JAVA CENTER OF EXCELLENCE

JAVA CENTER OF EXCELLENCE

Describes how to setup, configure, and use StrutsCore, which is part of the Ford Java Frameworks (FJF).

For more information or feedback, contact the Java Center of Excellence

Revision History

This page documents general changes/modifications made to the document.

Version	Update Date	Change Description	Author
1.0	08/28/2008	Begin initial draft version using the "JcriBase User Guide" as the starting point.	mToll2
1.0	09/03/2008 09/05/2008 09/19/2008	JCOE review participants include Julie Burnard, Dilip Ladhani, Prathyusha Yenamandala, Rick Minto, Scott Shepard..	
1.0	10/06/2008	Update initial draft version per JCOE reviews.	mToll2
1.0	11/14/08	Update with code examples from JAB.	mToll2
1.0	12/09/2008	Update following JAB code review.	mToll2
1.0	01/23/2009	Added ScBaseLogoutAction.	mToll2
1.0	01/28/2009	Matured appendixes with most current JAB code examples for Milestone 9 release.	mToll2
1.0	02/03/2009	Added support for 'no' button which behaves the same as a 'back' button.	mToll2
1.0	03/04/2009	Clarified intent by using the term "Token Synchronization".	mToll2
1.0	03/27/2009	Updated examples from JAB with v. 1.0 code samples.	mToll2
1.0.1	05/04/2009	Added "Why StrutsCore" section.	mToll2
1.2	2/2/2011	Added a lot more information about Token Synchronization.	Oshvarts
1.3.1	06/22/2012	Added policy to role interaction related to DLS for authorization tag	PSOMASUN
1.4.2	11/06/2014	According to CVE-2014-0114, Struts is vulnerable to injection of tainted parameters. Mitigate the issue by filtering out bad parameters identified by the CVE. This upgrade would be required for all applications using Struts at Ford.	Syadav3

Table of Contents

1 Introduction

This document is an implicit extension of the JCOE "Software Design Guide".

StrutsCore enables a standard Ford web-application presentation layer development protocol including associated automated method (or unit) tests. The intent is to deliver business functionality with *as few maintenance points as possible*. It is asserted that ***consistently applied implementation patterns promote agility and velocity*** throughout an application Systems Development Life Cycle and that consistency across an enterprise magnifies the positive impact.

StrutsCore is a logical extension of the Struts Framework v. 1.x.

1.1 Why StrutsCore?

One may wonder why StrutsCore exists. Or, *what benefits are provided above and beyond the Struts Framework?* This introductory section provides a listing with brief descriptions. Further details are found through this document.

Frameworks designated as 'core' logically wrap (or abstract) one or more existing technologies. A 'core' framework exhibits four generalized behaviors identified as **ELLP**:

- **E** – Exceptions are contextualized for Ford Motor Company
  - Converts any Struts exceptions to a Ford self-logging runtime exception (that only logs once).
  - Explicitly identifies the layer of the stack challenging the exception.
- **L** – Logging provides insight into internal framework processing
  - Several of the 18 Struts Lifecycle phases are logged with request/response feedback.
  - Value-added logging is declaratively activated and is independent of log level.
- **L** – Limited API is published
  - A reduced set of choices is exposed by the StrutsCore API. This lessens the need to evaluate and select Struts capabilities best contextualized for the Ford environment.
  - A limited API does not encumber an application from bypassing 'core' framework classes and invoking native (Struts framework) functionality directly.
- **P** - Pattern enforcement of coding conventions
  - Several class naming conventions are enforced by a coding convention exception thrown upon instantiation.
    - Action classes
    - Form Bean classes
  - Several method naming conventions are enforced by a coding convention exception.
    - Validate methods in Form Beans
    - Navigate methods in Action Classes

In addition, StrutsCore provides the following unique functions not included in Struts:

- **Authorization**
  - Various plugins for 'request authorization' are conveniently accommodated.
  - 'View authorization' is supported with a set of JSP tags.
- **Session Expiration**
  - A default implementation is provided for session expiration.
- **Token Synchronization**
  - A default implementation is provided for token synchronization.
- **Lookup Dispatch Form Processing**
  - A default implementation for resolving validation methods is supported.
- **Default Button Support**
  - Currently, the following three buttons have default implementations:
    - 'Back'
    - 'Cancel'
    - 'No'
- **Chained Action Support for Navigation**
  - The following action behaviors are supported:
    - Begin Action
    - Pre Action
    - Ajax Action
    - Post Action
    - Logout Action
- **Working Example**
  - The JAB reference application provides a working example of StrutsCore.

1.2 Assumptions

Your reading of this text presumes your learning style is best served by a document. If you are more of a 'hands-on' learner you may want to review the JCOE Adventure Builder (JAB) application (source code) instead (of reading further). Code samples in this document are from the JAB application.

1.3 Dependencies

StrutsCore is dependent upon:

- itCore
- Struts
- JUnit
- Cactus

- StrutsTestCase (future)

1.4 Conceptual Overview

1.4.1 Ford Layered Architecture

StrutsCore explicitly compliments the Ford JCOE "Software Design Guide" layered approach to application architecture.

1.4.2 Consistent API Style

StrutsCore promotes a base class extension style of development for both functional and automated method test code. Application teams are encouraged to establish their own 'layer' of base classes – even if they are initially empty. Initial implementation of a base class layer promotes reuse during development; the existence of base classes makes it convenient for duplicate logic to be 'promoted' (to the base class).

1.4.3 Exceptions

StrutsCore implements a ***stack boundary*** approach; that is, *all* exceptions thrown to clients are extensions of a StrutsCore runtime exception. The approach includes exceptions thrown by dependent technologies (including the JDK); they are caught and wrapped in a StrutsCore exception. Simply stated, no exceptions are permitted to "bubble-up" the stack unchallenged.

StrutsCore throws only runtime exceptions. It adheres to the same approach as Spring, Hibernate and a number of other well-known open source frameworks – and the approach used by contemporary specifications such as JSF and JPA. Checked exceptions emanating from dependent frameworks (or the JDK) are wrapped in an applicable StrutsCore runtime exception.

Nuance Alert: Since Ford stateful self-logging exceptions are only logged once, the chances of a cluttered log file are minimized when catching and (re)throwing exceptions at various layers of the stack.

A final point: StrutsCore leverages a `ScCodingConventionException` when promoting enterprise coding consistency.

1.4.4 Authorization

StrutsCore provides considerable augmentation of Struts authorization support. The typical contexts are:

- Request Authorization - Declarative Authorization of an Action
- View Authorization - JSP Tags
- Direct Programmatic Invocation – Utility Functions

A "resource" and/or "privilege" ***authorization string*** is supported with the following (example) syntax:

```
resourceA:priviledge7,resourceB,resourceC:priviledge7,resourceC:priviledge8
```

The ***authorization string*** syntax is supported by the following programmatic artifacts:

- Request Authorization (See Appendix – Request Authorization)
  - `ScTilesRequestProcessor.processAuthorization()`
- View Authorization (See Appendix – View Authorization)

[*TODO: CHANGE IF PROMOTED TO PRESENTATION FRAMEWORK]*
  - `IsAuthorizedForAllTag`
  - `IsAuthorizedForAnyTag`
  - `NotAuthorizedForAllTag`
  - `NotAuthorizedForAnyTag`
- Direct Programmatic Invocation

  [*TODO: CHANGE IF PROMOTED TO ITCORE]*
  - `ScUtilAuthorization.isAuthorizedForAll()`
  - `ScUtilAuthorization.isAuthorizedForAny()`
  - `ScUtilAuthorization.notAuthorizedForAll()`
  - `ScUtilAuthorization.notAuthorizedForAny()`

Note: The term "All" above implies a logical "And" operation; the term "Any" implies a logical "Or" operation.

