# Inspection Document

Version 1.0

Giorgio Pea(Mat. 853872), Andrea Sessa(Mat. 850082)

5/1/2016

# Contents

# 1 Introduction

The document that follows contains the results of an inspection process on the classes and the methods listed below.
This process is based on a given checklist(see reference section) that enumerates a series of code errors and software aspects to be considered. Additional thoughts over the quality and the functional role of the software artifacts analyzed are also mentioned.

# 2 Classes

Included in this section are the two java classes under analysis.

File: /appserver/web/web-core/src/main/java/org/apache/catalina/ssi/**SSIServlet.java**

Methods under inspection:

- *init()*

- *requestHandler(HttpServletRequest req, HttpServletResponse res)*

- *processSSI( HttpServletRequest req , HttpServletResponse res , URL resource )*

File: /appserver/web/web-core/src/main/java/org/apache/catalina/ssi/**SSIMediator.java**

Methods under inspection:

- *substituteVariables(String val)*

## 3 Functional Role

Included in this section is some information about the functional behavior of the analyzed classes and methods.

### 3.1 SSIServlet.java

From the Javadoc:

```
78  /**
79   * Servlet to process SSI requests within a webpage. Mapped to a path
        from
80   * within web.xml.
81   *
```

This class represents a Java EE Servlet used to process HTTP requests that include some SSI instructions.

SSI(Server Side Include) is a interpreted server-side scripting language, that is usually used to include the contents of one or more files into a web page.

Follows a description of the methods analyzed for this class.

- **Init()**
  From the javadoc of the method:

```
104      /**
105       * Initialize this servlet.
106       *
107       * @exception ServletException
108       *               if an error occurs
109       */
```

  This method takes care of initialize the Java EE Servlet by retrieving and setting the configuration parameters.
  The role of these configuration parameters is explained below:

    - **debug**: A flag used to manage debug messages: if debug = 0 then no debug messages are logged
    - **isVirtualWebappRelative**: A flag used to manage how file paths in SSI instructions are processed: if isVirtualWebappRelative = true then all the file paths are considered relative to the root of the web-server.
    - **expires**: Specifies the expiration time for HTTP requests managed by this Servlet(in seconds)
    - **buffered**: A flag used to manage how the content of HTTP responses processed by this Servlet are written on the output stram: if buffered = true then this writing is buffered(see processSSI() for additional explanations)
    - **inputEncoding**: Specifies the characters encoding of the HTTP requests processed by this Servlet

– **outputEncoding**: Specifies the characters encoding of the HTTP responses processed by this Servlet

- *requestHandler()*

  From the Javadoc of the method:

```
173    /**
174     * Process our request and locate right SSI command.
175     *
176     * @param req
177     *          a value of type 'HttpServletRequest'
178     * @param res
179     *          a value of type 'HttpServletResponse'
180     */
```

This method takes care of process HTTP get or post requests managed by this Servelet. In particular requestHandler() retrieves the url associated with a given request, checks if it corresponds to an existing resource on the web server, sets some configuration parameters for the processed HTTP request, and invokes processSSI().

In this block of code the method checks the debug flag of the Servelet; If debug is greater than zero then logs a message into the logger for debug purposes.

```
183        ServletContext servletContext = getServletContext();
184        String path = SSIServletRequestUtil.getRelativePath(req);
185        if (debug > 0)
186            log("SSIServlet.requestHandler()\n" + "Serving "
187                    + (buffered?"buffered ":"unbuffered ") + "
                         resource '"
188                    + path + "'");
```

In this block of code the method checks if the url associated with the given request does not exists or points to a resource in the 'WEB-INF' or 'META-INF' subdirectories, if so the method terminates and logs an error message.

```
189        // Exclude any resource in the /WEB-INF and /META-INF
                subdirectories
190        // (the "toUpperCase()" avoids problems on Windows systems)
191        if (path == null || path.toUpperCase(Locale.ENGLISH).
                startsWith("/WEB-INF")
192                || path.toUpperCase(Locale.ENGLISH).startsWith("/
                    META-INF")) {
193            res.sendError(HttpServletResponse.SC_NOT_FOUND, path);
194            log("Can't serve file: " + path);
195            return;
196        }
```

In this block of code the method tries to retrieve the resource associated with the url of the given request. If this resource doesn't exist the method terminates and logs an error message.

```
197         URL resource = servletContext.getResource(path);
198         if (resource == null) {
199             res.sendError(HttpServletResponse.SC_NOT_FOUND, path);
200             log("Can't find file: " + path);
201             return;
202         }
```

