Corporate Services for the Natural Resource Sector

Information Management Branch

Standards for Java Development

Last Updated:	July 20, 2012
Version:	1.1.0
Document:	NRS_Java_Development_Standards_1.1.0

Table of Contents

1. Version Control

Document Version	Revision	Date	Author(s)	Change Reference
1.00		2012-Mar-01	Mark Will	This standard was initially created for MOE/MAL and has been updated to become a Natural Resource Sector standard.
1.0.0	Final	April 2012	L Solomon	Published
1.1.0		July 2012	D. Dawson	Minor edits throughout.

2. Introduction

2.1 Purpose

The intent of this document is to describe the guidelines and standards to be followed when developing web applications using the Java programming language for the Sector. This document will focus on development standards *specific* to the Sector; it is not intended to be a general guide to Java programming. For detailed specifications concerning the Java programming language, refer to the Oracle website.

2.2 *Audience*

This document is directed at those who will be designing, developing and maintaining Java based application systems for the Sector. This includes external contractors, consultants, and business partners, as well as Sector employees.

2.3 *Scope/Exclusions*

The scope of this document covers all Java application development but is primarily focused on web-based deployments. The document does not address the low-level design of an application, but rather provides high level considerations for a starting point for detailed design. Application delivery standards are mentioned but the details are documented within other standards documents.

Where conflicts, if any, are perceived between this document and other standards, the Sector Business Portfolio Manager (BPM) must be consulted.

2.4 *Assumptions*

It is assumed that the audience has a working knowledge of the Java programming language, J2EE, Oracle's Designer (CASE) product, and Oracle relational database technology.

2.5 *Definitions*

The following definitions apply throughout this document.

2.5.1 Standards

A standard is a specific statement of the rules and constraints governing the naming, contents, and operations of software. A standard must be followed. There is a contractual obligation on the part of the vendor/developer to adhere to all relevant standards.

2.5.2 Guidelines

A guideline is a method or custom, which through common usage has become an accepted method of work. A guideline is not enforced, and is not a standard.

2.5.3 Sector

Unless otherwise specified, "Sector" is taken to collectively mean the Ministries and agencies which are included under the umbrella of the Natural Resource Sector. All are served by a

common Corporate Services Division and thus Information Management Branch (IMB) with the mandate to formulate and maintain application development standards.

2.5.4 Automatic Field

The term "Automatic Field" is used to mean a word or phrase within a document that whose value is generated based on information available to the system. In Microsoft Word, these are called "Field Codes". Examples relevant to this document include document filename, page number, total number of pages, and document date.

2.6 *Contacts*

All inquiries regarding these standards should be directed to the Business Portfolio Manager assigned to the project.

3 Deliverables

The following subsections describe the core types of deliverable components expected for each release of any application (as applicable). Additional deliverables may be specified for any application and will normally be indicated in the RFP.

For more detailed specifications regarding this topic refer to the *Application Delivery Standards* and the *Java Application Delivery Standards*; please note that these documents are available only on the Sector Intranet web server -- developers will require a VPN account to access these documents.

3.1 *Data Model*

Before any application can be delivered, the data model for the application must be approved by the Data Administration team. Every project will have a member of this team assigned to it and developers should keep in regular contact with this person while establishing, updating, and extending the data model.

For more detailed specifications regarding this topic refer to the *NRS Data Modelling Standard*.

3.2 *Executable Software*

All deliveries of Java based applications must be in Enterprise Archive (EAR) format. A separate EAR file is required for each application. EAR files will be deployed in the Java EE server under separate context roots, thus leveraging the Java Virtual Machine (JVM) as a mechanism for ensuring that each application is functionally isolated from other applications sharing the same JVM. In addition, the NRS Middleware Stack has multiple JVMs to isolate groups of applications.

NOTE: The Sector has a slightly different requirement for delivering Java-based application releases developed for the legacy JRun environment. Developers should consult with the Architecture Group and the Application Delivery Team well in advance of delivery.

All components that are required by the application to execute to its specification, excluding generally available standard software of the WebADE, must be provided to the Sector. This includes binary code and support files such as images, configuration files, etc.

For more detailed specifications regarding this topic refer to the *Application Delivery Standards* and the *Java Application Delivery Standards.*

3.3 *Source Code*

Source code for all software not included in the standard WebADE components must be provided as part of the project deliverables. This software will become the property of the Province of BC. All of the Java code must be compiled at the Sector site to ensure that there are no syntax errors and that it compiles in the Sector environment.

For more detailed specifications regarding this topic refer to the *Application Delivery Standards* and the *Java Application Delivery Standards*.

3.4 *Build Scripts*

All of the Java deployable components must be compiled at the Sector environment. It is not acceptable to skip the compilation step. The public domain tool Ant, available from the Jakarta project, must be used for component compilation.

The current Ant version is listed in the *Systems and Application Technology Standards Summary*.

NOTE: The Sector has a slightly different requirement for building Java-based application releases developed for the legacy JRun environment. Developers should consult with the Architecture Group and the Application Delivery Team well in advance of delivery.

3.5 *Development Tools*

The term 'IDE' refers to an integrated development environment tool for writing standalone Java classes, HTML pages, Java servlets, and Java Server Pages. Such a tool can assist with the graphical layout of the user interface of a Java-based application, with writing Java code, and with running, testing and debugging the application.

The Sector will permit application developers to use any IDE that generates Java EE 1.7 (or above) compliant code; proprietary code and extensions must not be used in the development of an application (e.g. use of Oracle's Business Components for Java (BC4J) will not be permitted).

3.6 *Documentation*

See Documentation for standards concerning required documentation associated with an application.

4 Development Rules & Guidelines

This section describes the Sector expectations of how applications will be developed from both macroscopic and microscopic perspectives. These expectations are illustrated in terms of rules, which specify what **must** be done, and guidelines, which explain how things **should** be done.

4.1 *Development Process*

The application development and delivery process is described in *Java Application Delivery Standards*.

4.1.1 Quality Assurance (Source Code Reviews)

The Sector has defined formal quality assurance guidelines that will apply to each phase of the *System Development Life Cycle* (SDLC).