1.4.5 Validation

Because of IT industry architectural variation in validation philosophies, the JCOE is compelled to make strong recommendations to further consistency at Ford Motor Company. The recommendations are embedded within the StrutsCore validation implementation. (See Appendix – Validation)

The "Software Design Guide" defines the terms declarative validation and programmatic validation; the statements here are an extension of the philosophy contextualized for the Struts technology. The JCOE encourages Ford validation to be:

- Implemented as precisely as possible for all user input.
- Completed within the "processValidate" phase of the Struts lifecycle.
- Declared as much as possible ***without*** the use of proprietary scripting languages.

As a convenience, the `strutscore-config.xml` descriptor document supports declarative suppression of all validations during development.

1.4.6 Session Context

As compensation for the Struts constraint of a single formbean per request, StrutsCore provides an assist for the storing of values in session. The following classes may be reviewed to understand the options:

- JSP Tags (See Appendix – Store Values In Session)

  [*TODO: CHANGE IF PROMOTED TO PRESENTATION FRAMEWORK]*
  - `GetValueTag`
  - `IsValueTag`
  - `NotValueTag`
- Direct Programmatic Invocation
  - `SessionUtil.setStringValue()`
  - `SessionUtil.getStringValue()`
  - `SessionUtil.isStringValue()`
  - `SessionUtil.notStringValue()`

1.4.7 Declarative Lifecycle Logging

Developers are familiar with "level-based" logging. Declarative logging (normally used during development) enables granular control of an explicit set of log statements.

Log statements may be declared as on ("true") or off ("false"). The logging provides insight into the request/response life cycle intended to enhance development velocity. (See Appendix – strutscore-config.xml)

2 Configuration

2.1 Required Files

StrutsCore is downloaded from the JCOE website.

2.2 Declarative Artifacts

Since StrutsCore is a *framework abstraction*[1] framework, most declarative programming is associated with the underlying technology.

However, StrutsCore has additional specific requirements:

- A `strutscore-config.xml` descriptor document is required for each environment.
- A servlet filter is required in `web.xml` declaring the `RequestContextFilter` (`itcore.jar`)
- A listener is required in `web.xml` declaring `ScConfigInitializer`
- The Struts resource bundle (typically `ApplicationResources.properties` in the root of the "src" folder) needs specific entries similar to:
  - `button.strutscore.httpGet.default=httpGet`
  - `button.strutscore.cancel.default=Cancel`
  - `button.strutscore.back.default=Back`
  - `button.strutscore.no.default=No`

3 Developing with StrutsCore

3.1 Controller – Action Classes

The Struts Lookup Dispatch Action machinery is augmented by extension. In turn, `ScBaseBeginAction`, `ScBasePreAction`, `ScBaseAjaxAction`, `ScBasePostAction` and `ScBaseLogoutAction` are extended by an application (preferable by an application level base class). By convention, methods used for resolving navigation begin with the verb 'navigate'.

[1] A "framework abstraction" wraps an existing technology and provides a Ford contextualized API.

A 'Begin Action' begins every unit of work (transactional sequence) and does not declare a form bean. Typical responsibilities include:

- Removal of stale form beans from session.

A 'Pre Action' is (optionally) used for pre-page processing and typical responsibilities include:

- Ensuring form bean attributes are properly staged for rendering a JSP. (A typical example is pre-population of drop down boxes for subsequent user selection.)

An 'AJAX Action' is (optionally) used for handling a HTTP-GET from a client for asynchronous processing. Typical responsibilities include:

- Staging of form bean attributes. (A typical example is pre-population of a dependent drop down box for user selection based on the user's prior selection.)
- Immediate validation of user input (prior to a page submission).

A 'Post Action' is used for post page processing and typical responsibilities include:

- Invocation of form bean post processing (as required).
- Resolving navigation to the next chained 'pre action' or tiles page.

A 'Logout Action' is used for post page processing and typical responsibilities include:

- Releasing of resources and redirecting.

Nuance Alert: StrutsCore has built-in support for 'cancel' button, 'back' button and 'httpGet' processing. Because request processing is recursive, chaining of actions requires special handling. Fortunately, this complexity is accomplished declaratively in the `struts-config.xml` descriptor document. (See Appendix – Navigation)

3.2 Form Bean

The Struts 'Form Bean' machinery is augmented by a `ScBaseLDForm` class providing lookup dispatch support similar to that found in a 'Lookup Dispatch Action' class. This strategy provides convenient (and consistent) support for the "Single-Form Wizard" pattern. Similar to *navigate* methods being invoked (on an action class), *validate* methods are invoked for a specific button on a page.

By convention, methods invoked from a 'PreAction' begin with the verb 'populate' and methods invoked from a 'PostAction' begin with the verb 'process'.

Nuance Alert: Consistent with a 'Lookup Dispatch Action' class: method mapping is required (*when validation is turned on*). This makes sense for an Action class – it is not always intuitive in a 'Form Bean' when validations are *not needed* for some buttons. Consider the following perspective: For each button on the page a mapping must exist. The mapped method may:

- ❖ invoke declarative validation(s)
- ❖ invoke programmatic validation(s)
- ❖ invoke both declarative and programmatic validations
- ❖ invoke no validations

3.3 Front Controller – Request Processor

The Struts 'Front Controller' machinery is augmented by the `ScTilesRequestProcessor` and supporting utility classes.

The reader is best served by (and encouraged to perform) a review of source code for these classes. In order to encourage scrutiny – no additional documentation is provided here. ☺ If so motivated – watch a request flow through this machinery in the debugger! Minimal, if any, configuration requirements are stipulated in the Javadoc.

Nuance Alert: StrutsCore provides a 'Front Controller' implementation of the traditional *Synchronizer Token* pattern. The term "Token Synchronization" describes the ability of this machinery to gracefully recover from an invalid (browser navigation) HTTP POST request or double-click and return the user to a prior (valid) client state without errant server-side processing.

3.4 Token Synchronization

3.4.1 Why the need for the 'Synchronizer Token Pattern'?

The main problem that this pattern is trying to solve is the double submission of forms. For example, these events can lead to a double submission:

- Clicking more than once on a submit control
- Using refresh button
- Using the browser back button to resubmit form
- Using browser history feature and re-submit form
- Malicious submissions to the server

A related problem has appeared recently is also helped by the Token Synchronization pattern. The problem manifests itself when a user opens two browser windows or browser tabs and mistakenly believes the work is they do in the different tabs is completely independent, whereas in fact the tabs share a session and will work on the same objects and beans because cookies that control the session state are shared between the two tabs.

3.4.2 Why a generalized solution?

StrutsCore implements a generalized solution on behalf of the company. This avoids individual applications evaluating every request/response interaction for exposure; and having to code the several steps required for eliminating risk.