In this block of code the method sets some header fields of the HTTP response to be processed: it sets the mime type, the characters encoding of the output text and the expiration time(in seconds, see init()). At the end of the method the processSSI() method is invoked.

```
203         String resourceMimeType = servletContext.getMimeType(path);
204         if (resourceMimeType == null) {
205             resourceMimeType = "text/html";
206         }
207         res.setContentType(resourceMimeType + ";charset=" +
                outputEncoding);
208         if (expires != null) {
209             res.setDateHeader("Expires", (new java.util.Date()).
                getTime()
210                 + expires.longValue() * 1000);
211         }
212         req.setAttribute(Globals.SSI_FLAG_ATTR, "true");
213         processSSI(req, res, resource);
```

- **processSSI()**
  *No Javadoc is available for this method*
  This method parses the SSI commands contained in a given HTTP request via the SSIProcessor class and writes the result of this processing in the output stream of the Servlet.

  In this block of code the method takes care of the initialization of the stream used to read the data contained in a given resource and checks the characters encoding of this data according to the configuration of the Servlet

```
233         URLConnection resourceInfo = resource.openConnection();
234         InputStream resourceInputStream = resourceInfo.
                getInputStream();
235         String encoding = resourceInfo.getContentEncoding();
236         if (encoding == null) {
237             encoding = inputEncoding;
238         }
239         InputStreamReader isr;
240         if (encoding == null) {
241             isr = new InputStreamReader(resourceInputStream);
242         } else {
243             isr = new InputStreamReader(resourceInputStream,
                    encoding);
244         }
245         BufferedReader bufferedReader = new BufferedReader(isr);
```

  In this block of code the method parses the SSI commands contained in the web page by using the process() method of the SSIProcessor class. The results are written on the Servlet output stream.

```
222         SSIProcessor ssiProcessor = new SSIProcessor(
                ssiExternalResolver)

247         long lastModified = ssiProcessor.process(bufferedReader,
248                 resourceInfo.getLastModified(), printWriter);
```

  In this block of code the method sets different types of output stream for the given HTTP response accordingly to the buffered flag of the Servlet, and flushes them.

```
226         if (buffered) {
227             stringWriter = new StringWriter();
228             printWriter = new PrintWriter(stringWriter);
229         } else {
230             printWriter = res.getWriter();
231         }

252         if (buffered) {
253             printWriter.flush();
254             String text = stringWriter.toString();
255             res.getWriter().write(text);
256         }
```

## 3.2 SSIMediator.java

From the Javadoc:

```
75 /**
76  * Allows the different SSICommand implementations to share data/talk to
        each
77  * other
78  *
```

This class takes care of how many different implementations of the SSI instructions can communicate and exchange data with each other.

Follows a description of the methods analyzed for this class.

- **substituteVariables()**
  From the Javadoc:

```
246     /**
247      * Applies variable substitution to the specified String and
              returns the
248      * new resolved string.
249      */
```

This method processes a given string in a way that HTML special characters are normalized and SSI variable are replaced with their actual value.

In this block of code the method checks that the given string contains '$' or '&, if not so the method terminates returning the original string(no processing is needed)

```
251         // If it has no references or HTML entities then no work
252         // need to be done
253     if (val.indexOf('$') < 0 && val.indexOf('&') < 0) return val;
```

In this block of code the method normalizes HTML special characters in the given string.

```
255         // HTML decoding
256     val = val.replace("&lt;", "<");
257     val = val.replace("&gt;", ">");
258     val = val.replace("&quot;", "\"");
259     val = val.replace("&amp;", "&");
```

In this block of code the method takes care of replacing '&#n' with 'n', where 'n' is an integer number, in the given string. See the Javadoc of StringBuilder(Java SE 7 class) for a detailed explanation of the methods.

The remaining code replaces SSI variables with their current value.

Variables are always in the form '$ varName' and could possibly be wrapped, ie. '${varName}'. This information has been collected by an direct analysis of the code and by means of the few comments inserted. The actual value of the variables found in the given string is retrieved using the 'getVariablesValue()' method(also defined in SSIMediator.java).

```
261        StringBuilder sb = new StringBuilder(val);
262        int charStart = sb.indexOf("&#");
263        while (charStart > -1) {
264        int charEnd = sb.indexOf(";", charStart);
265            if (charEnd > -1) {
266                char c = (char) Integer.parseInt(
267                        sb.substring(charStart + 2, charEnd));
268                sb.delete(charStart, charEnd + 1);
269                sb.insert(charStart, c);
270        charStart = sb.indexOf("&#");
271            } else {
272                break;
273            }
274        }
```

