Coding Standards for ATG Development

01/08/2008

Retail / eCommerce

Maheswar Karra
m.karra@tcs.com

Introduction

This document contains the coding standards that can be used in an ATG development project.

ATG Coding Standards

1. Declare class variables with a prefix of "m" (eg:mVariableName). But get/set methods can follow the following convention – getVariableName() and setVariableName().
2. Boolean accessor methods should have a prefix of "is" or "can" or "has" etc. Example: isEnabled().
3. Declare the parameters passed to a method with a prefix of "p".
4. Declare local variables within a method without any prefix.
5. All System Exceptions should be logged in the error log.
6. All Business Exceptions should be logged in the debug log
7. Form Handler should be designed to catch any exceptions that might occur, and display an appropriate message from the resource bundle in the HTTP response.
8. Do not reference a Sessionholder from the Tools.
9. Error messages should be handled through resource bundles.
10. All data objects/beans should implement toString() method. This can be used when logging messages.
11. Place all constants in a Constants interface class.
12. Any class which uses constants should implements Constants interface class
13. All Exceptions in the service layer should be propagated up to the application layer.
14. Component scope should be appropriate. For example, don't use a globally scoped component to hold session specific values.
15. Droplets should not contain html code.
16. Use isLoggingDebug check and logging debug extensively in all important methods.
17. LoggingError should always be true.
18. All logged messages should follow the following pattern: loggingDebug(CLASSNAME:METHODNAME: SEVERITY:message);
19. All error messages should be prepended with appropriate severity as per CVS logging standards (Eg. P1, P2 etc).
20. For repository XML files, use appropriate caching strategy and size.
21. There should be no changes to any ATG out-of-the-box DB tables. All additions should be made to CVS-specific tables.
22. For any ATG classes that are subclassed, ensure that modifications are done in a future-migration-friendly way. For example: If modifying the out-of-the-box Login code, add new code to existing "pre" and "post" methods that ATG has provided.

23. The Dynamo Scheduler service executes a schedulable component in a thread outside the HTTP request handling thread pool. Schedulable components must catch all exceptions and log an error message.

24. Package and import statements need to be organized. First, we should import cvs packages, then ATG packages and at last import Java packages.

    Eg:

```
package cvs.aaaa.bbbb;

// CVS
import cvs.packages.Class1;
import cvs.packages.Class2;
//ATG
import atg.nucleus.GenericService;
// Java
import java.util.Calendar;
```

25. All classes and interfaces must belong to a named package.
26. Package name should have only lower case characters.
27. Avoid naming non-fields with the prefix 'm'.
28. No need to import a type that lives in the same package.
29. Avoid modifiers which are implied by the context.
30. Avoid assignments in operands.
31. Each class should declare at least one constructor.
32. It is a good practice to call super() in a constructor.
33. The statement should not have any unnecessary parentheses.
34. Use explicit scoping instead of the default package private level.
35. Avoid instantiating String objects
36. Avoid calling toString() on String objects
37. Avoid concatenating nonliterals in a StringBuffer constructor or append().
38. Using equalsIgnoreCase() is cleaner than using toUpperCase/toLowerCase().equals().
39. Avoid appending characters as strings in StringBuffer.append.
40. String.indexOf(char) is faster than String.indexOf(String).
41. Avoid unused private fields.
42. Avoid unused local variables.
43. Avoid unused private methods.
44. Avoid printStackTrace(); use a logger call instead.
45. Method name should begin with a lower case character.
46. Class names should begin with an uppercase character.
47. Avoid having a field name matching the declaring class name.
48. Avoid having a field name with the same name as a method.