The Sector Information Management Branch (IMB) will review all Java source code as part of the Sector Quality Assurance process associated with the delivery of Java based applications to the Sector. Java source code delivered to the Sector ***must*** conform to Sun's Code Conventions for the Java Programming Language.

Java code delivered to the Sector that does not conform to Sun's Code Conventions for the Java Programming Language will not be accepted. Vendors will be required to modify their code to conform to Sun's Code Conventions before the application will be accepted by the Sector. Specifically, the code must pass an automated audit process without generating any errors or warnings.

The Sector employs the *CheckStyle* utility to analyze compliance of Java source code to Sun's *Java Coding Standards*. The Sector CheckStyle configuration file contains a customized sub-set of the rules included in the sun_checks.xml file that is included with the standard SourceForge *CheckStyle* distribution. See in-line comments in env_checkstyle.xml for a explanation of the checks, cross-referencing CheckStyle reference documents where necessary.

CheckStyle plug-ins are available on the SourceForge *CheckStyle* web site for many popular Java Integrated Development Environments (e.g. Eclipse, JBuilder, NetBeans, etc.). Vendors are expected to use CheckStyle within their development environment in order to ensure that Java code is consistent, well structured, and properly documented.

CheckStyle can also be run interactively from the command line or integrated in an Ant build script. The Sector provides a sample Ant build script along with the Sector CheckStyle configuration file, all packaged in a zipfile (*env_checkstyle.zip*).

The Sector may also perform review source code for such things as:

- structure in accordance to commonly accepted programming practices; complete and meaningful documentation
- portability and reusability

NOTE: The Sector has an additional utility for performing basic quality checks on application releases developed for the legacy JRun environment. Developers should consult with the Architecture Group and the Application Delivery Team well in advance of delivery.

4.1.2 Exemption from Standards

Any exemption from Sector system and application technology standards must be approved in writing by the Head of Architecture, IMB. Any such approval must be in place before development. This includes use of components that are outside Sector standards (including third-party java libraries), technology dependencies, failure to use mandatory security components, failure to use common components for common functions, use of unsupported component versions, requirement for components not within standard deployment patterns, or any other deviation from standards.

4.2 *Data Source Configuration*

Developers will need to replicate the data sources that the application will use in production. This involves (i) establishing the structure of the data sources, (ii) replicating those sources in the development environment (i.e. the vendor's environment), and (iii) populating the data tables appropriately for development.

In projects where the data sources already exist, the data source structure can be taken directly from them. In most projects, however, some data modeling will be required. In this case the data sources will be established as defined by the data model.

In some situations, the Sector may provide development-level data sources for developers to use. The Sector will generally supply developers with sample data.

Any vendor to whom data sources are provided must comply with the British Columbia Freedom of Information and Protection of Privacy Act. Attention is particularly drawn to section 30.1, which specifies that "A public body must ensure that personal information in its custody or under its control is stored only in Canada and accessed only in Canada."

Please contact the Sector Business Portfolio Manager for assistance.

4.3 *Coding Style*

The Sector bases its Java programming style on Sun's Code Conventions for the Java Programming Language, as found at the locations:

Code Conventions for the Java Programming Language

Code Conventions for the JavaServer Pages

These conventions include basic guidelines for naming (files and code), code layout, comments, and general programming practices.

Vendors should include JavaDoc style comments in their code as per Sun's Javadoc Standards; each source file should begin with a file header level standard Javadoc comment as shown below.

```
/*
   $Id:  java_standards_23.html,v  1.1  2006/03/31  01:07:06  teglover  Exp  $
```

```
   Copyright  (c)  2006,
   Government  of  British  Columbia,
   Canada
   All  rights  reserved.
   This information contained herein may not be used in whole or in part without the
   express written consent of the Government of British Columbia, Canada.
*/
```

4.4 *Source File Layout*

Java source files must have a specific layout, as described in the following subsections.

For more detailed specifications regarding this topic refer to the How to Write Doc Comments for the Javadoc Tool document.

4.4.1 Header block

Every source file should begin with a simple Javadoc style header block that identifies the file and provides a copyright disclaimer. The "header" comment block should appear as in the following example:

```
/**
 * @(#)MyClass.java
 * Copyright  (c)  2005,  Province  of  British  Columbia.
 * All  rights  reserved.
 */
```

4.4.2 Class Definition

The code defining the class should be immediately preceded by a Javadoc comment describing the class. This comment provides general information about the class and references to related items. A brief explanation or example of how the class can be used is encouraged but not required.

The class definition comment must contain complete Javadoc *@author* tags and a *@version* tag near the end.

Contributors should be listed in order of the date of their first contribution, with the original author listed first.

The Javadoc block for the class should end with *@author* and *@version* tags, as shown:

```
/**
 * This  class  is  used  to  ...
 * ...  etc  ...
 *
 * @author  Jane  Doe
 * @author  John  Doe
 * @version  2.3
 */
```

Use of other tags, such as *@see* and *@link* is encouraged to improve navigation in the generated Javadoc.

4.4.3 Methods, Initialisation Blocks, Variables, and Constants

The Javadoc blocks for each constructor and method, as well as non-private instance variables, must contain *@param*, *@return*, *@throws* Javadoc tags as appropriate. Also, each such block must end with the *@since* tag, as shown, if the product release currently under development is newer than the original version (e.g., not version 1.0):

```
/**
 * Verifies the name against a database of known champions ...
 * ... etc ...
 *
 * @param name
 * the name of the person to verify.
 * @param dbURI
 * the URI of the file containing the database of champions.
 *
 * This is used to obtain a reference in the file system relative
 * to the root location of the application.
 *
 * @return
 * <CODE>true</CODE> if found, </CODE>false</CODE> otherwise.
 *
 * @throws FileNotFoundException
 * if a file corresponding to <CODE>dbURI</CODE> cannot be found.
 *
 * @since 1.0
 */
```

4.4.4 JSP Documentation

A JSP file or fragment file must begin with a server side style comment:

```
<%--
  - Author(s):
  - Date:
  - Copyright Notice:
  - @(#)
  - Description:
--%>
```

This comment is visible only on the server side because it is removed during JSP page translation. Within this comment are the author(s), the date, and the copyright notice of the revision, an identifier and a description about the JSP page for web developers

4.5 *Programming Techniques*

For details regarding programming techniques, please refer to the Code Conventions for the Java Programming Language

.