In this block of code the method finds the position of the first '\$, eventually escaped, in the given string.

```
277            // Find the next $
278            for (; i < sb.length(); i++) {
279              if (sb.charAt(i) == '$') {
280                    i++;
281                    break;
282                }
283            }
284            if (i == sb.length()) break;
285            // Check to see if the $ is escaped
286        if (i > 1 && sb.charAt(i - 2) == '\\') {
287        sb.deleteCharAt(i - 2);
288                i--;
289                continue;
290            }
```

In this block of code the method identifies the name of the SSI variable processed, considering also the possibility that the variable name could be wrapped in curly braces.

```
291            int nameStart = i;
292            int start = i - 1;
293            int end = -1;
294            int nameEnd = -1;
295            char endChar = ' ';
296            // Check for {} wrapped var
297        if (sb.charAt(i) == '{') {
298                nameStart++;
299                endChar = '}';
300            }
301            // Find the end of the var reference
302            for (; i < sb.length(); i++) {
303                if (sb.charAt(i) == endChar) break;
304            }
305            end = i;
```

```
306              nameEnd = end;
307              if (endChar == '}') end++;
```

In this block of code the method replaces the name of the SSI variable processed with its actual value using the substituteVariables() method.

```
308              // We should now have enough to extract the var name
309              String varName = sb.substring(nameStart, nameEnd);
310              String value = getVariableValue(varName);
311              if (value == null) value = "";
312              // Replace the var name with its value
313              sb.replace(start, end, value);
314              // Start searching for the next $ after the value
315              // that was just substituted.
316              i = start + value.length();
317          }
318      return sb.toString();
```

# 4  Issues

In this section is included a list of problems found during the inspection of the assigned code.

## 4.1  SSIServlet.java

**General Considerations**
In general the class lacks of documentation: comments and Javadoc are not complete and where inserted are sometimes meaningless and very short.
All the member(internal) variables are declared as 'protected', it is always preferable to use the 'private' access modifier(see checklist 28).

- *init()*
    - Checklist[11]: All the if statements present in the body of this method do not use curly braces
    - Checklist[23]: The javadoc written for this method is not sufficient to explain its role and its behavior in the context of the SSIServlet class
    - Checklist[40]: All the comparisons present in the body of this method use improper operators, in fact the elements in comparison are always objects(strings in particular)
    - Checklist[18]: None of the instructions present in the body of this method is commented. This may be correct if all these instructions are self explicative, but at least the last if statement needs comments to explain what it tries to achieve
    - Checklist[14]: The following lines exceed the length of 80 characters

```
113              debug = Integer.parseInt(getServletConfig().
                   getInitParameter("debug"));


116              Boolean.parseBoolean(getServletConfig().
                   getInitParameter("isVirtualWebappRelative"));


119              expires = Long.valueOf(getServletConfig().
                   getInitParameter("expires"));


121          buffered = Boolean.parseBoolean(getServletConfig().
                 getInitParameter("buffered"));


126              outputEncoding = getServletConfig().getInitParameter
                   ("outputEncoding");
```

– Checklist[8,9]: The indentation of lines is made using tabs and not spaces

– Checklist[52,53]: In the following lines the Long.valueOf method may throw a NumberFormatException which is neither managed nor rethrown

```
119              expires = Long.valueOf(getServletConfig().
                   getInitParameter("expires"));
```

– Checklist[52,53]: In the following lines the Integer.parseInt method may throw a NumberFormatException which is neither managed nor rethrown.

```
113              debug = Integer.parseInt(getServletConfig().
                   getInitParameter("debug"));
```

• *requestHandler()*

– Checklist[8,9]: All indentations in the class are made using tabs and not spaces

– Checklist[11]: The following conditional block uses no enclosing braces

```
185          if (debug > 0)
186              log("SSIServlet.requestHandler()\n" + "Serving "
187                  + (buffered?"buffered ":"unbuffered ") + "
                       resource '"
188                  + path + "'");
```

– Checklist[18]: No comments from line 210 to the end of the function

– Checklist[29,33]: The following declarations of variables should be placed in top of the body of the method.

```
197          URL resource = servletContext.getResource(path);


203          String resourceMimeType = servletContext.getMimeType(
                 path);
```

– Checklist[40]: The comparisons present in the following lines use improper operators, in fact the elements in comparison are object

```
191            if (path == null || path.toUpperCase(Locale.ENGLISH).
                   startsWith("/WEB-INF")
```

```
198            if (resource == null) {
```

```
204            if (resourceMimeType == null) {
```

```
208            if (expires != null) {
```

