# Code Inspection

Version 1.0

Matteo Farè, Federico Feresini, Thierry Ebali Essomba

January 5, 2016

# Contents

# Chapter 1

# Assigned Classes

The two classes assigned are *ExtendedAccessLogValve* and *AccessLogValve*, and both are in the *org.apache.catalina.valve* package. The path of the the classes is:
4.1.1/appserver/web/web-core/src/main/java/org/apache/catalina/valves.

We have reviewed the method

```
decodeAppSpecific(String fields, int i, FieldInfo fieldInfo)
```

from the first class and the methods

```
setPattern(String pattern)
```

```
postInvoke(Request request, Response response)
```

from *AccessLogValve*

# Chapter 2

# Functional Role of Set of Classes

The main function of the two classes is basically the same: creating log files. The only difference is that *ExtendedAccessLogValve* creates log files which conform to the Working Draft for the Extended Log File Format defined by the W3C. Thus, while *AccessLogValves* recognises pattern made of literal text strings, prefixed by the '%' character, the other class pattern attribute are made up of format tokens, that can need an additional prefix, such as c for "client", s for "server" and others.
All the informations here described were discovered reading the javadoc of the two classes (even though some methods presents an extremely poor javadoc and others don't even have one) and looking at the code and at the comments. Furthermore, in order to better understand the function of the classes, we have retrieved some informations on the web.
Now we are going to list some of the functions of each class, and explain the role of the method inspected.

## 2.1 AccessLogValve

The class AccessLogValve generates a web server access log with a configurable pattern. Getting into the details , the logged message may include constant text or any of the following replacement strings:

- %a - Remote IP address
- %A - Local Ip address
- %b - Bytes sent, excluding HTTP headers, or '-' if no bytes were sent
- %B - Bytes sent, excluding HTTP headers
- %h - Remote host name

- %H - Request protocol
- %l - Remote logical username from identd (always returns '-')
- %m - Request method
- %p - Local port
- %q - Query string (prepended with '?' if it exists, otherwise an empty string
- %r - First line of the request
- %s - HTTP status code of the response
- %S - User session ID
- %t - Date and time, in Common Log Format
- %u - Remote user that was authenticated
- %U - Request URL path
- %v - Local server name
- %D - Time taken to process the request, in milliseconds
- %T - Time taken to process the request, in seconds

In addition, the caller can specify one of the following aliases for commonly utilized patterns: Common - %h %l %u %t %r %s %b Combined - %h %l %u %t "%r" %s %b "%Refereri" "%User-Agenti"
It also provides the possibility to write information from the cookie, incoming header, the Session or something else in the ServletRequest:

- %{xxx}i for incoming headers
- %{xxx}c for a specific cookie
- %{xxx}r xxx is an attribute in the ServletRequest
- %{xxx}s xxx is an attribute in the HttpSession

### 2.1.1 SetPattern

The setPattern method is defined as a public void, line 447. It uses constants defined into the class Constants.java, placed into the same package org.apache.catalina.valves. The analyzed method sets pattern (variable defined into the same class as private string by the line 239) into one of the feasible options according to the input string:

```java
    /**
    * Set the format pattern, first translating any recognized
    alias.
    *
    * @param pattern The new pattern
    */
    public void setPattern(String pattern) {
    [...]
    }
```

If the input string correspond to the constant `COMMON_ALIAS` (defined as "common"), the pattern variable assumes the value of `COMMON_PATTERN`(defined as "%h %l %u %t "%r" %s %b")

```java
if (pattern.equals(Constants.AccessLog.COMMON_ALIAS))
  pattern = Constants.AccessLog.COMMON_PATTERN
```

if the input string correspond to the constant `COMBINED_ALIAS` (defined as "combined"), the pattern variable assumes the value of `COMBINED_ALIAS` (define as "%h %l %u %t "%r" %s %b "%Refereri" "%User-Agenti"")

```java
if (pattern.equals(Constants.AccessLog.COMBINED_ALIAS))
  pattern = Constants.AccessLog.COMBINED_PATTERN
```

Then it assigns to the local variable pattern the value of the parameter received as inupt. The method proceeds analysing the pattern value assigned by the previous statements, and defines the value of common (variable defined into the same class as private boolean, line 226) and the value of combined (variable defined into the same class as private boolean, line 233) according to the used pattern.