4.6 *Internal Documentation*

Source for WebADE applications must be thoroughly commented, as described below.

Use Javadoc-style comments for all classes, methods, and class & instance variables, for all access levels. Use "//" syntax for local variables in methods and for large blocks of code, whenever their purpose or behaviour is contextually unclear from the perspective of an experienced programmer who is otherwise uninvolved with the source code. Local variables in

methods need not be commented if their scope is narrow and if their use is readily deduced from the context.

Comment lines should not extend beyond the 72nd character position of a page (assuming fixed-width font).

For more detailed specifications regarding this topic refer to the How to Write Doc Comments for the Javadoc Tool document.

4.6.1 Methods and Instance Variables

A multi-line Javadoc style comment that describes the purpose and/or behaviour of the method, its arguments, its return type, and any exceptions thrown explicitly within the method should precede each method.

Non-private class & instance variables should also be preceded by a Javadoc comment, which may be a single-line comment if brief enough, as in:

```
/** Holds the unprocessed input string */ public String rawInputString;
```

4.6.2 Code Within Methods

Provide regular comments for all major subsections of code, using white space liberally, but not excessively, to distinguish these sections. The major subsections in a typical source file include the overall class declaration, the class and instance variables, the constructors, and the methods.

Comments are not required for obvious items, such as simple "getter" and "setter" bean pattern methods.

4.6.3 Custom Tags

The Javadoc comment for a custom tag class should include a description on how to use the tag in a JSP, including the syntax, parameters, and where specifically the tag should be used within a JSP. An example is desirable.

4.7 *Error Handling*

Applications should track errors, reporting internal problems to the log(s) using Commons Logging and reporting usage errors to the user.

Internal problems usually manifest themselves as Java exceptions and those can usually be handled within the code; otherwise, the problem is normally severe and requires operator intervention to correct.

Interaction errors occur when users attempt to perform an action incorrectly with respect to the "normal" flow of the application, such as entering invalid input into a form or attempting to access a page out of sequence. There are three main issues with either type of error handling:
How to identify them
Where to respond to them
How to display them

4.7.1 Exception handling

When preparing to handle an exception, consider if it can be prevented. The following may provide some ideas:

- Provide default values for anticipated data when their availability and validity cannot be guaranteed.
- Make sure that input fields are trimmed of excess spaces.
- Use defaults to avoid null values unless specifically anticipated.

The sections below provide information on the how to identify the exception as well as how to properly react to the exception within your code.

4.7.1.1 Identification

Error handling should use Java exceptions wherever possible to ensure that the occurrence of errors is explicit and that they must be dealt with. This should include class constructors that cannot tolerate certain conditions (e.g., if a constructor must have all non-null arguments, it should throw an `IllegalArgumentException` so that the caller can abandon the object it attempted to create).

In general, specific exception classes should be defined to handle errors beyond what the standard Java API exception classes are intended for.

The Sector core components add several exception classes. These are introduced in the technical document that comes with the WebADE components.

4.7.1.2 Response

Exceptions should be handled at the point in the code where it makes most sense to do so. For example, exceptions need not be caught "immediately" in the code when they are generated. Instead, methods that generate exceptions, either locally or through a call to another method, can be declared to throw that exception with the expectation that it will be handled "further down the procedure stack".

Since Java does not require handling of some exceptions, developers should be careful to make sure that they handle all exceptions that could occur. Unfortunately, this can sometimes demand the handling of a large number of exceptions, so grouping their "handler" code is encouraged as long as the granularity of response is sufficient. For example, the `Exception` type alone might be caught instead of several different types of exceptions individually, as long as it is sufficient to know that an exception occurred.

When an exception is thrown, the application should catch and attempt to process it immediately if the error can be easily handled at that point and if it makes logical sense to do so. An example of an error that would be easy to handle is if an input field in a Web browser form is incorrectly entered and the application specifies to replace erroneous input with default values. This kind of exception may not even be important enough to log.

4.7.1.3 Displaying

This section describes what types of error conditions should be reported upon detection and to who/where. Errors may be reported to application logs so that operators can review them, or to client display pages so that users are notified, or both. For information on how to log errors, see the Logging section. For information on how to display errors for users, see the section

Errors should be logged if they indicate definite or potential problems with the application or system. They should also be logged if they indicate dangerous actions, whether malicious or not, that could compromise system integrity. Individual applications may call for additional logging of user activity for statistics gathering or monitoring. Critical messages should be sent directly to an operator using e-mail messages. Errors should be reported to the user if they indicate incorrect input or an inability by the system to process the current request. Other messages resulting from error conditions, such as temporary limitations on input or an impending system shutdown, should also be delivered to the user if possible. In no case should stack traces or Java-language-specific messages be delivered to users. The Java environment provides the means to capture events that would otherwise result in this kind of message being sent to users so that a more appropriate message can be returned instead.

4.7.2 Interactive error handling

4.7.2.1 Identification

Interactive errors are due to incorrect use of the application by users and can be detected during the processing of requests as part of the application's business logic.

4.7.2.2 Response

When interactive errors occur, the flow of the application may be affected depending on (i) the type of error, (ii) the type of request, and (iii) the current state of the user's session (possibly also the current state of the system). For example, a user might enter invalid input into a form, forcing the application to request that the form be resubmitted with corrected input.

4.7.2.3 Displaying

Regardless of how application flow is affected, users should be notified of errors that have occurred due to incorrect use of the application. For example, if a user has entered invalid form input, then that should be pointed out when the user is requested to resubmit the form.

4.8 *Logging*

The activity of applications must be logged so operators can trace errors and can monitor performance/usage issues. Developers must use the Apache group's Jakarta Commons Logging tool (Log4J) for logging in Java code.