- Checklist[52,53]: In the following line the method getResource may throw a 'MalformedURLException', which is neither managed nor rethrown

```
197            URL resource = servletContext.getResource(path);
```

- Checklist[15]: Wrong line breaking in the following line

```
191            if (path == null || path.toUpperCase(Locale.ENGLISH).
                   startsWith("/WEB-INF")
192                || path.toUpperCase(Locale.ENGLISH).startsWith("
                      /META-INF")) {
```

- **processSSI()**

  - Checklist[23]: No Javadoc has been written for this method
  - Checklist[28]: The method has the protected access modifier but it is never invoked from its subclasses.
    It should be declared as 'private'
  - Checklist[18]: None of the instructions present in the body of this method is commented. This may be correct if all these instructions are self explicative, but most of the instructions present in this method are not self explicative
  - Checklist[40]: All the comparisons present in the body of this method use improper operators, in fact the elements in comparison are always objects(strings in particular)
  - Checklist[29,33]: In these lines local variables are defined and assigned to a value. Since these assigments and definitions do not depend from the result of previous instructions, they must be put in the top of the body of the method

```
233            URLConnection resourceInfo = resource.openConnection();
234            InputStream resourceInputStream = resourceInfo.
                   getInputStream();
235            String encoding = resourceInfo.getContentEncoding();
```

```
239            InputStreamReader isr;
```

- Checklist[1]: In this line a local variable of the type "InputStreamReader" is defined. The name of this variable is "isr" which does not convey any immediate meaning about the role and the use of this variable

```
239            InputStreamReader isr;
```

11

- Checklist[52,53]: The method openConnection throws an IOException that is not managed
The method getInputStream throws an IOException that is not managed

```
233          URLConnection resourceInfo = resource.openConnection();
234          InputStream resourceInputStream = resourceInfo.
                 getInputStream();
```

The getWriter method on the HttpServletResponse object throws a IOException, a IllegalStateException, UnsupportedEncodingException which are neither managed nor rethrown

```
230              printWriter = res.getWriter();
```

```
255              res.getWriter().write(text);
```

- Checklist[58]: The close method should be invoked on the bufferedReader variable and on the isr variable, since these variables are not used anymore in the rest of the method and they represent streams of bytes readers

```
247          long lastModified = ssiProcessor.process(bufferedReader,
248              resourceInfo.getLastModified(), printWriter);
```

## 4.2 SSIMediator.java

**General Considerations**
In general the class lacks of documentation: the Javadoc is not complete and many instructions blocks are left with no comments at all.
All the member(internal) variables are declared as 'protected', it is always preferable to use the 'private' access modifier(see checklist 28).

- *substituteVariables()*

    - Checklist[8,9]: All indentations in the class are made using tabs and not spaces
    - Checklist[23]: The Javadoc provided for the method is not complete
    - Checklist[11]: The following conditional blocks use no enclosing braces

```
253      if (val.indexOf('$') < 0 && val.indexOf('&') < 0) return
             val;
```

```
303                  if (sb.charAt(i) == endChar) break;
```

```
307              if (endChar == '}') end++;
```

```
311              if (value == null) value = "";
```

– Checklist[1]: The only parameter of the method is named 'val' which which does not convey any immediate meaning about its role and use

– Checklist[15]: Wrong line breaking in the following line

```
266                 char c = (char) Integer.parseInt(
267                     sb.substring(charStart + 2, charEnd));
```

– Checklist[52,53]: In these lines the method indexOf may throw a NullPointerException, which is neither managed nor rethrown

```
262        int charStart = sb.indexOf("&#");
```

```
264        int charEnd = sb.indexOf(";", charStart);
```

```
270        charStart = sb.indexOf("&#");
```

– Checklist[40]: In the following line improper comparison operators are used, in fact the elements in comparison are object

```
311            if (value == null) value = "";
```

– Checklist[29,33]: The variables in following lines should be placed in the top of the body of the method

```
309            String varName = sb.substring(nameStart, nameEnd);
310            String value = getVariableValue(varName);
```

# 5 Additional Issues

In this section are included additional problems and issues not present in the checklist

## 5.1 SSIServlet.java

- *init()*

    - In these lines we have the assignment of properties of the class, and this assignment does not depend from the result of previous instructions. Given that, these instructions should be put in the top of the body of the method

```
115          isVirtualWebappRelative =
116              Boolean.parseBoolean(getServletConfig().
                     getInitParameter("isVirtualWebappRelative"));

121          buffered = Boolean.parseBoolean(getServletConfig().
                 getInitParameter("buffered"));

123          inputEncoding = getServletConfig().getInitParameter("
                 inputEncoding");
```

