ADF Coding Standards

Application Name – shortname like srdemo, speaker (lowercase)

Packages – prefixed by client or company name

Framework Extensions – should be in separate pkg. eg – FrameworkExtensions, frmwkext

BC – should be in Model or DataModel

Top Level BC directories –

Entities - EO

Design - Diagrams

Service - AM

Organize your directories to suit project lifecycle and source control system (typically Subversion these days) and library management tools like Maven. E.g. your build directories have to be outside your trunk folder. E.g.src\main\java, src\test\java folder for unit testing. Include a ‘database’ folder for your SQL schema creation scripts etc.

TaskFlow Naming Standards

<taskflowname> - eg. createemployees

<taskflowname>.view

<taskflowname>.view.backing

<taskflowname>.view.framework - used for overriden component classes.   
<taskflowname>.view.pageDefs

<taskflowname>.view.servlet - any servlet, filter or listener code that extends the JEE servlet that is used in generating output. eg. a servlet that dynamically renders a BLOB into a downloadable file.

<taskflowname>.controller - code specific to the controller layer within the ViewController package will include the sub-package name ‘controller’

<taskflowname>.controller.lifecycle – code that overrides the ADF controller such as the ErrorHandlerImpl class

<taskflowname>.controller.managed – ViewController managed beans should go into a sub-package name "managed"

<taskflowname>.controller.servlet - any servlet, filter or listener code that extends the JEE servlet that is used in handling input. eg. a servlet that manages security based on a non-standard login mechanism.

ViewController Naming Standards

Put View Controller code in a ‘ViewController’ project

For each ADFm page def file, its name should reflect the web page it supports with the suffix PageDef.Eg. ViewEmployeesPageDef

The name of each managed bean should reflect their function, eg. Versioner or CartProcessor, and not include information about its scope etc (since this can change).

Backing Beans Naming Standards

All backing beans should be in a ‘backing’ package

Backing beans should have the same name as the web page that they are used in, eg. ViewEmployees.jspx has a backing bean createemployees.view.backing.ViewEmployees

Backing beans should implement java.io.Serializable if the application is to be deployed on cluster

BC Projects

If you anticipate having more than around 50 entities you should we have separate model projects for different data subsystems or application functionality areas

When it’s needed to reuse ADF BC components between different model projects, use the import functionality for ADF BC with JAR files.

BC Entity Objects

Entities - put associations in an ‘associations’ sub-package to keep them out of the way of the other entities (which can get quite a large list anyway)

For clarity entity names should always same as the underlying table. Aino suggests "Reference entities (and their attributes) are prefixed with ‘Lkp’ for lookup, i.e. ‘Lkp<entityname>’. I would suggest for a moderately sized project these should be in a separate ‘lookups’ sub-package.  
Transient attributes, calculated on the EO, should be prefixed with 'Trnsnt"

BC View Objects

Put View Links in a "links" sub-package

VO Name – ReadingsByMeterVo (not VO)

LOV views – we recommend a separate package such as app.model.view.lov.

ViewLink names - if you have aliases for tables and FKs based on those the defaults work pretty well I reckon (I always have 3 char aliases and so an FK might be ABC\_DEF\_FK in which case the view link becomes "AbcDefFkLink").

ViewObject instance names in AM - interesting one - do you rename or do you leave the JDev default <View><number>? Currently I rename but I'm not sure it's worth the effort

Bundle up View Objects and View Links into functional subsystems and package with task flows if you are using them.

BC Application Modules

Application Module naming - seems to be nicer to call it <app>Service these days. How about multiple application modules? How about nested vs root names?

If your project is fairly big then you should consider bundling up entities, and possibly views, into separate application modules which can then be nested in your final applications.   
<Aino suggests: "We group all general read-only view objects that are used for LOV's and dropdowns into one application module and nest that into your other AM's." - worth considering as a standard practice?>

An Example Project Structure

* Common (app.common)
* UnitTest (app.unittest)
* Framework (app.framework)
* Model (app.model)
* app.model.services
* app.model.entities
* app.model.entities.associations
* TaskFlows (app.tasks)
* app.tasks.task1
* app.tasks.task1.views
* app.tasks.task1.views.viewlinks
* app.tasks.task1.backing
* app.tasks.task2
* app.tasks.task2.views
* app.tasks.task2.views.viewlinks
* app.tasks.task1.backing
* app.tasks.task3
* app.tasks.task3.views
* app.tasks.task3.views.viewlinks
* app.tasks.task1.backing
* ViewController (non-task flow specific UI)
* app.ui.backing
* app.ui.utility
* app.ui.pageDefs

Workspace Name - For example a workspace named VehicleLicensingSystem should have a correlating

VehicleLicensingSystem.jws file located with the directory VehicleLicensingSystem.