Developers may configure Log4J as they see fit to support their own in-house development, but in preparing the delivery release, they **must** ensure that the logging is re-configured correctly for production. This means:

- sensitive information, such as passwords, are not logged

- logging is set to ERROR in the delivered source tar file. The logging config file must include the WebADE entries:

```
log4j.category.ca.bc.gov.webade=error
log4j.logger.ca.bc.gov.webade=error
```

- logging is **not**, repeat **not**, sent to standard output (i.e. stdout)
- log files are directed to /apps_ux/logs

- log files follow the format below

```
Log File Name: <appname>.log
Log File Type: Rolling  Log  File
Max Backups: 6
Max File Size: 1MB
Log File Format: [date/time] - [priority] - [thread] - [file:line] - message
```

- This Log File Format implies that

```
*.layout.ConversionPattern=[%d{ISO8601}] - [%-5p] - [%t] - (%F:%L) - %m%n
```

Developers may use the basic priorities provided with Commons Logging. The following is a description of the levels and their uses as specified in the Commons Logging documentation:

- **fatal** - Severe errors that cause premature termination. Expect these to be immediately visible on a status console.
- **error** - Other runtime errors or unexpected conditions. Expect these to be immediately visible on a status console.
- **warn** - Use of deprecated APIs, poor use of API, 'almost' errors, other runtime situations that are undesirable or unexpected, but not necessarily "wrong". Expect these to be immediately visible on a status console.
- **info** - Interesting runtime events (startup/shutdown). Expect these to be immediately visible on a console, so be conservative and keep to a minimum.
- **debug** - detailed information on the flow through the system. Expect these to be written to logs only.
- **trace** – more detailed information. Expect these to be written to logs only.

Again, ERROR must be set in the delivered source tar file.

The following example illustrates how to implement Commons Logging in Java code:

```
... etc ...

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;

... etc ...

public class Foo {
private static Log log = LogFactory.getLog(Foo.class);

... etc ...
```

```
public void doFoo( String msg ) {
log.info( "doFoo(String): This is a log statement" ); if ( msg == null ) {
log.warn( "doFoo(String): parameter msg is null" );
```

NOTE: The Sector has an additional requirement for configuring logging on application releases developed for the legacy JRun environment. Developers should consult with the Architecture Group and the Application Delivery Team well in advance of delivery.

4.9 *XML Compressors & Accelerators*

In instances where an application is required to perform heavy XML or GML processing, the requirements and the methodology proposed should be discussed with the IMB's Architecture Group before the design is finalized.

4.10 *Java Naming Conventions*

4.10.1 Servlets

The main servlet in a WebADE application must be called ControllerServlet. Any other servlet class must have the Servlet suffix in its name.

4.10.2 JavaBeans

Each other bean class should have the Bean suffix in its name and should be a valid Java bean (i.e. is serializable, has getter+setter methods, and has a "no-arguments" constructor).

4.10.3 Tags

Each custom tag class must have the "Tags" suffix in its name. The .tld file must be placed in the application's WEB-INF directory. Also, if the tag classes are contained in a separate library archive (jar) from the main application classes, this archive should be placed in the WEB-INF/lib directory.

Identification of tags to an application within the web.xml configuration file will use the application short name or webade as the prefix, as in:

```
<%@ taglib uri="/WEB-INF/xyzTags" prefix="xyz" %>

<%@ taglib uri="/WEB-INF/webadeTags" prefix="webade" %>
```

4.10.4 Other Classes and Components

Classes that extend other classes from standard WebADE components, including Struts, J2EE, and JDBC drivers, should be named consistently in a manner that reflects their association (and purpose). For example, an extension of the Struts action class:

```
org.apache.struts.action.Action
```

should contain the suffix Action.

Other classes, as well as JSP pages and other targets, should be named using general conventions.

4.10.5 Java Package Naming

Every servlet and accompanying Java class will belong to a Java package. One main package should be defined for an application, with possible sub-packages, containing all of the application-specific classes. The name of the main package should be the same as the short application name.

All WebADE packages must have the following prefix:

```
ca.bc.gov.srm.app
```

This will be followed by the short name of the specific application, in lower case to conform to package naming conventions. The package for the XYZ application would be:

```
ca.bc.gov.srm.app.xyz
```

The fully-qualified package name of a top-level action class in the XYZ application called MyStrutsAction would be:

```
ca.bc.gov.srm.app.xyz.MyStrutsAction
```

NOTE: Some classes, other than the standard tools, are intended to be common to more than one (and in most cases all) applications. These are the "common" WebADE classes, and those developed specifically for the Sector will belong in or below the package:

```
ca.bc.gov.webade
```

4.11 *Sector Database Object Naming*

The names of stored procedures and data tables should be prefixed by the application's acronym followed by the underscore character, as in:

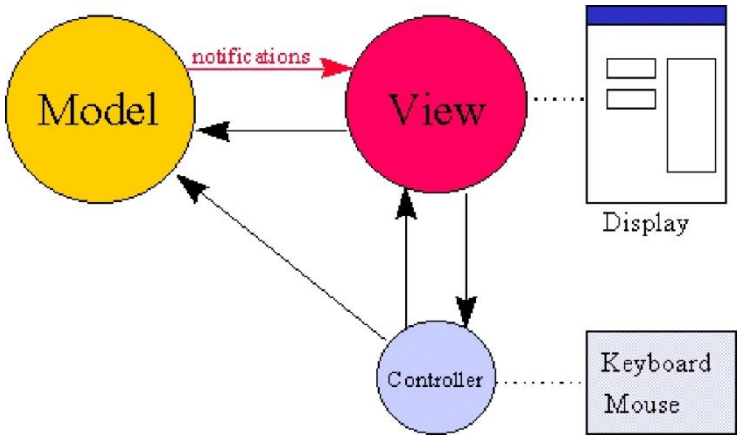
```
XYZ_getInspectionReport()  //  sample  procedure
XYZ_AvailableReports       //  sample  table  name
```

Reference the *Object Naming Conventions* section of the *Oracle Designer Standards and Guidelines* for more detailed information in this area. Standards for versioning of Data Definition Language (DDL) and database objects may be found in the *Application Delivery Standards* document.