### 2.1.2 postInvoke

As the invoke method, postInvoke logs a message specifying the request and the response, according to the format specified by the pattern properties. First of all the method postInvoke analyzes if the input parameter is valid, and if the used pattern is common or combined (it is defined by the previous class setPattern).

```
if (condition!=null &&
null!=request.getRequest().getAttribute(condition)) {
return;
}
```

```
// Check to see if we should log using the "common" access
 log pattern
if (common || combined) {
[...]
}
```

According to the different kinds of patterns, the postInvoke method updates the variable 'result' (defined as StringBuilder, line 639) in order to create an appropriate message based on the defined pattern. After generating a variable 'result' which contains references to remote user, date and time of the sent request, remote host, user-agent, etc.. the method keeps on updating the variable 'result' by extracting from the variable 'pattern' (defined as private String, line 239. And settled by the method setPattern, from line 447 to line 467) possible information contained by braces at the beginning of the string variable pattern.

## 2.2 ExtendedAccessLogValve

- Logs a message summarizing request and response.
- Renames the existing log file.
- Returns the client to server data.
- Returns the server to client data.
- Logs a specified message to the log file.
- Decodes app specific data.
- Opens a new log file for the specified date.
- Decodes given patterns.

### 2.2.1 decodeAppSpecific

As it is possible to read in the javadoc, this method receives as inputs a String that contains the pattern to decode, the index where the pattern begins in the string, and a parameter used to store the result. These are the types of pattern that this method recognises:

1. x-A(...)
2. x-R(...)
3. x-C(...)
4. x-S(...)
5. x-H(...)
6. x-P(...)

It checks if the pattern is well formed: the capital letter be one of those listed above and it has to contain an opening and a closing bracket. If one of these two condition isn't valid, the method returns -1 and writes an error message on the log. Otherwise it check some parameters of the pattern and saves the necessary informations in the third input parameter. It returns the new index of the input string.

In order to understand all these informations, comments have been very helpful.

```
/* Move past 'x-' */
i+=2;
```

```
/* test that next char is a ( */
if (i+1!=fields.indexOf('(',i)) {
log.log(Level.SEVERE, WRONG_X_PARAM_FORMAT);
return -1;
}
```

```
 i+=2; /* Move inside of the () */
```

```
/* Look for ending ) and return error if not found. */
int j = fields.indexOf(')',i);
if (j==-1) {
log.log(Level.SEVERE, X_PARAM_NO_CLOSING_BRACKET);
return -1;
}
```

# Chapter 3

# List of Issues

In this section we will specify for each method inspected the issues we have found, according to the points of the checklist of the Code Inspection Assignemnt Document. Since there are some points regarding parts of the class the method belongs to, it's important to remember that the first method belongs to *ExtendedAccessLogValve*, and the other two to *AccessLogValve*.

## 3.1 decodeAppSpecific

- [12] A comment line should separate instance variables from constructors.
- [13] Some lines exceed 80 characters (lines 1527 and 1543).
- [18] In this method, comments are adequately used, but in other parts of the class they could be used in a better way, since some blocks of code are very hard to understand.
- [23] The javadoc of this method is complete, but others are extremely poor and some methods don't even have one (i.e. postInvoke, line 669).
- [25] There are some errors in the declarations order:
    1. Constructor should be declared after instance variables.
    2. Some static variables are declared after the constructor.
- [27] Some method of this class are quite long (i.e. decodePattern, 119 lines).
- [44] In the support class FieldInfo (line 1560) is used a set of named constant, an enum class could be a better solution.

