1 Introduction

This document aims to present the outcome of a code inspection perfomed over the ServiceEventHandler class of Apache OfBiz project. The class can be found at
apache-ofbiz-16.11.01/apache-ofbiz-16.11.01/framework/webapp/src/main/java/org/apache/ofbiz/webapp/event/ServiceEventHandler.java

2 Functional Role

The main functionality of this class is to invoke a service. In particular, this class invokes a service whose request has been triggered by an event. To do that, this class checks that all the parameters are well formed, and sets up a map filled with those parameters. The definitions of such maps have been encoded in XML files that are located at
apache-ofbiz-16.11.01/framework/service/servicedef/services.xml

A brief explanation on how this xml maps are used and how the service framework works can be found at **here**

3 Checklist

3.1 Naming Conventions

Nothing to say

3.2 Indention

Nothing to say

3.3 Braces

There are three if statements followed by one instruction not included between braces at lines 106, 253, 394

3.4 File Organization

3.5 Wrapping Lines

3.6 Comments

There's a commented out block of code in lines 180-184, whithout any explanation

3.7 Java Source Files

Javadoc is minimal. It describes only the main goal and doesn't help to fully understand the way it's achieved. There's no javadoc for the static method, but it is widely commented.

3.8 Package and Import Statements

Nothing to say about that

3.9 Class and Interface Declarations

The invoke method is too long. As it deals mostly with multiple checks on inputs and environmental setup, it could be splitted in different subroutines

3.10 Initialization and Declarations

The variable serviceName is delcared and initialized to Null on line 93, but is later set to event.invoke, on line 102. We suggest to directly intialize serviceName to event.invoke.
Many declarations appear in the middle of a code block. This may be due to the fact the the invoke method is too long. A possible solution to this problem could be using private methods, making the whole class more readable.
Here are the lines and the declarations that are not compatible with this rule:

- Line 86:
  - DispatchContext dctx = dispatcher.getDispatchContext();
- Line 92 and 93:
  - String mode = SYNC;
  - String serviceName = null;
- Line from 109 to 112:
  - Locale locale = UtilHttp.getLocale(request);
  - TimeZone timeZone = UtilHttp.getTimeZone(request);
  - HttpSession session = request.getSession();
  - GenericValue userLogin = (GenericValue) session.getAttribute("userLogin");
- Line 115:
  - ModelService model = null;
- Line 132:
  - boolean isMultiPart = ServletFileUpload.isMultipartContent(request);
- Line 133:
  - Map<String, Object> multiPartMap = new HashMap<String, Object>();

- Line 146 and 147:
  - String sizeThresholdStr = EntityUtilProperties.getPropertyValue("general", "http.upload.max.sizethreshold", "10240", dctx.getDelegator());
  - int sizeThreshold = 10240;
- Line 155 and 156:
  - String tmpUploadRepository = EntityUtilProperties.getPropertyValue("general", "http.upload.tmprepository", "runtime/tmp", dctx.getDelegator());
  - String encoding = request.getCharacterEncoding();
- Line 159:
  - ServletFileUpload upload = new ServletFileUpload(new DiskFileItemFactory(sizeThreshold, new File(tmpUploadRepository)));
- Line 162:
  - FileUploadProgressListener listener = new FileUploadProgressListener();
- Line 171:
  - List<FileItem> uploadedItems = null;
- Line 234 and 235:
  - Map<String, Object> rawParametersMap = UtilHttp.getCombinedMap(request);
  - Set<String> urlOnlyParameterNames = UtilHttp.getUrlOnlyParameterMap(request).keySet();
- Line 238:
  - Map<String, Object> serviceContext = new HashMap<String, Object>();
- Line 249:
  - Object value = null;
- Line 312:
  - List<Object> errorMessages = new LinkedList<Object>();
- Line 336:
  - Map<String, Object> result = null;
- Line 361:
  - String responseString = null;

3.11 Output Format

Some outputs signlaing errors throughout the code are a bit general. Maybe the author should provide additional information on these problems. These outputs can be found at:

- Line 120 and 124: the output displays a "problems getting the service model" message

- Line 175: the output displays a "problem reading uploaded data" message
- Line 357 and 358: the output is a generic "service invocation error" message

Moreover, from line 400 to 411 the output displays a long message. We suggest to use "\n" to make the output error more readable

3.12 Exceptions

Line 151 and 152 contain an inconsistency. The error message clearly states "Unable to obtain the threshold size from general.properties; using default 10K", implying that sizeThreshold will be set to 10240. On line 152, however, we find that sizeThreshold is set to -1. Either the error message or the assignment at line 152 is wrong.