5 Application Design

5.1 *Architecture Overview*

Sector Web applications follow the Model-View-Controller (MVC) "type 2" design pattern; the MVC architecture was originally developed by Xerox PARC and was used as the framework for developing the GUI interface for Microsoft Windows and Apple's Mac O/S. A schematic representation of the Model-View-Controller scheme is shown in the figure below.



Model-View-Controller Scheme

The MVC "type 2" scheme as applied to Java application development consists of the following:

- A single controller servlet that receives, validates, and directs incoming requests;
- Action classes that process requests (one class per request type), set data within corresponding state beans, and return results from the processed request;
- State bean classes that contain state information for requests and/or results;
- JSPs to present the outcome of a request.

The controller servlet determines which "action" is requested, uses the appropriate action class to process the request, and then uses the result to select the appropriate JSP to generate the presentation of the outcome of the request. The controller servlet provides the action class instance with the user-input data through an instance of the action-specific state bean. This bean is also retained in the user session for future use.

The action class will use data from the action-specific state bean and will perform transactions with the database to process the request. The action class next determines the result of its processing, acquires and populates a state bean instance corresponding to the result, then returns the result to the controller servlet. The state bean instance is retained in the user session for future use.

The JSP selected for output based on the processing result obtains the appropriate state bean instance from the user's session and uses this bean to generate the presentation content. The bean is passed to the JSP as a session attribute.

General JSPs are intended for presentation and therefore should only be used at the end of a chain of processing on the server. For example, an HTTP POST request will initially be processed by a "controller" servlet to make sure it is in the correct sequence. The servlet may use instances of other Java classes to further process the data. Finally, the servlet will hand off presentation processing to a JSP page. A general design goal for JSPs is that they contain no embedded Java code (i.e., Java scriptlets). Java beans and tags (standard and custom) should be employed to acquire the required content and formatting.

Variables should be defined in as narrow a scope as practical so that their purpose and content is easy to determine by a reader unfamiliar with the code. This strategy also minimizes coding errors. For example: Simple "one-off" variables such as loop counters should be defined within the loop structure. Variables needed in multiple locations, such as in a "try" block plus one or more of its corresponding "catch" blocks should be declared in a sufficiently higher level that they are "available" in each location.

5.2 *Deployment Pattern*

Sector applications must conform to a standard Oracle/Java deployment pattern; any application components that are not accommodated in the standard pattern require an Exemption from Standards as described in this document.

The standard Oracle/Java deployment pattern is laid out in
http://www.env.gov.bc.ca/csd/imb/3star/arch/docs/Deployment_Patterns.pdf

5.3 *Movel View Controller Framework*

The Sector standard is to employ the use of Spring Web MVC for building web applications with Java Servlet and JavaServer Page (JSP) technology. Spring Web MVC encourages application architectures based on the Model-View-Controller (MVC) design paradigm.

Spring Web MVC includes the following primary areas of functionality:

- A controller servlet that dispatches requests to appropriate Action classes provided by the application developer.
- JSP custom tag libraries, and associated support in the controller servlet, that assists developers in creating interactive form-based applications.
- Utility classes to support XML parsing, automatic population of JavaBeans properties based on the Java reflection APIs, and internationalization of prompts and messages.

5.4 *Java Server Pages*

JSPs are intended for presentation and therefore should only be used at the end of a chain of processing on the server. For example, an HTTP POST request will initially be received and examined by a "controller" servlet. The servlet may use instances of other Java classes to process the request and store results of the processing. Finally, the servlet will hand off presentation processing to a JSP page.

A general design goal for JSPs is that they contain no embedded Java code (i.e., Java scriptlets). The common components of the WebADE include custom tags to help reduce the need for Java scriptlets and to promote a consistent look & feel. Developers should produce a custom tag library for an application and should use state beans to present data from a session or request context in a page to further reduce code.

5.5 *Abstract Persistence Layer - Object Relational Mapping*

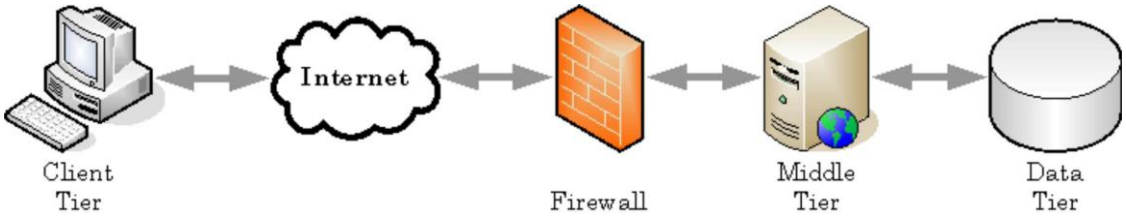
The Sector currently permits three data access models for object-relational mapping (ORM):

- Hibernate ORM
- a manual mapping of objects to database tables;
- the use of Oracle stored procedures for mapping objects to database tables.

The first, Hibernate ORM, is the preferred model.

6 Application Environment

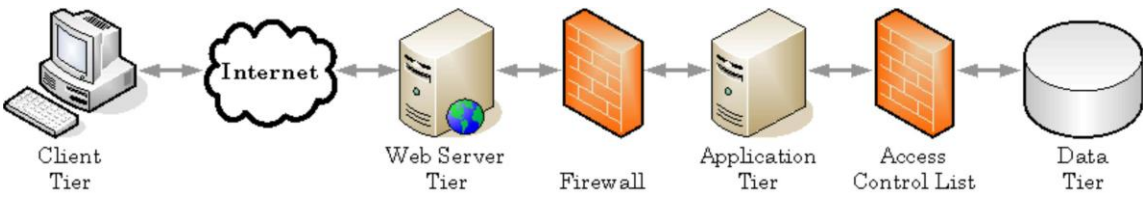
The Sector has implemented Java based web applications in a three-tier deployment based upon the Java 2 Platform Enterprise Edition (Java EE) specification. The Sector standard is to build and deploy Java applications based on server-side processing; use of applets is not permitted in the design or construction of Sector Java applications.