3.4.3 Are there any constraints imposed by a generalized solution?

Yes. Only the HTTP POST browser method is supported. Therefore, an application needs to ensure that all interaction between the client and server impacting server-side transactions is handled by an HTTP POST. By working closely with CDU a development team can ensure that non-trivial interactions between client and server are designed as HTTP POST operations.

3.4.4 Is there any value-add enabled by a generalized solution?

Yes. The StrutsCore implementation demonstrates "Transparent Recovery". (Read on for implementation details.)

3.4.5 How is token synchronization handled?

The ScTilesRequestProcessor provides a 'process()' method that invokes a 'ScUtilTokenSynchronization.preProcessTokenSynchronization()' method:

This method is invoked prior to the Struts `RequestProcessor.process()` method. Both this method and `postProcessTokenSynchronization()` should be invoked within the same synchronized block on session id. This method implements the Synchronizer Token Pattern to accomplish token synchronization for all HTTP POST operations - with the exclusion of multi-part forms. Reloads, or out-of-sequence invocations, of a POST will result in the last valid page being displayed without any processing occurring. If an out-of-sequence POST submission can not be gracefully handled, or if an application configures the appropriate option in its configuration file, a `ScTokenSynchronizationException` is thrown. There is additional memory consumed to implement this transparent recovery strategy.

The ScTilesRequestProcessor 'process()' method also invokes a 'ScUtilTokenSynchronization.postProcessTokenSynchronization()' method. This method is invoked after all recursive request processing is completed. Sufficient state is saved in session so transparent recovery may occur in the event a subsequent request is out-of-sequence.

3.4.6 What are the scenarios token synchronization is trying to prevent?

Token Sync is trying to prevent out of order execution, defined as navigation using an order other than that provided by application. Examples:

- A customer, used to double-clicking Windows icons, tries to buy a product and double-clicks the buy icon. Application may see it as two distinct orders and double-processes the transaction.
- A customer refreshes the “thank-you-for-your-order” screen, inadvertantly resubmitting the order and buying items in their shopping cart again.
- A user may open another tab to create a second workstream within the application, not realizing that since tabs share a single session, he’s actually working on the same object in both tabs.
- A customer bookmarks an intermediate step in a workflow and later returns to that bookmark, leaving an object in half-populated state and bypassing a lot of application’s validation.
- A user uses the back button to return to a previous page and changes some values, invalidating the entire workflow.

In all of these cases, it is desirable to either gracefully handle the situation or inform the user that their navigation is not supported on the site, requesting they return to the start of the transaction to make sure it follows all the business rules.

3.4.7 How is token synchronization implemented?

Token sync implementation is controlled through options in the StrutsConfig file, and through optional annotations (@TokenSynchronizable and @NonTokenSynchronizable) on action classes.

Initial configuration is performed in the struts-config.xml file. The following is an example setup section:

```
<property-group name="TokenSynchronization">
    <property name="defaultBehavior">synchronized</property>
    <property name="behaviorOnError">restoreState</property>
</property-group>
```

The `token-synchronization` group is the parent group of the token synchronization settings group.

The *`defaultBehavior`* element establishes default behavior for Token Synchronization. Setting the value to *synchronized* synchronizes all actions by default; setting the value to *unsynchronized* does the opposite. Framework default is *synchronized*, to remain compatible with previous releases of Strutscore and to provide a viable default implementation, but for many projects, the better choice is to set the value to *unsynchronized* and then use @TokenSynchronizable annotation on PostAction classes to enable synchronization of only specific actions. With the default value of *synchronized*, marking any Action class with the @NonTokenSynchronizable annotation turns off annotation processing for that action.

The *`behaviorOnError`* group establishes what the framework should do in case of a Token Synchronization failure. Option's value `restoreState` transparently displays the last successfully generated page; this option is best for preventing double-submits of form buttons, and will guarantee that, for example, a customer is not charged twice if they click the "buy" button two times. The value `exception` will throw an ScTokenSynchronizationException and will transition control to the application's error page, where the situation can be explained to the user. This behavior is best when the application is trying to prevent out of order execution, such as a user working on two different tabs at the same time or using browser navigation (bookmarks, back button, etc) rather than application's own navigation. The value `clientWarning` will do the same as `restoreState`, except it will additionally display a JavaScript warning when the state is restored, to let the user know they have followed inappropriate navigation.

After configuration is completed, the application can further customize behavior using annotations. These would be most useful on PostAction classes, and take the following form:

```
26  * This class performs a Search Booking
27  */
28 @TokenSynchronizable
29 public class SearchBookingPostAction extends JabBasePostAction {
30
31     /** The Class Name used for Logging */
32     private static final String CLASS_NAME =
33             SearchBookingPostAction.class.getName();
34
35     /** The Logger instance used for Logging */
36     private static final ILogger log =
37             LogFactory.getInstance().getLogger(CLASS_NAME);
38
39     /**
40      * Constant for the navigateSearch method name to be used by the
```

Marking an action with @TokenSynchronizable annotation allows synchronizing an action that would otherwise not be synchronized, i.e. if the configuration file's *behaviorOnError* setting is set to *unsynchronized*. Inversely, @NonTokenSynchronizable annotation disables synchronization on actions that would otherwise be synchronized, either if the configuration file's *behaviorOnError* setting is set to *synchronized* or if the configuration is not set at all (since *synchronized* is the framework default).

3.4.8 Caching

Important note: The application should be mindful of caching headers on web pages in order to make sure Token Sync processing is done properly. At least the last page (JSP) of transactional processing should have its cache turned off (e.g. issue non-caching headers) in order to make sure a fresh token is created prior to the form's submission.

3.4.9 Recommendation

JCOE recommends that the default be set to unsynchronized, and actions that are transactional in nature (the "buy" screen and similar screens where submitting something twice would result in undesired double processing) have an @TokenSynchronizable annotation on them. Alternatively, applications that must ensure in-sequence processing (for example, if they have many wizard-like sequence screens) should turn on synchronization by default.

The following is JAB's default configuration of Token Sync behavior:

```
<property-group name="TokenSynchronization">
      <!-- Token sync default behavior. Choices are "unsynchronized" or
      "synchronized"; for more information on what these do, see StrutsCore
      documentation.
      Should likely be unsynchronized by default in apps using newer
      versions of the framework. -->
    <property name="defaultBehavior">unsynchronized</property>
    <!--  What to do in case of a token sync error. Choices are
    "restoreState" to restore last known good state, or "exception" to
    throw an exception and let the application handle it, or "clientWarning"
    to display a JavaScript warning message. -->
    <property name="behaviorOnError">clientWarning</property>
</property-group>
```

JAB demonstrates the recommendation by turning synchronziation off by default and using @TokenSynchronizable annotations on transactional (e.g. data-modifying) PostActions.

3.5 StrutsCore with Security Vulnerability Mitigation

A recently discovered (2013) vulnerability in Struts 1.x requires that all applications using StrutsCore v1.4 (or earlier) must follow the instructions in this section to mitigate the risk of injection of malformed parameters.

3.5.1 Technical Background

JAB5 demonstrates how to implement a parameter filter to prevent the injection exploit.

