid inline page/fragment JavaScript and CSS** - Avoid placing custom JavaScript and CSS in your page and page fragments as these must be loaded by the browser each time the page/fragment is accessed. Instead place the JavaScript and CSS respectively in separate .JS and .CSS files so they can be cached by the browser.[43]

---

[42] Blog: Steven Davelaar - http://bit.ly/jspincludevsdeccomponent

[43] Oracle ADF Fusion Middleware Performance and Tuning Guide 11.1.1.7.0 – Section 8.2.2 Tuning ADF Faces - http://docs.oracle.com/cd/E28280_01/core.1111/e10108/adf.htm#BDCBGHDI

- [ADFcg1-03051] – **Commit to JavaScript cross browser testing** - Any JavaScript that is used needs to be tracked and undergo specific cross-browser testing, accessibility testing and security review.
- [ADFcg1-03052] - **JavaScript strings invoked from the ExtendedRenderKitService should be minimal** - This ensures that the packets sizes for roundtrips are kept to a minimum and also ensures that the code that is invoked in this way is subject to the same scrutiny as other JavaScript used by the system.
- [ADFcg1-03053] - **Component ID Lookup using AdfPage.PAGE.findComponentByAbsoluteId()** - Component lookup in this way, with the complete and absolute ID of the component in question is the most reliable.
- [ADFcg1-03054] - **Make no assumptions about the DOM layout of a component** - The DOM generated by a particular component may change based on both the browser and the release. Always code defensively if you must deal with the DOM directly (unusual).

Messages

The following general guidelines apply to messages displayed to the user at runtime such as validation errors or informational messages:

- [ADFcg1-03055] - **Don't hardcode human readable text** - Do not hardcode human readable text in your messages. Rather store any hardcoded text in a resource bundle and refer to the key-value pair instead.

Pages/Page Fragments

The following general guidelines apply to both JSPX/JSF pages and page fragments:

- [ADFcg1-03056] - **Avoid inline HTML use** - Embedded HTML in JSF pages is often responsible for breaking the behavior of the ADF Faces Layout components. Additionally HTML embedded in a JSF page assumes that the page is indeed being rendered in a conventional browser and may link the application to a particular subset of browsers.
- [ADFcg1-03057] - **Don't hardcode human readable text** - Do not hardcode human readable text in your pages/fragments. Rather store any hardcoded text in a resource bundle and refer to the key-value pair instead.
- [ADFcg1-03058] - **Only one root component per page/fragment** - ADF Faces requires one root component per page. If you end up with more than one root element, for example a splitter and a popup, they should be surrounded with an af:group element.

Bookmarkable Pages

The following guidelines apply to pages that have been configured to be bookmarked:

- [ADFcg1-03059] - **Validation checks in bookmarkable activities** - Parameters passed on the URL to a bookmarkable view activity constitute an additional attack surface for the application. Always consider the security implications of generating bookmarkable views for usage such as deep links provided with emails.

Pages

The following specific guidelines apply to pages:

- [ADFcg1-03060] – **Use JSPX rather than JSP for 11gR1 applications** - If using a JDeveloper 11gR1 release (which doesn't use Facelets), use JSPX XML compliant files rather than JSP.

Page Fragments

There are currently no guidelines for page fragments.

Testing

The following guidelines apply to testing page and page fragments:

- [ADFcg1-03061] - **Manual tests for UIs should include stretch / shrink behavior** - Don't assume that your users will always be running at a particular resolution, or that they will always have the browser maximized
- [ADFcg1-03062] - **Manual tests for UIs should include changing font size** - As part of your manual testing you should routinely increase and decrease the font size in the browser (usually using the Ctrl + / Ctrl - key combinations) to ensure that the layout adapts in a usable way.

Page Templates

The following guidelines apply pate templates:

- [ADFcg3-03063] - **Define a facetRef for popups in your page template** - Defining a facet for popups in your page template allows developers to groups all popups in once place in a page. This also prevents popups from being stamped out multiple times when accidentally put in a table.
- [ADFcg1-03064] - **Carefully use page template nesting** - Be cautious when nesting page templates. Not all versions of ADF have been able to support template nesting, and it can make the re-construction of all of the constituents of the page more complex and harder to understand.