4 Checklist

Naming Conventions

1. All class names, interface names, method names, class variables, method variables, and constants used should have meaningful names and do what the name suggests.
2. If one-character variables are used, they are used only for temporary âĂIJthrowawayâĂİ variables, such as those used in for loops.
3. Class names are nouns, in mixed case, with the first letter of each word in capitalized.
4. Interface names should be capitalized like classes.
5. Method names should be verbs, with the first letter of each addition word capitalized.
6. Class variables, also called attributes, are mixed case, but might begin with an underscore ('_') followed by a lowercase first letter. All the remaining words in the variable name have their first letter capitalized.
7. Constants are declared using all uppercase with words separated by an underscore.

Indention

8. Three or four spaces are used for indentation and done so consistently.
9. No tabs are used to indent.

Braces

10. Consistent bracing style is used, either the preferred âĂIJAllmanâĂİ style (first brace goes underneath the opening block) or the âĂIJKernighan and RitchieâĂİ style (first brace is on the same line of the instruction that opens the new block).

11. All `if`, `while`, `do-while`, `try-catch`, and `for` statements that have only one statement to execute are surrounded by curly braces.

File Organization

12. Blank lines and optional comments are used to separate sections (beginning comments, package/import statements, class/interface declarations which include class variable/attributes declarations, constructors, and methods).

13. Where practical, line length does not exceed 80 characters.

14. When line length must exceed 80 characters, it does NOT exceed 120 characters.

Wrapping Lines

15. Line break occurs after a comma or an operator.

16. Higher-level breaks are used.

17. A new statement is aligned with the beginning of the expression at the same level as the previous line.

Comments

18. Comments are used to adequately explain what the class, interface, methods, and blocks of code are doing.

19. Commented out code contains a reason for being commented out and a date it can be removed from the source file if determined it is no longer needed.

Java Source Files

20. Each Java source file contains a single public class or interface.

21. The public class is the first class or interface in the file.

22. External program interfaces are implemented consistently with what is described in the javadoc.

23. Javadoc is complete (i.e., it covers all classes and files part of the set of classes assigned to you).

Package and Import Statements

24. If any package statements are needed, they should be the first non-comment statements. Import statements follow.

Class and Interface Declarations

25. The class or interface declarations shall be in the following order:
    (a) class/interface documentation comment;
    (b) class or interface statement;
    (c) class/interface implementation comment, if necessary;
    (d) class (static) variables;
        i. first public class variables;
        ii. next protected class variables;
        iii. next package level (no access modifier);
        iv. last private class variables.
    (e) instance variables;
        i. first public instance variables;
        ii. next protected instance variables;
        iii. next package level (no access modifier);
        iv. last private instance variables.
    (f) constructors;
    (g) methods.
26. Methods are grouped by functionality rather than by scope or accessibility.
27. Check that the code is free of duplicates, long methods, big classes, breaking encapsulation, as well as if coupling and cohesion are adequate.

Initialization and Declarations

28. Variables and class members are of the correct type. Check that they have the right visibility (public/private/protected).
29. Variables are declared in the proper scope.
30. Constructors are called when a new object is desired.
31. All object references are initialized before use.
32. Variables are initialized where they are declared, unless dependent upon a computation.
33. Declarations appear at the beginning of blocks (block: any code surrounded by curly braces '`{`' and '`}`'). The exception is a variable can be declared in a `for` loop.

Method Calls

34. Parameters are presented in the correct order.
35. Check that the correct method is being called, or should it be a different method with a similar name.
36. Method returned values are used properly.

Arrays

37. There are no off-by-one errors in array indexing (that is, all required array elements are correctly accessed through the index).
38. All array (or other collection) indexes have been prevented from going out-of-bounds.
39. Constructors are called when a new array item is desired.

Object Comparison

40. All objects (including Strings) are compared with `equals` and not with `==`.

Output Format

41. Displayed output is free of spelling and grammatical errors.
42. Error messages are comprehensive and provide guidance as to how to correct the problem.
43. The output is formatted correctly in terms of line stepping and spacing.

Computation, Comparisons and Assignments

44. Implementation avoids âĂIJbrutish programmingâĂİ[1].
45. Check order of computation/evaluation, operator precedence and parenthesizing.
46. Liberal use of parenthesis is used to avoid operator precedence problems.
47. All denominators of a division are prevented from being zero.
48. Integer arithmetic, especially division, is used appropriately to avoid causing unexpected truncation/rounding.
49. Comparison and Boolean operators are correct.

[1] See `http://users.csc.calpoly.edu/~jdalbey/SWE/CodeSmells/bonehead.html` for more information.

50. Check throw-catch expressions, and check that the error condition is actually legitimate.

51. Check that the code is free of any implicit type conversions.

Exceptions

52. Check that the most relevant exceptions are caught.

53. Appropriate action are taken for each catch block.

Flow of Control

54. In a `switch` statement, check that all cases are addressed by break or return.

55. All switch statements have a default branch.

56. All loops are correctly formed, with the appropriate initialization, increment and termination expressions.

Files

57. All files are properly declared and opened.

58. All files are closed properly, even in the case of an error.

59. EOF conditions are detected and handled correctly.

60. All file exceptions are caught and dealt with accordingly.