The `ParamFilter.java` class of StrutsCore will create a ServletRequest wrapper that overwrites `getParameterNames()` method and checks the parameter names against a customizable regular expression. If a parameter name matches this regular expression, it will be removed from the list returned by `ServletRequest.getParameterNames()`.

WAS 8 applications are not required to specify `filter` and `filter mapping` in `web.xml` for ParamFilter as WAS8 (servlet 3.0) supports the `@WebFilter` annotation used in StrutsCore (for WAS 8). WAS 6.1 (servlet 2.4) does **not** support `@WebFilter` annotation and it is required to specify `filter` and `filter mapping` in `web.xml`.

3.5.2 Applications Using StrutsCore for WAS 8

Application teams just need to update to strutscore-fjf.jar (WAS 8 version) v1.4.2 or higher.

3.5.3 Applications Using StrutsCore for WAS 6.1

1. Application teams need to update to strutscore-fjf.jar (WAS61 version) v1.4.2 or higher.
2. In `web.xml`, add the following entries for `filter` and `filter mapping`:

```
<filter>
      <display-name>ParamFilter</display-name>
      <filter-name>ParamFilter</filter-name>
      <filter-class> com.ford.it.strutscore.filter.ParamFilter
      </filter-class>
      <init-param>
            <param-name>excludeParams</param-name>
            <param-value>(.*\.|^|.*|\[('|"))(c|C)lass(\.|('|")]|\[).*
            </param-value>
      </init-param>
</filter>
```

```
<filter-mapping>
      <filter-name>ParamFilter</filter-name>
      <url-pattern>*.do</url-pattern>
</filter-mapping>
<filter-mapping>
      <filter-name>ParamFilter</filter-name>
      <url-pattern>*.jsp</url-pattern>
</filter-mapping>
```

4 Appendices

4.1 Appendix – Struts v. 1.3 Request / Response Lifecycle

Struts One	ScTilesRequestProcessor
0. process	0. Overridden/Finalized and invokes: processUnitTesting() **- or -** processSessionExpiration() processRequestInitialization() ScUtilTokenSynchronization.preProcessTokenSynchronization(); includes lifecycle logging.
1. processMultipart	
2. processPath	
3. processException	Overridden – Ensures all exceptions are wrapped by a Ford self-logging exception.
4. processLocale	
5. processContent	
6. processNoCache	
7. processPreprocess	
8. processCachedMessages	
9. processMapping	Overridden and invokes processHttpGet() which invokes the processBackAndCancel() method; and lifecycle logging.
10. processRoles	Overridden/Finalized and invokes: processAuthorization() for request authorization status; and lifecycle logging.
11. processActionForm	Overridden for request lifecycle logging.
12. processPopulate	Overridden for request lifecycle logging
13. processValidate	
14. processForward	
15. processInclude	
16. processActionCreate	
17. processActionPerform	Overridden to support token synchronization and lifecycle logging.
18. processForwardConfig	Overridden – Changes method modifier from protected to public.
	ScUtilTokenSynchronization.postProcessTokenSynchronization()

4.2 Appendix – StrutsCore Lifecycle FAQ

How is a HTTP GET intercepted?

The ScTilesRequestProcessor provides a 'processHttpGet()' method:

This method establishes a default key/value pair on the request in the cases where an HTTP GET was submitted. Typical scenarios include: (1) The submission was triggered via JavaScript with the intention of returning the user to the same page. (2) The submission was triggered from a hyperlink.

How are default 'back', 'cancel' and 'no' buttons intercepted?

The ScTilesRequestProcessor provides a 'processBackAndCancel()' method:

This method handles a default cancel or back button submission. The Struts cancel key (or token) is set to avoid subsequent processing - such as validation and form bean population.

4.3 Appendix – Cohesion: Form Bean vs. Action Class FAQ

How are responsibilities divided between the Form Bean and Action Classes?

When reviewing the JAB code base, some developers are surprised at the amount of logic contained within Form Beans and how little logic is contained within Action Classes.

At Ford we promote a 'fat form' / 'slim action' approach. The division of responsibilities and rationale are explained in further detail below.

Why a 'fat form' approach at Ford?

There are several thought-paths moving us toward a 'fat form' approach.

1) **Encapsulation**: The 'Software Design Guide' promotes a 'Common Layer' inclusive of Business Objects that cross architectural layers. At Ford we recommend that BO instances are contained within the Form Bean for convenient interaction with a JSP via 'dot' notation. The principle of object encapsulation dictates that methods interacting with the BOs (and read only interfaces) are contained within the Form Bean.

2) **Training**: The 'fat form' / 'slim action' approach is easily communicated to new developers as we strive for consistency in our corporate code base. Simply stated, use Action Classes primarily to control navigation and use the Form Bean for model (as in MVC 2) processing.

3) **JSF Architecture**: The JSF (a newer competing presentation framework) approach provides insight on industry direction. The need for partitioning logic between an action class and a bean is absent since JSF implements declared Managed Beans and no action class equivalent.

4) **Unit Testing**: It is considerably more efficient to unit test non-navigation logic contained within a form then then invoking logic spread across multiple action classes.

What about populating components with values before a page is rendered?

Again to further encapsulation, it is recommended that a Form Bean method (beginning with the verb *populate*) is used to modify the state of Form Bean attributes (fields) prior to rendering a page. Such methods are typically invoked from a 'PreAction' class.

What about transactional logic requiring access to a BO after submission?

Since the Form Bean contains the BOs, and again to further encapsulation, it is recommended that the verb *process* is used to modify the state of Form Bean attributes (fields) after submission. Such methods are typically invoked from a 'PostAction' class.

So which class serves as a client to the Business Layer? The Form Bean or the Action Class?

There is a architectural nuance here worth considering. In this approach the BO (or RO interface) actually serves as the 'Model' for the Model-View-Controller (MVC II) architecture. Since it is

contained within the Form Bean, it is convenient for interaction between the Presentation Layer and the Business Layer to be handled by the Form Bean.

What is left for the Action Class?

The JAB-style of development relegates the Action Classes to serve primarily a navigational (controller) responsibility.

Wait just a minute here … what about session memory consumption?

Some developers are concerned that placing large numbers of instances in a session-scoped form will have a negative impact on memory consumption. It is worthwhile to consider that ROs are normally references to shared cache. JAB-style development is predicated on the use of Persistence technology that includes caching.