Regions

The following guidelines apply to regions:

- [ADFcg1-03065] – **Limit the number of page regions** - Limit the number of ADF regions you render in a page to 10.[44]
- [ADFcg1-03066] - **Inactivate task flow regions that aren't displayed** - For task flows in regions that aren't currently displayed, set the associated task flow region activation to deferred (11.1.1.4.0+) or conditional with a rule in the refreshCondition property.
- [ADFcg1-03067] - **queueActionEventInRegion should only be used for navigation** - This API is designed to trigger a navigation action in a region. It is not intended as a general executor of bound methods within the region and should not be used as such.

---

[44] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 23.11.3 What You May Need to Know About Configuring a Page to Render an Unknown Number of Regions - http://docs.oracle.com/middleware/1212/adf/ADFFD/taskflows_regions.htm#sthref574

Security

The following guidelines apply to security:

- [ADFcg1-03068] - **Avoid visible=false where possible for UI components** - components with visible set to false are still delivered to the client, just not shown. Any hidden data can be exposed, and values for the invisible component can be sent to the server creating a security risk.
- [ADFcg1-03069] - **Don't use the test-all role** - The test-all role is only intended for use during development and must be stripped out before security testing begins
- [ADFcg1-03070] - **Don't use user based identity checks** - Applications should not expect hard-coded user identity e.g. "admin" as a user.
- [ADFcg1-03071] - **Used ADF Security for FMW projects** - For applications that are to plug into a wider Oracle FMW solution we recommend that the ADF Security mechanism is used as the backbone security infrastructure, assuming that the target platforms will support this. Using ADF Security ensures that security permissions are externalized from the application and the application has maximum compatibility with the Fusion Middleware security infrastructure via OPSS.
- [ADFcg1-03072] - **Present only sanitized error messages to end users** - Error messages that reveal information such as database table names or stack traces to end users of the application are leaking information that may be of use to a hacker.

Skins

The following guidelines apply to skinning your application:

- [ADFcg1-03073] - **Avoid inlineStyle and contentStyle CSS added to a page** - CSS added directly in a page is not compressed, it makes the page harder to manage and maintain, and increases the page download size. Use inlineStyle and contentStyle properties for dynamic color coding only and have the properties referencing a managed bean property to return the CSS string
- [ADFcg1-03074] - **Use AFStretchWidth / AFAuxiliaryStretchWidth** styles rather than absolute 100% width styling - These pre-existing styles take into account the overall geometry management of the various ADF layout containers in different browsers and will give more reliable results.
- [ADFcg1-03075] - **Do not use percentage height styles** - Heights expressed as a percentage are not reliable and behavior differs between browsers. Use absolute values for height if height must be specified at all
- [ADFcg1-03076] - **Follow the "Less is Best" pattern with respect to use of width and height styling** - Most layout problems resolve to over-use of styling, causing the developer to fight against the mechanics of the layout components. Only inject width and height as a last resort.
- [ADFcg1-03077] - **Use spacers in preference to setting absolute height / width when whitespace is required** - Where styling is being used to enforce a certain amount of whitespace between objects us a spacer as the mechanism to do this as it will ensure that the space is retained even when the original abject expands (e.g. through a font size change)
- [ADFcg1-03078] - **Reusable styles should be encoded into the skin definition** - Maximize your re-use and ensure ease of maintenance. This will also tend to reduce style proliferation.

Task Flows

The following general guidelines apply to task flows:

- [ADFcg1-03079] – **Avoid multiple task flow definitions in one task flow file** - for each task flow XML metadata file restrict the file to one task flow definition.[45]
- [ADFcg1-03080] - **Unbounded task flow should only be used for entry point**s - Any view exposed through the adfc-config unbounded Task Flow is accessible via the URL and open to user manipulation. As such the actual pages exposed in this way should be restricted.