Component Tier	Component Name
Client Tier (Intranet)	Internet Explorer
Client Tier (Extranet)	Internet Explorer & Netscape
Middle Tier	Oracle Fusion Middleware (WebLogic)
	Java Enterprise Edition (Java EE)
	Sector standard Java Libraries
	Sector Common Components
Data Tier	Oracle Database
Reporting	Oracle Reports Server

Component versions are listed in the Systems and Application Technology Standards.

The Sector is moving towards a conventional 3-tier deployment to conform to standards defined by the office of the CIO through the *Security Enhancement Project*.



Component Tier	Component Name

Client Tier (Intranet)	Internet Explorer
Client Tier (Extranet)	Internet Explorer & Netscape
Web Server Tier	Apache
Application Tier	Oracle Fusion Middleware (WebLogic)
	Java Enterprise Edition (Java EE)
	Sector standard Java Libraries
	Sector Common Components
	Oracle Reports Server
Data Tier	Oracle Database

6.1 *Client Tier (Workstation)*

Java EE applications use a thin-client interface. A thin-client is the distributed portion of the application where the client program does not process data, but instead passes data to the server for processing.

All applications must be built using server-side java. No applications shall be built using client-side Java. Applications shall not have any dependency to client-side java runtimes such as JInitiator or JRE.

Operations such as database queries and execution of business rules are off-loaded to servlets or Java Beans (including EJB's) executing on the Java EE server where they can leverage the security, speed, services and reliability of Java EE server-side technologies.

Developers building public web applications must ensure that their applications work with most common web browsers including Internet Explorer Version 7 and above. Generated HTML code must comply with the HTML 5 Standard and XML should comply with the XML 1.1 Standard.

Developers building applications that use the Sector Internet Map Framework (IMF) must design their applications to use Microsoft's Internet Explorer (IE) version 6 or above; the IMF has been optimized for IE and will not function correctly when used with other browsers.

6.2 *Middle Tier (Internet Application Server)*

A middle tier application server provides a runtime environment for the application and provides access to business functions and data. The Sector standard is Oracle Fusion Middleware (OFM).

The Sector is moving to a middle tier architecture in which the web server (Apache) tier is separated from the application tier. In the target model, the web server tier is in the DMZ and the application tier is behind the firewall in a segmented security zone.

NOTE: The Sector has a legacy JRun application server environment on which a large number of existing applications are expected to require maintenance using that platform until the planned update. Developers should consult with the Architecture Group.

6.3 *Data Tier (Oracle Database)*

Oracle is the chosen database platform of the BC Government and has been adopted as the standard database by most ministries.

7 Security

The central component for managing application security is WebADE. For information on WebADE (see http://www.webade.org/). The WebADE supplies various APIs for use in authorizing a user to an application. This framework used in conjunction with the government supported global user directories (IDIR, BCeID and MyID) is used to authenticate and authorize all users of an application.

The security framework selected by the Sector for web-based applications is intended to provide a secure, scalable mechanism for protecting the Sector data. The framework can be classified by one of the following categories of access:

- **Public Access;** no authentication required;
- **Basic (a.k.a. Individual)**; identified but neither trusted nor verified;
- **Personal (a.k.a. Verified Individual)**; Individuals with a confirmed identity; identified and verified
- **Business:** bodies outside of government that have a business relationship with government; trusted;
- **Internal:** internal users (e.g. employees); trusted;

These categories of access can be further differentiated as follows:

Public Access - must be query only; access to data is restricted through the use of a generic database account which remains hidden from the end-user at all times.

Basic (a.k.a. Individual, or MyID) - the identity of the party connecting to the Sector application server must be determined; the user is challenged to enter a unique username and password; user authentication will be effected against the MyID directory service. A Secure Socket Layer (SSL) implementation is required to ensure confidential data sent from the client is encrypted. Database connections are to be made using a generic account in order to take advantage of database connection pooling.

Personal (a.k.a. Verified Individual) - the identity of the party connecting to the Sector application server must be determined; the user is challenged to enter a unique username and password; user authentication will be effected against the BCeID directory service. Personal is a person who has a personal rather than a business relationship with government (e.g. related to an individual's personal health records). Both query and insert/update operations will be permitted. A Secure Socket Layer (SSL) implementation is required to ensure confidential data sent from the client is encrypted. Database connections are to be made using a generic account in order to take advantage of database connection pooling.

Business - the identity of the party connecting to the Sector application server must be determined; the user is challenged to enter a unique username and password; user authentication

will be effected against the BCeID directory service. The Business category applies to registered businesses, partnerships, proprietorships, foreign governments, etc. Note that Business may also apply to an individual whose interactions with government are of a business rather than a personal nature (e.g. buying products or services from the province). Both query and insert/update operations will be permitted. A Secure Socket Layer (SSL) implementation is required to ensure confidential data sent from the client is encrypted. Database connections are to be made using a generic account in order to take advantage of database connection pooling.

Internal - the identity of the party connecting to the Sector application server must be determined; the user will be challenged to enter a unique username and password; user authentication will be effected against the IDIR directory service. Both query and insert/update operations will be permitted. A Secure Socket Layer (SSL) implementation is required to ensure that confidential data sent from the client is encrypted. Database connections are to be made using a generic account in order to take advantage of database connection pooling.

Note: If the database account used by the application is for a named user (e.g. IDIR or BCeID), the username and IP address should be logged by the application.

7.1 *Authentication*

User authentication for applications will be implemented using the BC Government's Enterprise Security Gateway, built on Netegrity SiteMinder. The service provides the "Common Login Page" (CLP) that ***must*** be used by all applications requiring user authentication. The CLP takes advantage of the Single Sign On (SSO) feature of Siteminder and also provides features such as expired passwords/password change and enforcement of Terms of Use agreements.

The SSO feature of the SiteMinder product allows a user to use multiple government e-services (SiteMinder enabled) while only having to authenticate on the initial request. WebADE fully supports SiteMinder and its use is transparent to the developer.

Conceptually the Authentication (SSO enabled) process is as follows:

- The client attempts to access a resource within the application (welcome page) that is protected by Netegrity.
- Netegrity checks for SSO secure cookie (user already signed in via another service)
    - If cookie exists and user has access, the user will be granted access to the resource without having to log in again.
    - If cookie does not exist, the user is redirected to the Common Login Page. Once logged in correctly, the user is then redirected back to the original application request (welcome page).