4.4 Appendix – Logout

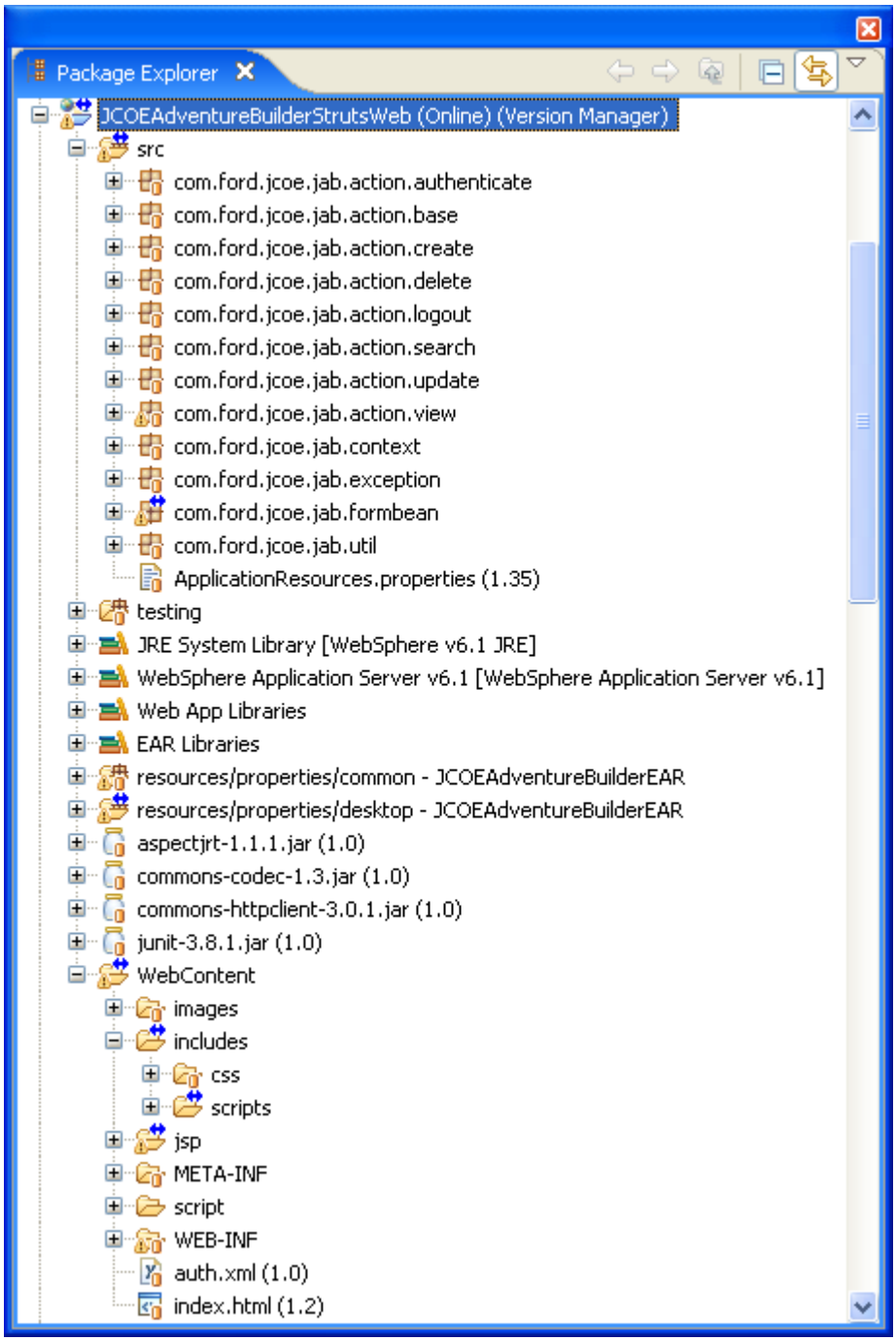
Descriptor document (struts-config.xml) example:

```
<action path="/logout"
      type="com.ford.jcoe.jab.action.logout.JabLogoutAction"
      parameter="method">
</action>
```

Programmatic invocation (JabLogoutAction) example:

```
/**
 * This call back method defines the absolute or relative URI for which
 * control should be forwarded AFTER log-out occurs. There are three
 * options. 1.) A returned <code>null</code> resolves to the WSL logout
 * page. 2.) A relative URI may be supplied beginning with a leading '/' 3.)
 * An absolute URI is also supported.
 *
 * @return String - Indicates <code>null</code> or relative URI or
 *         absolute URI.
 */
@Override
protected String redirectUriCB() {

    String METHOD_NAME = "redirectUriCB";
    log.entering(CLASS_NAME, METHOD_NAME);

    log.exiting(CLASS_NAME, METHOD_NAME);
    return "/homePagePre.do";
   }
```

4.5 Appendix – Web Project Packaging



4.6 Appendix – Request Authorization

Descriptor document (fjf-security-plugins-config.xml) example:

```
<property-group name="AuthorizationProviderFactory">
    <property name="JAB">
        com.ford.jcoe.jab.util.JabAuthorizationProviderFactory
    </property>
</property-group>
```

Descriptor document (strutscore-config.xml) example:

```
      <property-group name="Authorization">
            <property name="AuthorizationProviderFactoryPropertyName">
                  JAB
            </property>
      </property-group>
```

Descriptor document (web-config.xml) example:

```
    <listener>
        <description>
            This listener loads fjf-security-plugins-config.xml
        </description>
        <display-name>AuthorizationInitializer</display-name>
        <listener-class>
            com.ford.it.security.plugins.AuthorizationInitializer
        </listener-class>
    </listener>
```

Descriptor document (struts-config.xml) example:

```
<!-- Home page related actions -->
<action path="/homePagePre"
            type="com.ford.jcoe.jab.action.view.HomePagePreAction"
            name="searchForm"
            scope="session"
            validate="false"
            parameter="method"
            roles="agent,customer">
      <forward name="httpGet" path="tile.home"></forward>
      <forward name="home" path="tile.home"></forward>
      <forward name="cancelChained" path="tile.home"></forward>
</action>
<action path="/homePagePost"
type="com.ford.jcoe.jab.action.view.HomePagePostAction"
            name="searchForm"
            scope="session"
            validate="false"
            parameter="method"
            roles="agent">
      <forward name="cancel" path="tile.authentication"></forward>
      <forward name="createBooking"
path="/createBookingStep1Begin.do"></forward>
</action>
```

4.7 Appendix – View Authorization

JSP (header.jsp) example:

With only Resource authorization and no Data Condition & Resource Data Interaction:

```
<!-- Authorized Agent-->
<sc:isAuthorizedForAny authorizationString="agent">
     <div id="user_id">
       <ul>
       <li>Welcome! (<%=request.getUserPrincipal()%>)</li>
       <li>Booking Agent: jabagent</li>
       <li><a href="<%=request.getContextPath() %>/logout.do">Logout</a></li>
       </ul>
     </div>
</sc:isAuthorizedForAny>

<!-- Authorized Customer-->
<sc:isAuthorizedForAny authorizationString="customer">
     <div id="user_id">
       <ul>
       <li>Welcome! (<%=request.getUserPrincipal()%>)</li>
       <li>Customer:
<%=RequestContext.getLocalInstance().getUserId("unknown")%></li>
       <li><a href="<%=request.getContextPath() %>/logout.do">Logout</a></li>
       </ul>
     </div>
</sc:isAuthorizedForAny>
```

With only Resource authorization, Data Condition and no Resource Data Interaction:

```
<!—Create Vehicle Order-->
<sc:isAuthorizedForAny authorizationString="CreateVehicleOrder:execute"
       dataConditions="ModelYear:2012:WRITE,VehicleLineCode:Focus:READ">
     <div id="user_id">
       <ul>
       <li>Welcome! (<%=request.getUserPrincipal()%>)</li>
       <li> Create Vehicle Order: jabagent</li>
       <li><a href="<%=request.getContextPath() %>/logout.do">Logout</a></li>
       </ul>
     </div>
</sc:isAuthorizedForAny>
```