## 3.2 setPattern

- [9] The whole method code presents indention issue. Each line, from 447 to 467, is indented by the used of tabs instead of the used of the spaces.

- [11] This method presents a lot of if-else statements which imply the use of braces. The whole method should have been written as:

```java
public void setPattern(String pattern) {

  if (pattern == null)
  {
  pattern = "";
  }
  if (pattern.equals(Constants.AccessLog.COMMON_ALIAS))
  {
  pattern = Constants.AccessLog.COMMON_PATTERN;
  }
  if (pattern.equals(Constants.AccessLog.COMBINED_ALIAS))
  {
  pattern = Constants.AccessLog.COMBINED_PATTERN;
  }
  this.pattern = pattern;

  if (this.pattern.equals(Constants.AccessLog.COMMON_PATTERN))
  {
  common = true;
  }
  else
  {
  common = false;
  }

  if (this.pattern.equals(Constants.AccessLog.COMBINED_PATTERN))
  {
  combined = true;
  }
  else
  {
  combined = false;
  }
  }
```

The “Allman” style was chosen (first brace goes underneath the opening block)

- [12] The whole class presents an issue of file organization. Blank lines are used even to separate statement into the same method with a logical point of view. Statements which perform the same logical algorithm are not separated, but the statements which performs different tasks into the same class are separated by blank lines. The rewritten code in the previous paragraph presents this kind of issue; blank lines split the code into three parts, each one sets the value of a different variable (pattern, common and combined).

- [25] AccessLogValve presents a problem of class declaration. First problem is about the order of the constructor. It should follow the class static variables and the instance variables, but it is declared first of all the statements into the class.

- [27]Even if the class is mainly composed by short methods, it contains some methods which are longer than the average length, such as the private method replace, starting from line 884, ending to the line 1006. Or the public method postInvoke, starting from line 620, ending to the line 760.

## 3.3 postInvoke

- [1] Line 622: the long variable "t2" has a name that not immediately suggests its meaning. it would be better to use a self-explanatory name
- [9] All the lines of the method show the use of tabs for indention contrary to good practice. The division of the statement at line 631 is correct because it is good practice to start a new line after an operator, but the indention is not correct.
- [11] In several blocks of the method the use of parentheses does not conform to good practice.

  1. Line 648

```
 if (isResolveHosts())
 result.append(req.getRemoteHost());

EXACT SOLUTION:

 if (isResolveHosts()){
 result.append(req.getRemoteHost());
 }
```

  2. Line 656

```
  if (value == null)
  result.append("- ");

 EXACT SOLUTION:

  if (value == null){
  result.append("- ");
  }
```

  3. Line 692

```
   if (length <= 0)
   value = "-";

  EXACT SOLUTION:

   if (length <= 0){
   value = "-";
   }
```

4. Line 702

```
if(referer != null)
result.append(referer);

EXACT SOLUTION:

if(referer != null){
result.append(referer);
}
```

5. Line 711

```
if(ua != null)
result.append(ua);

EXACT SOLUTION:

if(ua != null){
result.append(ua);
}
```

- [12] In the method we can see a frequent use of blank lines to separate lines of code that belong at the same section. It's a good practice use blank lines to separate beginning comments, package/import statements, class/interface declaration etc.
- [15] Line 736 -744: it's not used the same convention to break a line.
- [23] This method doesn't have a javadoc.
- [33] There are some issues about declarations:
  1. Line 629: declaration of the variable "time" should be done at the beginning of the block. The initialization position is however correct because obtainable only after the previous computation
  2. Line 638: declaration and initialization of the variable "data" should be done at the beginning of the block since it is independent from the previous computation. it can be observed, however, that in the case in which occurs the first return instruction (line 626) declaration and initialization of the variable are useless because the object is not used.
  3. Line 639: declaration of the variable "result" should be done at the beginning of the block. The object is correctly created with the use of the constructor.
  4. Line 690: declaration and initialization of the variable "length" should be done at the beginning of the method. The same consideration of the object "data" can be done for it