A "WebADE Developer Module" is available in order to emulate the government Siteminder environment for the purposes of application development. See http://www.webade.org/.

7.2 *Authorization*

Sector standard is to use WebADE for authorization. General introduction and documentation on WebADE is available at http://www.webade.org/.

In addition to identifying the client, an application must know whether or not a client is authorized for any request it makes to the application.

The WebADE provides several methods in its API, along with underlying constructs, to support the association of users with actions. The technical documentation provided with the WebADE software describes this through detail and example.

Chart A

Authorisation to application actions

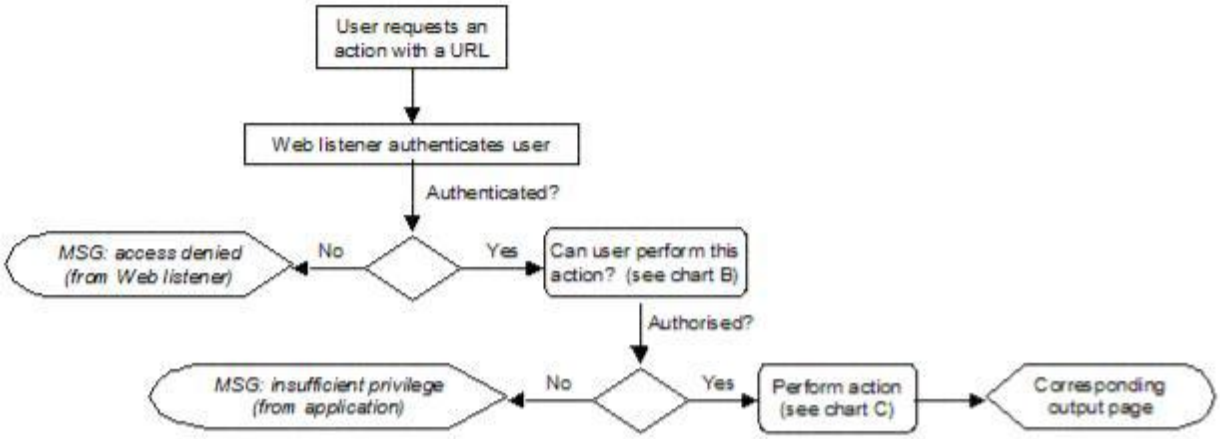
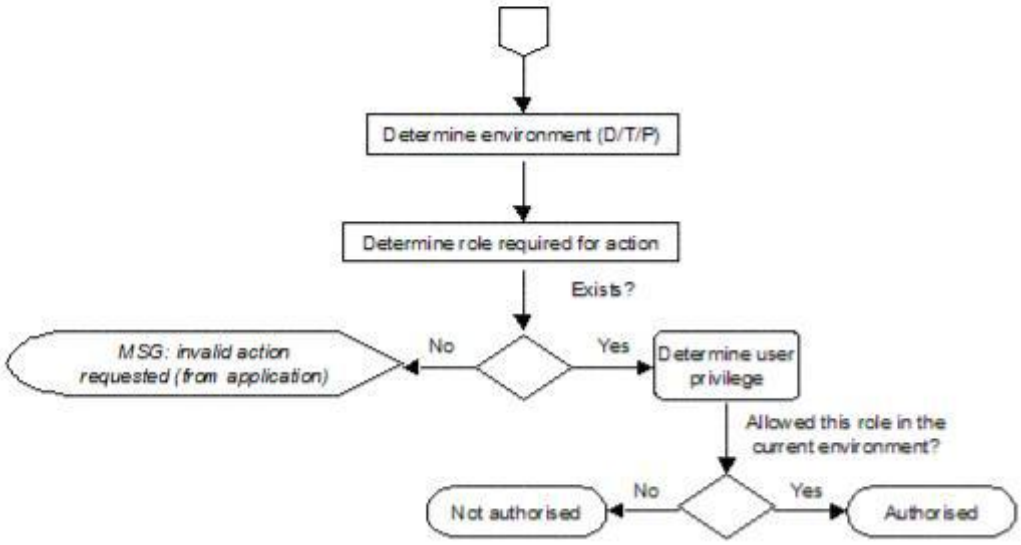
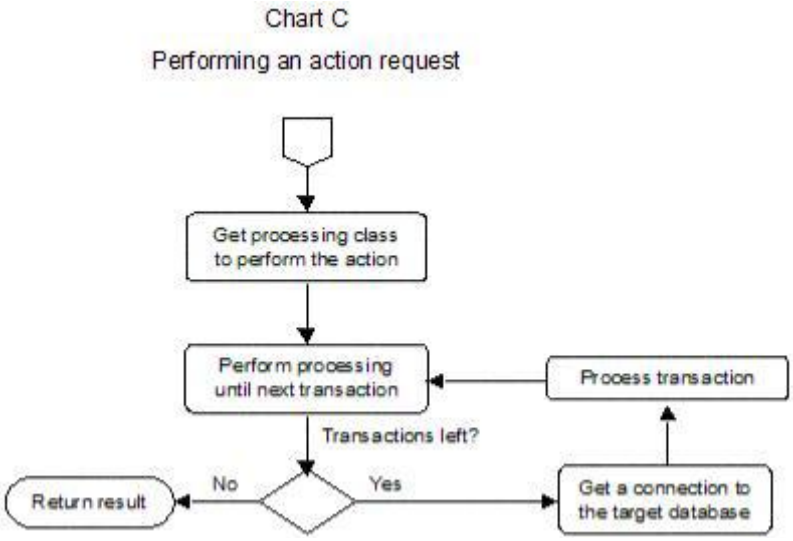


Chart B

Determining user privilege





7.3 *Database Access*

After a client has been authenticated by the Web server and authorized to the application, processing of the request begins. This might involve any number of transactions with any number of databases (usually just one). Applications access the database through "generic" proxy accounts, rather than through specific accounts corresponding to the client initiating the request. There is a proxy account for each unique level of access needed to the database. For example, there could be an "admin" account, a "power user" account, and a "general user" account.