With Resource authorization, Data Condition & Resource Data Interaction:

```
<!—Create Vehicle Order-->
<sc:isAuthorizedForAny authorizationString="CreateVehicleOrder:execute"
       dataConditions="ModelYear:2012:WRITE,VehicleLineCode:Focus:READ"
      resourceDataInteraction="true">
    <div id="user_id">
      <ul>
      <li>Welcome! (<%=request.getUserPrincipal()%>)</li>
      <li> Create Vehicle Order: jabagent</li>
      <li><a href="<%=request.getContextPath() %>/logout.do">Logout</a></li>
      </ul>
    </div>
</sc:isAuthorizedForAny>
```

Note: Refer **Appendix – aps-config-property.xml** for additional APS configuration related to Policy to Role interaction**.**

4.8 Appendix – Validation (Declarative and Programmatic)

Abridged descriptor document (`struts-config.xml`) example:

```
<!-- Create booking Step1 related actions -->
<action path="/createBookingStep1Begin"
      type="com.ford.jcoe.jab.action.create.CreateBookingStep1BeginAction"
      validate="false" parameter="method" roles="agent">
</action>

<action path="/createBookingStep1Pre"
      type="com.ford.jcoe.jab.action.create.CreateBookingStep1PreAction"
      name="bookingForm" scope="session" validate="false"
      parameter="method" roles="agent">
</action>

<action path="/createBookingStep1Post"
      type="com.ford.jcoe.jab.action.create.CreateBookingStep1PostAction"
      name="bookingForm" scope="session" validate="false"
      parameter="method" roles="agent">
</action>

<!-- Create booking Step2 related actions -->
<action path="/createBookingStep2Pre"
      type="com.ford.jcoe.jab.action.create.CreateBookingStep2PreAction"
      name="bookingForm" scope="session" validate="false"
      parameter="method" roles="agent">
</action>

<action path="/createBookingStep2Post"
      type="com.ford.jcoe.jab.action.create.CreateBookingStep2PostAction"
      name="bookingForm" scope="session" validate="true"
      input="createBookingStep2Pre.do" parameter="method" roles="agent">
      <set-property property="cancellable" value="true"/>
</action>
```

Nuance Alert: Notice the `<set-property property="cancellable" value="true"/>` for the last action declaration. This entry is required by Struts to avoid an InvalidCancelException. (http://wiki.apache.org/struts/StrutsUpgradeNotes128to129)

Form bean abridged example starting with adding validation methods:

```
public class BookingForm extends JabBaseForm {

@Override
protected void addValidationMethodsCB(
        ScValidateMethodAttributes validateMethodAttributes) {

    String METHOD_NAME = "addValidationMethodsCB";
    log.entering(CLASS_NAME, METHOD_NAME);

        validateMethodAttributes.add(JabConstant.STEP1_CONTINUE_ACTION_PATH,
                JabConstant.CONTINUE_RB_KEY,
                ScConstant.VALIDATE_DECLARATIVE_INVOKE_METHOD_NAME);
        validateMethodAttributes.add(JabConstant.STEP1_CONTINUE_ACTION_PATH,
                JabConstant.ADVENTURE_TYPES_RB_KEY,
                ScConstant.VALIDATE_DECLARATIVE_SUPPRESS_METHOD_NAME);
        validateMethodAttributes.add(JabConstant.STEP1_CONTINUE_ACTION_PATH,
                JabConstant.ADVENTURE_DETAILS_RB_KEY,
                ScConstant.VALIDATE_DECLARATIVE_SUPPRESS_METHOD_NAME);
        validateMethodAttributes.add(JabConstant.STEP2_CONTINUE_ACTION_PATH,
                JabConstant.CONTINUE_RB_KEY,
                JabConstant.START_END_DATE_VALIDATION_METHOD_NAME)

    log.exiting(CLASS_NAME, METHOD_NAME);
    return;
}
```

Form bean example of a method including programmatic validation:

```
/**
 * This method invokes declarative validations and then programmatic
 * validations for StartDate and EndDate.
 *
 * @param mapping
 * @param request
 * @return <code>ActionErrors</code> - A class encapsulating error messages.
 * @throws Exception
 */
public ActionErrors validateStartEndDates(final ActionMapping mapping,
        final HttpServletRequest request) throws Exception {

    final String METHOD_NAME = "validateStartEndDates";
    log.entering(CLASS_NAME, METHOD_NAME);

    // Fire the (declarative) validation method.
    ActionErrors actionErrors = validateDeclarativeInvoke(mapping, request);

    // Declare/Initialize
    Calendar startCal = processDate(this.getStartDate());
    Calendar endCal = processDate(this.getEndDate());
    Calendar calToday = getToday();

    // Start Date must be >= today and End date must be >= Start date.
    if (startCal.after(endCal)) {
        addMessage("errors.bookingForm.endDate.early", actionErrors);
    }
    if (startCal.before(calToday)) {
        addMessage("errors.bookingForm.startDate.early", actionErrors);
    }
    log.exiting(CLASS_NAME, METHOD_NAME);
    return actionErrors;
}
}
```

Abridged descriptor document (`validation.xml`) example:

```
<form name="/createBookingStep2Post">
      <field property="existingCustomer" depends="required">
      <arg position="10" name="required"
            key="label.bookingForm.existingCustomer" />
      </field>
      <field property="startDate" depends="required,date">
            <arg0 key="label.bookingForm.startDate" />
            <var>
                  <var-name>datePattern</var-name>
                  <var-value>MMM-dd-yyyy</var-value>
            </var>
      </field>
      <field property="endDate" depends="required,date">
            <arg0 key="label.bookingForm.endDate" />
            <var>
                  <var-name>datePattern</var-name>
                  <var-value>MMM-dd-yyyy</var-value>
            </var>
      </field>
</form>
```

4.9 Appendix – Store Values In Session

The value is set in session example:

```
SessionUtil.setStringValue("keyName", "the value", request);
```

JSP examples using tags:

```
The value is <sc:getValue name="keyName"></sc:getValue>.

<sc:isValue name="keyName" value="will not display">
      This text is displayed if a match is found for the 'value'.
</sc:isValue>
```

Text displayed from JSP:

The value is the value.

The value is retrieved from session example:

```
String value = SessionUtil.getStringValue("keyName", request);
```

4.10 Appendix – strutscore-config.xml

Descriptor document (strutscore-config.xml) example:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE property-group PUBLIC "PropertyGroup.dtd" "PropertyGroup.dtd">

<property-group name="ScConfig">

      <property-group name="ScBaseIncontainerTestCase">
            <property name="authenticationEnabled">true</property>
            <property name="authenticationType">basic</property>
            <property name="userID">wasadm01</property>
            <property name="password" encrypted="false">wasadm01</property>
      </property-group>

<!--  Validation Suppressed - Validation may be suppressed during development.
-->
    <property-group name="Validation">
        <property name="suppressValidation">false</property>
    </property-group>


<!--  Declarative Logging - Log statements  are on (true) or off (false).
      Normally activated (true) during development only.
-->
      <property-group name="DeclarativeLogging">
            <property name="requestReceived">true</property>
            <property name="requestLifecycle">true</property>
      </property-group>

<!--- The 'AuthorizationProviderFactoryPropertyName' is optionally used to
represent the authorization provider utilized.
It maps to a fjf-security-plugins-config.xml AuthorizationProviderFactory
property group.