Activities

The following guidelines apply to task flow activity types such as views, method calls, task flow return activities and similar:

- [ADFcg1-03081] – **Avoid parent actions for task flows not defined in the same workspace** - Only use parent action activities for task flows coupled within the same workspace. As parent actions aren't defined in the task flow specification, external consuming task flows will not know of the parent action.[46]
- [ADFcg1-03082] – **Define single task flow return commit/rollback instances** - If a task flow requires a task flow return commit or rollback, ensure there is only one of each in the task flow.
- [ADFcg2-03100] - **Avoid excessive consecutive method call activities in a task flow** - task flows allow more than one method call activity between other activity types. While one or two consecutive method calls activities is likely to have inconsequential overhead in terms of accessing the binding layer, if you have a situation where you have several method calls in a row, consider rewriting them as a single method call so access to the binding layer is optimized.

Documentation

The following guidelines apply to documenting task flows:

- [ADFcg1-03083] - **Use of diagram annotations or <description> attribute to describe purpose of the task flow** - As Task Flows represent re-usable components it is very important to document their purpose in a clear manner for future maintenance.
- [ADFcg1-03084] - **Provide clear documentation on external dependencies (session vars, security roles, task flow templates, parent actions, contextual events, Bindings)** - As Task Flows represent re-usable components it is very important to document their API and dependencies in a clear manner for future maintenance.

Exception Handlers

The following guidelines apply to task flow exception handlers:

---

[45] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 20.1 About ADF Task Flows - http://docs.oracle.com/middleware/1212/adf/ADFFD/taskflows.htm#ADFFD1632

[46] Task Flow Design Fundamentals - Section Navigation Side Effects - http://bit.ly/adftaskflowfund

- [ADFcg1-03085] – **Define task flow exception handlers** - Every task flow should include an exception handler mechanism.[47]

Managed Beans

The following guidelines apply to managed beans used by task flows:

- [ADFcg1-03086] - **Task flows should declare all of the beans used internally** - Unless there is a hard-coded encapsulation model for a particular task flow it should not assume that any upstream flow has pre-declared a particular managed bean definition as that introduces an external dependency and assumes a certain load order.

Navigation and Control Flow Rules

The following guidelines apply to navigation and control flow rules:

- [ADFcg1-03087] - **Correctly form URLs for redirection** - Simply using the JSF ExternalContext redirect() function with a path to a viewId will result in the creation of a new session. Always encode the path correctly using the controller context.
- [ADFcg1-03088] - **Perform programmatic navigation with queueActionEventInRegion,setViewId on the controllerContext or queued ActionEvent on a hidden component** - Do not use the JSF navigationHandler directly. This bypasses the JSF lifecycle and important parts of ADFs processing (e.g. Metadata commit)

Parameters

The following guidelines apply to task flow parameters:

- [ADFcg1-03089] – **Use required parameters** - Use the required attribute for any parameter to ensure that the design time and runtime environments can identify that a particular parameter is optional or required.[48]
- [ADFcg1-03090] - **Do not use implicit input parameter storage** - By default ADF will store input parameters for bounded task flows in the pageFlowScope reserved for the bounded task flow. Programmers should not use this mechanism but rather write the parameter to an explicitly defined pageFlowScope bean variable defined by the programmer. This assists in two fashions, firstly breakpoints can be put in the accessors of the pageFlowScope bean to watch any activity, and secondly a runtime check will be performed to ensure the datatypes of the passed parameter matches that of the defined destination.[49]
- [ADFcg1-03091] - **Do not retrieve from the implicit input parameter storage** - Further to the previous rule, developers should read the parameter values from the explicitly defined pageFlowScope

[47] Task Flow Design Fundamentals - Section Exception Handling - http://bit.ly/adftaskflowfund
[48] Task Flow Design Fundamentals - Section Parameters - http://bit.ly/adftaskflowfund
[49] Task Flow Design Fundamentals - Section Parameters - http://bit.ly/adftaskflowfund

bean variable defined by the programmer rather than that written to the implicit pageFlowScope reserved for the bounded task flow.[50]

- [ADFcg1-03092] - **Do not use implicit output parameter storage** - By default ADF will store output parameters from a bounded task flow in the pageFlowScope of the calling bounded task flow. Similar to the input parameters programmers should not use this mechanism but rather write the parameter to an explicitly defined pageFlowScope bean variable defined by the programmer (note that output parameters are only applicable for page based bounded task flow, not fragment based bounded task flow).[51]
- [ADFcg1-03093] - **Do not retrieve from the implicit output parameter storage** - Further to the last rule and similar to the input parameter rule, developers should read parameter values from the explicitly defined pageFlowScope bean variable defined by the programmer rather than that written to the implicit pageFlowScope reserved for the calling bounded task flow.[52]

Security

There are currently no guidelines for task flow security.