These proxy accounts correspond to the roles defined for the application. Therefore, in the example just provided, there would be 3 roles: one corresponding to the "admin" account, one corresponding to the "power user" account, and one corresponding to the "general user" account. The names do not have to match, but are typically similar; they are associated by the WebADE configuration for the application. Note that since a 1-to-1 mapping exists between database proxy accounts and roles, any task requiring multiple data sources will have to use multiple roles (with suitably identifying names).

Note that all connections to the database are pooled. Therefore a pool exists for every type of connection (i.e., every proxy account) required by the application to every data source. Developers will not have to establish or manage these pools --- the core WebADE components will do this and will provide a simple interface to allow the application to acquire connections on demand (assuming it is authorized for the current action request).

Database connections are not created in application code; instead, they are acquired from the WebADE, which is responsible for creating and managing them. Therefore, account and database connection information should not appear in an application's code or local configuration.

Developers must contact their Sector Business Portfolio Manager in order to obtain a list of proxy accounts and application roles that will correspond to connection pools to be used by the application. The developer will also need to provide minor configuration information for the development and deployment environments such as # of initial connections, data source URL, id, password, etc. The details on how to do this relatively simple task are provided in the WebADE technical documentation.

7.4 *Sector Database Role Naming*

The names of database roles are subject to naming standards. Refer to the *Oracle Designer Standards and Guidelines*:

http://www.env.gov.bc.ca/csd/imb/3star/sdlc/4design/des6i_std/des6i_std.html

for more detailed information in this area. The Oracle Roles must be reflected in the submitted Oracle Designer data model as defined in those standards.

7.5 *Enforcing Application Flow*

Normal flow for application requests that involve processing (i.e. not including fetches of static items such as image files) should follow the Model-View-Controller pipeline as described in the Architecture Overview section. The presence of JSPs makes this even more important because they have the capability to acquire or present information intended for a restricted audience. A clever user can directly access a JSP, bypassing the normal pipeline through the controller servlet and the Struts action processing classes.

The WebADE provides a simple means to avoid this problem:

- A custom tag is available to verify that a request has followed the MVC pipeline
- If included near the top of a JSP, this tag will perform the verification before allowing the remainder of the JSP to be processed
- If the verification fails, then the tag will direct the request to a configurable target (e.g. an error page).
- The tag can be included in any or all JSPs where this check is require

In order to protect applications from Cross-Site Scripting attacks, the Enterprise Security Gateway (Siteminder) will reject URLs that contain certain characters in either the base URL or the query string (parameters).

Applications must not use any of the following characters in URL or query string (parameters), whether escaped or not:

- single quote ('), character %27
- left angle bracket (<), character %3c
- left angle bracket (<), character %3c

Note that escaping of the characters does not mask them from detection. Thus, all of the following sequences will be rejected as they all resolve to a single left angle bracket (<): %3c, %253c, %25%33%43, %25%32%35%33%43, etc.

8 Reports

The Sector has selected Oracle Reports 11g as the standard tool for report generation. Developers can develop Oracle Reports using the standard client-side reporting tools (i.e. Oracle Reports Builder) and then deploy the report modules on the Sector dedicated reports server.

Standards for developing web based Oracle Reports for the Sector may be found in the document Oracle Developer Standards and Guidelines. The Sector has also developed standards for the delivery and deployment of web based Oracle Reports; these *Reports Server Delivery Standards* are available to authorized users on the Sector Intranet server.

NOTE: The Sector JRun-based legacy environment has a large installation of applications with reports built using Crystal Reports 10 and deployed using Crystal Enterprise 10 together with a custom Sector code library for invoking these reports within WebADE applications. In some cases it may be necessary to perform maintenance on these applications using the Crystal technology so developers should consult with the Architecture Group early in the design phase for direction.

8.1 *Designing Reports*

Details on designing reports, including Sector standards, can be found in a separate document on the Application Development Web site.

8.2 *Charting*

The Sector has deployed the JFreeChart library as part of the standard Oracle Fusion Middleware 11g (OFM 11g) configuration. This is considered mandatory for charting in Sector applications. Further information for the product can be found using the link below:

- JFreeChart

9 Common Components

9.1 *Mandatory Common Components*

Any component listed in the *Systems and Application Technology Standards Summary* document is mandatory when the functionality it provides is a requirement of the application. This includes but is not limited to:

- WebADE for security
- JFreeChart for charting
- Log4J for logging (mandatory in all applications)

9.2 *Additional Java Libraries*

A list of standard java libraries is provided in the *Systems and Application Technology Standards*. Use of any other third party library, whether commercial, GPL or from any other source, is considered outside of standard and requires an Exemption from Standards as described in this document.

10 Documentation

10.1 *Release Notes*

A properly constructed, complete, and up-to-date release notes document is vital for every release. Sector personnel rely on this document to determine what work must be done to ensure successful deployment of a release.

10.2 *Application Use*

A single general user guide will be provided for the application. Normally this will be the responsibility of the application custodian to produce in consultation with the developer; however, it may be included in the specific deliverables for the project.

The user guide will be written in Microsoft Word format according to a template, which will be available on the component workbench along with special instructions for creating the document according to Sector standards.

10.3 *Application Configuration*

Two types of documentation covering the configuration of the application are required:

- A set of technical instructions for compiling the application, referred to as the Readme in the *Java Application Delivery Standards.*
- A description of the configuration to assist non-programmers in understanding the functional composition of the application.

10.4 *Instructions for Compilation*

The Sector uses the Apache Jakarta Ant utility to compile and build Web applications. Any special requirements for compiling the application must be clearly documented. This will include:

- A description of the platform requirements
- Environment settings
- Organization of all software components

The current Ant version is listed in the *Systems and Application Technology Standards*.

10.5 *Design Revisions to As-Built*

A final deliverable for an application is a revised version of the application's System Design document. If there are any deviations from the System Design document submitted to the Sector prior to commencement of the system build (e.g. database version), these should be reflected in a re-delivered version of the System Design. This final revision should be versioned as a minor point release - e.g., a final revision of the 3.2.0 design document (for version 3.2.0 of the application) would be versioned as 3.2.1.