For example:

<property-group name="Authorization">
      <property name="AuthorizationProviderFactoryPropertyName">
            default
      </property>
</property-group>
-->

</property-group>
```

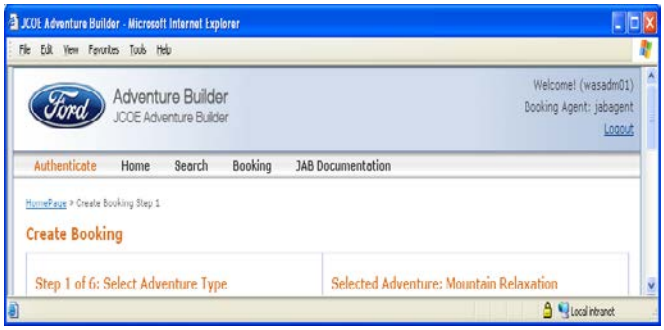
4.11 Appendix – ApplicationResources.properties

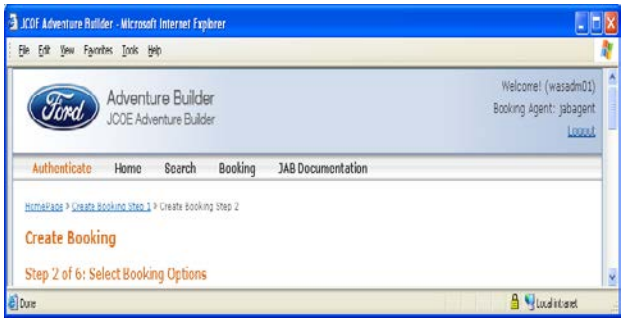
Property file document (ApplicationResources.properties) example:

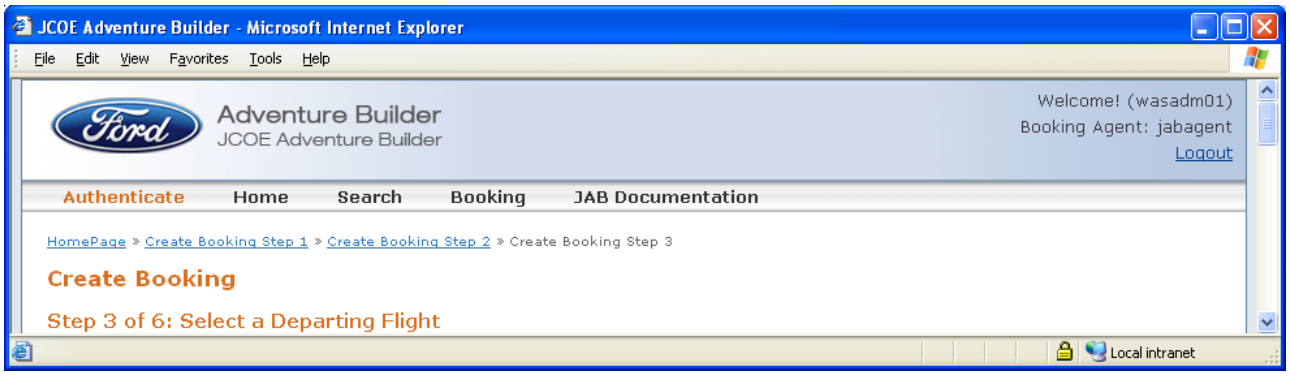
```
# Default buttons provided from StrutsCore. These are required.
button.strutscore.httpGet.default=httpGet
button.strutscore.cancel.default=Cancel
button.strutscore.back.default=Back
button.strutscore.no.default=No

# Custom buttons used in Java Adventure Builder application.
# Format: button.<buttonText>=<Button Text>
button.enterAsAgent=Enter As Agent
button.enterAsCustomer=Enter As Customer
button.continue=Continue
button.bookAdventure=Book Adventure
button.save=Save
button.updateFlight=Update
button.updateBooking=Update

# Ajax requests used in Java Adventure Builder application.
# Format: ajax.<In a JSP: method='methodName'>=<methodName>
ajax.navigateAdventureTypes=navigateAdventureTypes

# Labels used in Java Adventure Builder application
# Format: label.<formName>.<fieldName>=<Label Text>
label.bookingForm.existingCustomer=Is this an existing customer?
label.bookingForm.customer=If yes, select customer ID
label.bookingForm.headCount=Number of People
label.bookingForm.startDate=Start Date
label.bookingForm.endDate=End Date

# Custom errors for Java Adventure Builder application
errors.bookingForm.startDate.early=The 'start date' must be >= today.
errors.bookingForm.endDate.early=The 'end date' must be >= the 'start date'.

# Optional header and footer for <errors/> tag.
errors.header=<ul>
errors.prefix=<li>
errors.suffix=<br>
errors.footer=</ul>
```

4.12 Appendix – Navigation (Declarative)

The "backChained", "noChained" (which works like 'back' with a button labeled 'no') and use of "redirect" with cancel establish the action chaining termination points for recursive processing. That is, in the following abridged example `struts-config.xml` descriptor, the chaining of actions within a single request/response cycle is configured with "backChained".







Nuance Alert: The "cancelChained" is also supported if a recursive request processor trip is preferred instead of a 'redirect'.

```
<!-- Create booking Step1 related actions -->
<action path="/createBookingStep1Begin"
      <forward name="httpGet" path="/createBookingStep1Pre.do"></forward>
</action>
<action path="/createBookingStep1Pre"
      <forward name="httpGet" path="tile.bookingStep1"></forward>
      <forward name="backChained" path="tile.bookingStep1"></forward>
</action>
<action path="/createBookingStep1Post"
      <forward name ="httpGet" path="tile.bookingStep1"></forward>
      <forward name="continue" path="/createBookingStep2Pre.do"></forward>
      <forward name="cancel" redirect="true" path="/homePagePre.do"></forward>
</action>

<!-- Create booking Step2 related actions -->
<action path="/createBookingStep2Pre"
      <forward name="httpGet" path="tile.bookingStep2"></forward>
      <forward name="continue" path="tile.bookingStep2"></forward>
      <forward name="backChained" path="tile.bookingStep2"></forward>
</action>
<action path="/createBookingStep2Post"
      <forward name="continue" path="/createBookingStep3Pre.do"></forward>
      <forward name="cancel" redirect="true"
path="/createBookingStep1Begin.do"></forward>
      <forward name="back" path="/createBookingStep1Pre.do"></forward>
</action>

<!-- Create booking Step3 related actions         -->
<action path="/createBookingStep3Pre"
      <forward name="httpGet" path="tile.bookingStep3"></forward>
      <forward name="continue" path="tile.bookingStep3"></forward>
      <forward name="backChained" path="tile.bookingStep3"></forward>
</action>
<action path="/createBookingStep3Post"
      <forward name="continue" path="/createBookingStep4Pre.do"></forward>
      <forward name="cancel" redirect="true"
path="/createBookingStep1Begin.do"></forward>
      <forward name="back" path="/createBookingStep2Pre.do"></forward>
</action>
```

4.13 Appendix – tiles.xml

Descriptor document (tiles-defs.xml) example:

```
<tiles-definitions>
	<definition name="layout" path="/jsp/layout/layout.jsp">
		<put name="title" value="JAB" type="string" />
		<put name="mainJSInclude" value="/jsp/include/mainJSInclude.jsp" />
		<put name="customJSInclude" value="" type="string" />
		<put name="header" value="/jsp/layout/header.jsp" />
		<put name="body" value="${body}" />
		<put name="footer" value="/jsp/layout/footer.jsp" />
	</definition>