Templates

The following guidelines apply to task flow templates:

- [ADFcg1-03094] – **Define a common task flow template** - For all bounded task flows define a single inherited task flow template that can be used to provide common functionality and stubs for adding functionality in one place without having to reconfigure every existing task flow. Such functionality could include a the task flow template initializer and finalizers calling stub methods in a relating bean, where logging logic could be added in the future.[53]
- [ADFcg1-03095] – **Prefix task flow template activity names** - Prefix all task flow template activity names with an acronym representing the task flow template such that the activities when inherited by a bounded task flow don't have name collisions.[54]

Testing

The following guidelines apply to testing task flows:

- [ADFcg1-03096] - **Bounded task flows with parameters should have a test harnesses** - Being able to test a bounded task flow in isolation ensures that you have not inherited any unexpected dependencies, or at least thoroughly understand the dependencies that you do have. The test harness page can also be used as part of the documentation for the task flow.

Transactions

---

[50] Task Flow Design Fundamentals - Section Parameters - http://bit.ly/adftaskflowfund
[51] Task Flow Design Fundamentals - Section Parameters - http://bit.ly/adftaskflowfund
[52] Task Flow Design Fundamentals - Section Parameters - http://bit.ly/adftaskflowfund
[53] Task Flow Design Fundamentals - Section Task Flow Templates - http://bit.ly/adftaskflowfund
[54] Task Flow Design Fundamentals - Section Task Flow Templates - http://bit.ly/adftaskflowfund

The following guidelines apply to task flow transactions:

- [ADFcg1-03097] -**Use isolated data control scope only when required** - Having an isolated data control scope can be very useful for those specific scenarios where you need separate transactions. Be aware of the cost in terms of resources and connections when using this facility.
- [ADFcg1-03098] – **Use task flow transactions for non-ADFBC / JTA use cases** - In the case of ADF BC applications use isolated data control scope to manage separate transactions. Task Flow transaction are primarily aimed at scenarios with mixed transactional data sources (EJB)
- [ADFcg1-03099] – **Use task flow end transaction behavior only used for transactional flows** - When using ADF/BC, transaction management should be controlled explicitly by calls to the Commit and Rollback operations on the Bindings. Allowing Task Flows to issue commits and rollbacks can have unexpected results depending on the context from which the Task Flow in question is called[55]

[55] ADF Prematurely Terminated Task flows - http://bit.ly/ICYZUp

WebLogic Server

The following guidelines apply to objects in the configuration of WebLogic Server for ADF applications:

Data Sources

The following guidelines apply to data sources on WebLogic Server:

- [ADFcg1-04000] - **Use WLS "Oracle Thin" drivers only** - only use "Oracle Thin" data source drivers for ADF applications on WL Do not use the WLS "Oracle Thin XA" data source drivers, they are not compatible with ADF applications. [56]
- [ADFcg2-04002] – **Don't use weblogic.jdbc.extensions.getVendorConnection()** - Use of this method is strongly discouraged as the connection cannot be returned to the pool.

High Availability

The following guidelines apply to highly available ADF applications installed on WebLogic Server:

- [ADFcg1-04001] - **For clustered environments set High Available for ADF Scopes** - For ADF applications that will be deployed to clustered environments ensure the adf-config.xml High Available for ADF Scopes option is set.57

[56] Oracle ADF Fusion Developer's Guide 12.1.2.0.0 - Section 48.3.7 What You May Need to Know About JDBC Data Source for Oracle WebLogicServer - http://docs.oracle.com/middleware/1212/adf/ADFFD/deployment_topics.htm#sthref928

[57] Oracle Support Note 1468116.1