49. Use ArrayList instead of Vector.
50. Switch statements should have a default label.
51. Avoid instantiation through private constructors from outside of the constructor's class.
52. The default label should be the last label in a switch statement.
53. Avoid using equals() to compare against null.
54. Avoid protected fields in a final class. Change it to private or package access.
55. A switch statement should always contain a break statement.
56. An abstract class should contain abstract methods.
57. Always use equals() to compare object references.
58. If an empty constructor is used, then purpose of usage should be documented.
59. Always log the exception before rethrowing Caught Exception because original stack trace may be lost.
60. Avoid empty catch blocks.
61. Avoid empty try blocks.
62. Avoid empty finally blocks.
63. Avoid empty 'if' statements.
64. Avoid empty synchronized blocks.
65. Avoid empty while statements.
66. Avoid returning from a finally block.
67. Do not use 'if' statements that are always true or always false.
68. Avoid creating BigDecimal with a decimal (float/double) literal. Use a String literal.
69. Do not start a literal by 0 unless it's an octal value.
70. Consider replacing this Enumeration with the newer java.util.Iterator.
71. Consider replacing this Vector with the newer java.util.List.
72. Consider replacing this Hashtable with the newer java.util.Map.
73. Avoid instantiating Integer objects. Call Integer.valueOf() instead.
74. Finalize should do something besides just calling super.finalize().
75. Finalize methods should not be overloaded.
76. Last statement in finalize method should be a call to super.finalize().
77. If you override finalize(), make it protected.
78. Avoid calling finalize() explicitly.
79. Avoid using if statements without curly braces.
80. Avoid using 'while' statements without curly braces.
81. Avoid using 'if...else' statements without curly braces.
82. Avoid using 'for' statements without curly braces.
83. A catch statement should never catch throwable since it includes errors.

84. A method/constructor shouldn't explicitly throw java.lang.Exception.
85. Avoid throwing null pointer exceptions.
86. A catch statement that catches an exception only to rethrow it should be avoided.
87. Abstract classes should be named 'AbstractXXX'.
88. Always import the necessary classes not the full package.
89. Do not use public non-final class fields.
90. Declare only one variable per line.
91. Connections should be wrapped by a finally statement that calls cleanup.
92. Possible unsafe assignment to a non-final static field in a constructor.
93. Instead of copying data between two arrays, use System.arrayCopy method.
94. Avoid using enum as an identifier; it's a reserved word in JDK 1.5.
95. Avoid an instanceof check being performed on the caught exception. Create a separate catch clause for this exception type.
96. Avoid throwing certain exception types. Instead use a subclassed exception.
97. Avoid instantiating Boolean objects; you can usually invoke Boolean.TRUE instead.
98. Usage of the Collection.toArray() method will throw a ClassCastException if an array of the desidered class is not passed as a parameter to Collection.toArray() method.
99. Ensure that resources (like Connection, File Pointer objects) are always closed after use.
100. Avoid constructor calls a Overridable method.
101. Avoid methods with high Cyclomatic Complexity. (Value of Cyclomatic Complexity can be configured).
102. Don't Use log Debug in the Catch Block use logError method and pass the two parameters
103. Don't Use logDebug method in the Catch Block, Instead use logError method.
104. Some ‘for’ loops can be simplified to while loops - this makes them more concise.(example : “for (;true;)” can be replaced with “while (true)”).
105. Avoid instantiating Integer objects. Call Integer.valueOf() instead.
106. A logError in a catch block should also include the exception object to log the stack trace.
107. Avoid using implementation types (i.e., HashSet), instead use the interface (i.e, Set).
108. Tools methods should throw only CVSException or CVSBusinessException.
109. Do not references services directly, instead reference the interface.
110. Null check should not be applied on a variable. If the variable is null you'll get a NullPointerException.
111. Avoid assigning a "null" to an Object (outside of its declaration).
112. Avoid using the resolveName in the FormHandler to get reference to a componenet from another. Instead add them to the properties.
113. A signature (constructor or method) shouldn't have Exception in throws declaration.
114. Singleton instances should not be used. Use ATG Global instance instead.
115. System.out.print should not be used. Instead use a logger.