	<definition name="tile.home" extends="layout">
		<put name="title" value="JAB - HomePage" type="string" />
		<put name="pageName" value="home" type="string" />
		<put name="pageLabel" value="HomePage" type="string" />
		<put name="pageAction" value="home.do" type="string" />
		<put name="pageParent" value="" type="string" />
		<put name="body" value="/jsp/display/home.jsp" />
	</definition>

	<definition name="tile.bookingStep1" extends="layout">
		<put name="title" value="JAB - Create Booking" type="string" />
		<put name="pageName" value="bookingStep1" type="string" />
		<put name="pageLabel" value="Create Booking Step 1" type="string" />
		<put name="pageAction" value="createBookingStep1.do" type="string"
/>
		<put name="pageParent" value="home" type="string" />
		<put name="body" value="/jsp/add/bookingStep1.jsp" />
		<put name="customJSInclude"
value="/jsp/include/bookingStep1JSInclude.jsp" />
	</definition>

</tiles-definitions>
```

4.14 Appendix – web.xml

Descriptor document (web.xml) example:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app
    id="WebApp_ID"
    version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">
    <display-name>JCOEAdventureBuilderStrutsWeb</display-name>

    <!-- FILTERS -->

    <filter>
        <display-name>
        RequestFilter</display-name>
        <filter-name>RequestContextFilter</filter-name>
        <filter-class>com.ford.it.context.RequestContextFilter</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>RequestContextFilter</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!-- LISTENERS -->
    <listener>
        <description>
            This listener loads strutscore-config.xml
        </description>
        <display-name>ScConfigInitializer</display-name>
        <listener-class>
            com.ford.it.strutscore.context.ScConfigInitializer
        </listener-class>
    </listener>
    <listener>
        <description>
            This listener loads fjf-security-plugins-config.xml
        </description>
        <display-name>AuthorizationInitializer</display-name>
        <listener-class>
            com.ford.it.security.plugins.AuthorizationInitializer
        </listener-class>
    </listener>

    <!-- SERVLETS -->
    <servlet>
        <servlet-name>action</servlet-name>
```

```
        <servlet-class>
            org.apache.struts.action.ActionServlet
        </servlet-class>
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <init-param>
            <param-name>debug</param-name>
            <param-value>2</param-value>
        </init-param>
        <init-param>
            <param-name>detail</param-name>
            <param-value>2</param-value>
        </init-param>
        <init-param>
            <param-name>validate</param-name>
            <param-value>true</param-value>
        </init-param>
        <init-param>
            <param-name>definitions-config</param-name>
            <param-value>/WEB-INF/tiles-defs.xml</param-value>
        </init-param>
        <load-on-startup>2</load-on-startup>
    </servlet>

    <servlet>
        <servlet-name>ServletRedirector</servlet-name>
        <servlet-class>
            org.apache.cactus.server.ServletTestRedirector
        </servlet-class>
    </servlet>

    <servlet>
        <servlet-name>JspRedirector</servlet-name>
        <jsp-file>/jspRedirector.jsp</jsp-file>
    </servlet>
    <servlet>
        <description>ServletTestRunner</description>
        <display-name>ServletTestRunner</display-name>
        <servlet-name>ServletTestRunner</servlet-name>
        <servlet-class>
            org.apache.cactus.server.runner.ServletTestRunner
        </servlet-class>
    </servlet>

    <!-- MAPPINGS -->
    <servlet-mapping>
        <servlet-name>ServletRedirector</servlet-name>
        <url-pattern>/ServletRedirector</url-pattern>
    </servlet-mapping>

    <servlet-mapping>
        <servlet-name>JspRedirector</servlet-name>
        <url-pattern>/JspRedirector</url-pattern>
    </servlet-mapping>
```

```
    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
    <servlet-mapping>
        <servlet-name>ServletTestRunner</servlet-name>
        <url-pattern>/ServletTestRunner</url-pattern>
    </servlet-mapping>

    <!-- WELCOME FILE -->
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <error-page>
      <error-code>400</error-code>
      <location>/jsp/exception/clientErrorPage.jsp</location>
    </error-page>
    <error-page>
      <error-code>401</error-code>
      <location>/jsp/exception/clientErrorPage.jsp</location>
    </error-page>
    <error-page>
      <error-code>404</error-code>
      <location>/jsp/exception/clientErrorPage.jsp</location>
    </error-page>
    <error-page>
      <error-code>403</error-code>
      <location>/jsp/exception/authorizationException.jsp</location>
    </error-page>
    <error-page>
      <error-code>408</error-code>
      <location>/jsp/exception/clientErrorPage.jsp</location>
    </error-page>
    <error-page>
      <error-code>500</error-code>
      <location>/jsp/exception/serverErrorPage.jsp</location>
    </error-page>
    <error-page>
      <error-code>501</error-code>
      <location>/jsp/exception/serverErrorPage.jsp</location>
    </error-page>
    <error-page>
      <error-code>502</error-code>
      <location>/jsp/exception/serverErrorPage.jsp</location>
    </error-page>
    <error-page>
      <error-code>503</error-code>
      <location>/jsp/exception/serverErrorPage.jsp</location>
    </error-page>
    <error-page>
      <error-code>504</error-code>
      <location>/jsp/exception/serverErrorPage.jsp</location>
    </error-page>
    <security-constraint>
      <display-name>
```

```
      AllRequests</display-name>
      <web-resource-collection>
            <web-resource-name>AllRequests</web-resource-name>
            <url-pattern>/*</url-pattern>
            <http-method>GET</http-method>
            <http-method>PUT</http-method>
            <http-method>HEAD</http-method>
            <http-method>TRACE</http-method>
            <http-method>POST</http-method>
            <http-method>DELETE</http-method>
            <http-method>OPTIONS</http-method>
      </web-resource-collection>
      <auth-constraint>
            <description>
            AllUsers</description>
            <role-name>AllUsers</role-name>
      </auth-constraint>
    </security-constraint>

    <security-role>
      <role-name>AllUsers</role-name>
    </security-role>

    <!-- RESOURCES -->
    <resource-ref id="ResourceRef_1221502707969">
        <description>JAB non-XA datasource.</description>
        <res-ref-name>JAB_SQLServerNonXA</res-ref-name>
        <res-type>javax.sql.DataSource</res-type>
        <res-auth>Application</res-auth>
        <res-sharing-scope>Shareable</res-sharing-scope>
    </resource-ref>
</web-app>
```

4.15 Appendix – aps-config-property

Related to resource data interaction (i.e. "**Policy to Roles**" APS service) for DLS, application must configure *"***ApplicationResourcePolicyRoleService***"property* group in the APS configuration file. This service name should be matched with the one that has been defined in the **wscore-config.xml** file related to the Policy to Role service.

```
<property-group name ="ApsAPI">
   .
   .
   .
   .
    <!-- Define a Policy to Role web service name -->
    <property-group name="ApplicationResourcePolicyRoleService">
        <property name="ServiceName">
            ApplicationResourcePolicyRoleServiceV1
      </property>
    </property-group>
</property-group>
```