    - In the body of this method continuous calls to the methods of the object returned by the getServletConfig() method are performed. This is inefficient since the above mentioned object can be stored in a local variable and so made accessible without method calls

    - Methods and properties of the superclass of a class must be referenced by that class using the "super." prefix. This should be done for reasons of clarity and readability, so that the developer can immediately distinguish the manipulation of methods and properties of the superclass of the current class from the manipulation of those which belong to the current class.
      This behavior is not followed in this method(all lines)

- *requestHandler()*

    - Methods and properties of the superclass of a class must be referenced by that class using the "super" prefix. This should be done for reasons of clarity and readability, so that the developer can immediately distinguish the manipulation of methods and properties of the superclass of the current class from the manipulation of those which belong to the current class.
      This behavior is not followed in lines:

```
183          ServletContext servletContext = getServletContext();
```

      And in all lines that present the invocation of 'log()'

    - Methods and properties of the current class must be referenced within the class using the "this." prefix. This should be done for reasons of clarity and readability, so that the developer can immediately distinguish the manipulation of methods and properties of the current class from the manipulation of

those which belong to the superclass of the current class.
This is also useful for distinguish the manipulation of properties of the current class and local variables. This behavior is not followed in lines:

```
185          if (debug > 0)
186              log("SSIServlet.requestHandler()\n" + "Serving "
187                      + (buffered?"buffered ":"unbuffered ") + "
                            resource '"
188                      + path + "'");


207          res.setContentType(resourceMimeType + ";charset=" +
                 outputEncoding);
208          if (expires != null) {
```

  – In these lines the ternary operator ? is used. The expression is syntactically valid but the use of ? makes it counter intuitive and less readable.
  It is preferable to use a classic if-else block.

```
186              log("SSIServlet.requestHandler()\n" + "Serving "
187                      + (buffered?"buffered ":"unbuffered ") + "
                            resource '"
188                      + path + "'");
```

- **processSSI()**

  – Methods and properties of the superclass of a class must be referenced by that class using the "super" prefix. This should be done for reasons of clarity and readability, so that the developer can immediately distinguish the manipulation of methods and properties of the superclass of the current class from the manipulation of those which belong to the current class.
  This behavior is not followed in this method(all lines).

  – Methods and properties of the current class must be referenced within the class using the "this." prefix. This should be done for reasons of clarity and readability, so that the developer can immediately distinguish the manipulation of methods and properties of the current class from the manipulation of those which belong to the superclass of the current class.
  This is also useful for distinguish the manipulation of properties of the current class and local variables. This behavior is not followed in lines:

```
220              new SSIServletExternalResolver(getServletContext(),
                 req, res,
221                  isVirtualWebappRelative, debug,
                        inputEncoding);
222          SSIProcessor ssiProcessor = new SSIProcessor(
                 ssiExternalResolver)
223                  debug);


226          if (buffered) {


252          if (buffered) {
```

15

- In the following two lines of code each statement declares a variable to hold an object and then assigns to it a 'null' value. In general assign a 'null' value to a fresh declared variable used to hold an object, is an useless operation since this is the default behavior of Java.

```
224        PrintWriter printWriter = null;
225        StringWriter stringWriter = null;
```

- In the following block of code the first if statement is redundant, it should be deleted and its contents copied into the body of the second if statement.

```
236        if (encoding == null) {
237            encoding = inputEncoding;
238        }
239        InputStreamReader isr;
240        if (encoding == null) {
241            isr = new InputStreamReader(resourceInputStream);
242        } else {
243            isr = new InputStreamReader(resourceInputStream,
                    encoding);
244        }
```

- [**CRITICAL**]In these following lines:

```
233        URLConnection resourceInfo = resource.openConnection();
234        InputStream resourceInputStream = resourceInfo.
                getInputStream();
```

The getInputStream and getContentEncoding methods cannot be called on a "URLConnection" object before opening an actual connection to the resource referred by the object itself (to solve this problem the "connect" method must be called on the "URLConnection" object before calling getInputStream, see the javadoc of the URL class)

## 5.2    SSIMediator.java

No particular additional issues has been found in the 'substituteVariables()' method.

# 6  Appendix

## 6.1  References

- javaCheckList.pdf: Contains the check list used to inspect the code present in this document.

## 6.2  Tools Used

- Atom/ LaTeX: To redact this document

- Eclipse: To simulate the behavior of the assigned code

## 6.3  Hours of Work

- Andrea Sessa: 14.5 hours

- Giorgio Pea: 13 hours