116. Avoid unnecessary constructors - the compiler will generate these for you.

117. Avoid an operation on an Immutable object (BigDecimal or BigInteger). The result of this operation is a new object.

118. Avoid the use of overriding method if it merely calls the same method defined in a superclass.

119. Call Thread.notifyAll() rather than Thread.notify()

Layout of Java Source Files (*.java)

The layout for a class will be broken up into the following main sections:

a. Copyright section
b. Java File Description
c. Package Name
d. Imports
e. Constants
f. Member Variables (protected. Then private)
g. Constructors
h. Methods
i. Inner Classes

Each section will be prefaced by an appropriate header block defining the section.

Copyright Section

```
/*
 * Class Name: RxRefillValidationWorker.java
 *
 * Copyright 2007 Corporation. All Rights Reserved
 * This software is the proprietary information of Client Name.
 * Use is subject to license terms.
 *
 * @author: Author Name
 *
 */
```

Package Name

Package names should occur on the first non-commented line of the source file and should following the naming conventions defined in this document.

Imports

Immediately following the package name should be the imported class names. Do not use .* for imports. List out each class explicitly that is needed for the source class.

File Description

This is a brief description of what the file is, does and represents. It should describe the overview of how to use the file (or classes therein). JavaDoc compatible commenting style is required.

```
/**
 * This class schedules batch job for auto terminating dependent access and sending
    Termination Notice to parents / guardian before 45 days of Dependent access
    termination.
 *
 **/
```

JSP CODING STANDARDS

120. Use ResourceBundles to contain all jsp labels, title bars, etc.
121. Error URL and Success URL is set in FormHandler from JSP Page.
122. Form method 'Get' is not used. Instead 'Post' method is used always.
123. HTML comments have not been used - this is to prevent comments getting dispatched to the client's browser.
124. Every form should have only one FormHandler enclosed within it.
125. ErrorForEach droplet is used to display error messages in the web page resulting from a form submission.
126. There are no validations in the JSP except for Javascript validations. All other Form validations are performed in the Form Handler.

Layout of JSP Files (*.jsp)

The layout for a jsp page will be broken up into the following main sections in order:

a. JSP File Description
b. JSP page directive(s)
c. Optional tag library directive(s)
d. Page Imports
e. JavaScript section
f. HTML and JSP code
g. Comments Section

Each section will be prefaced by an appropriate header block defining the section.

Copyright Section

```
<%--
 * JSP Name: CurrentPrescriptions.jsp
 *
 * Copyright 2007 CVS Caremark Corporation. All Rights Reserved
 * This software is the proprietary information of CVS Caremark Corp.
 * Use is subject to license terms.
 *
 * @author: Mohamed Sadath
 *
--%>
```

JSP File Description

This is a brief description of what page does and represents.

```
<%-- ======================================== Description
```

```
     * This page …
     *
     * @updated $DateTime: 2007/09/03 14:20:07
  * Description: This page fragment displays Refill Reminders and Number of linked accounts of a
                 user.

     --%>
```

Taglibs

This first non comment section should be the taglib declaration section

E.g.:

```
<%-- ========================================= Taglibs --%>
<%@ taglib uri="/paf-taglib" prefix="paf" %>
<%@ taglib uri="/core-taglib" prefix="core" %>
<%@ taglib uri="/dsp" prefix="dsp" %>
```

Page Imports

This next section should contain dsp:importbean and page imports

```
<%-- ========================================= Imports --%>
<%@ page import="atg.repository.Repository" %>
<%@ page import="atg.repository.RepositoryItem" %>
<%@ page import="atg.repository.RepositoryItemDescriptor" %>
<%@ page import="atg.beans.DynamicPropertyDescriptor"%>
<dsp:importbean bean="/atg/dynamo/droplet/ForEach"/>
```

Comments

Any block of code which requires commenting should use java comments. Comments should clearly state the purpose and function of the code.

```
<%-- This is a comment --%>
```

Embedded java

Embedded java should follow the Java nomenclature and documentation standards.

Where practical, use custom tags or droplets rather than using inline java.

If inline Java is used, the section should be commented heavily as to why the use of inline java was necessary.