Index

1. Introduction

Scriba is a tool to generate documentation from source code. The content of the documentation is extracted directly from the source code and/or any other files found on the project tree. Unlike many other tools Scriba does not generate API-like documentation neither the generated documentation necessarily follows the same structure as the source code.

The purpose of Scriba is to provide an easy way to have source code and documentation mixed on the same set of files. The problem Scriba wants to solve is the mismatch between code and documentation that tends to occur on the life-cycle of a project, specially as the project grows older. What usually happens is that developers are busy producing code that works, and as documents are kept on completely different folder structures they never go and keep them up to date. Hopefully this script will be of great help for developers, specially on Agile and Extreme Programming environments where for their nature changes on the code happen a lot more often.

The main goal of Scriba is that we do not hinder the normal development work in any way:

- Scriba should not to give any extra work to the developers to create the documentation.
- Scriba should not interfere with any other software documentation tools.
- Scriba should not pollute the source code with strange syntax or extra sections.
- Scriba documentation should be able to be used as code comments and the code comments should also be able to be used as Scriba documentation.

With those principles in mind we made Scriba use the normal code comments sections to take the contents of the documents it will generate. We also have given the ability to format the text in a way that can easily be read as a normal text file but that will render nicely when exporting as a proper document.

2. How Does it Work?

Scriba creates a list of text files to process using the Linux *find* command on the directory specified by the *-i* command line argument. We can also pass extra parameters to the find command with the *-ifind* parameter. This can be useful in case we need to exclude certain directories from the source code processing. For example, to exclude all the files from the "ckeditor" folder we can do:

```
> scriba -o doc -ifind '! -path "*/ckeditor/*"'
```

Each file is processed to look for Scriba encoded information. The order in which the files are processed is not important as Scriba generates the documents according to the section names and numbers as defined in the contents of the files and that is completely independent on the file they are defined into.
Scriba will read the files and each time it detects that we are inside a comments section the software will check if that comment section contains any of the Scriba tags. If a Scriba tag is found it means that the given comment is going to be part of the generated documentation contents. Any content after the initial Scriba tag and until the end of the comment section will be put on the generated document. All that information is kept in memory so we can define and/or add information on the same generated document from completely different source files in completely different directories.

Once all the documents have been processed the software proceeds to create all defined documents using the given templates and styles.

2.1. Command Line Usage

Srciba runs on the command line on Linux based system. It is a Perl script that uses some standard Linux commands to perform some of the actions. For example we use the "find" command to be able to find and filter the files we will process for documentation contents.

The script command line accepts the following parameters:

```
scriba [-i <path>] [-ifind <find command parameters>] [-o <output
path>] [-t <template name>] [-jsdoc <jsdoc URL>] [-pdf] [-info] [-
infoinline]

    • -i <path>: path to the input directory Scriba will process
      in order to extract all the document creation section.
      Default: "./"
    • -ifind <find command parameters>: parameters to pass to find
      in order to give better control of which files Scriba will
      use when looking into files in order to find the document
      data.
      Default: ""
    • -o <output path>: the output directory path where we will
      create the documents stored in the given source code.
      Default: "./ScribaGen/"
```

```
    - -t <template name>: the name of the templates we will use to
      render the generated documents, the software is distributed
      with two templates "default" and "print".
      Default: "default"
    - -jsdoc <jsdoc URL>: tells Scriba to parse JSDoc information
      and to generate links whenever a module or function is
      referenced on the Scriba comments. The modules are
      referenced simply by stating their names as they appear on
      the JSDoc definition. For functions you need to write them
      in the form of Module.Function.
      Default: no JSDoc processing.
    - -pdf: Tells Scriba to generate a PDF file on top of the
      default HTML generated using an external tool.
      Default: no PDF generated
    - -info: Tells Scriba to print on the console a summary of
      lines of code, comments and Scriba comments.
      Default: no info printed
    - -infoinline: Tells Scriba to add to the generated document
      embedded inline information on what sections come from what
      files. It is useful for debug purposes.
      Default: no inline debug information on output
```

2.2. Scriba Sections and Functionality

At the moment we have done extensive tests both in Perl and JavaScript. In theory it should also work on C/C++/Java/... but more languages are going to be added in the future.

2.2.1. Document Organization

All the Scriba sections are comments which their initial characters are **scrb:** The content of the section will be stored to be part of the documentation. The section ends when the comment section ends.
As we are simply putting the content from the source code into the generated HTML document we can use any HTML tag to format the output. But as reading the source from the text file could become complicated Scriba provides some easy to read short-codes that will transform into the proper HTML tag without loosing clarity on the plain text.

```
// scrb:
// This section will be part of the documentation.
```

If we find a Scriba section marker followed by a number **scrb: X** it means that the Scriba section does not end at the end of the comment but after X lines of comments that are empty.
This feature is particularly useful when you want to integrate and reuse comments from other tools or purposes to be included as part of the generated documentation.

```
// scrb: 2
// This section will be part of the documentation.
// This section will also be part of the documentation.
//
// This section will also be part of the documentation.
//
// This section will NOT be part of the documentation
// as it is part of some other documentation system like
// JSDoc information.
```

Perhaps the most important feature is the way Scriba organizes documents and chapters. The way it works is that the Scriba software starts processing a file for Scriba document sections. There are two special section tags that tell to what **Document** and **Chapter** the next Scriba comments belong to. This is valid until the next time another document or chapter section occur.
This means that once a document and chapter have been defined the next Scriba comments will all go into that specific chapter.
For example this text is a good few comments below the following definition:

```
// scrb: Document: UserGuide
// scrb: Chapter: 2.2. Scriba Sections and Functionality
```

Note that the name of the *document* is going to be used as the file name for the document.
Note that the chapter numbering has an implication on the type of HTML *heading* tag Scriba will use and the way the document index will look like. The numbering format Scriba understands follows the following pattern:

```
<0-9>+.[<0-9>+.[<0-9>+.[etc]]]
```

This means that X. AAAA will be an H1 type of heading.
This means that X.Y. AAAA will be an H2 type of heading.
This means that X.Y.Z. AAAA will be an H3 type of heading.

2.2.2. Document Formatting

If inside a Scriba section we find a line (or a series of lines) starting with an **asterisk** Scriba will transform them into an unordered list section.

```
// * Item 1
// * Item 2
// * Item 3
// * Item 4
```

Will be rendered the following way:

- Item 1
- Item 2
- Item 3
- Item 4

If inside a Scriba section we find a line (or a series of lines) starting with a **hash** Scriba will transform them into an ordered list section.

```
// # Item 1
// # Item 2
// # Item 3
// # Item 4
```

Will be rendered the following way:

1. Item 1
2. Item 2
3. Item 3
4. Item 4

If inside a Scriba section we find a comment line that contains only an **underscore** this will tell Scriba to insert a *paragraph break* at that point.
If we find the **underscore** at the end of a line this will tell Scriba to insert a *line break* at that point.

```
// AAAA content, AAAA content, AAAA content, AAAA content
// AAAA content, AAAA content, AAAA content, AAAA content
// _
// BBBB content, BBBB content, BBBB content, BBBB content_
// BBBB content, BBBB content, BBBB content, BBBB content
```

Will be rendered the following way:

AAAA content, AAAA content, AAAA content, AAAA content AAAA content, AAAA content, AAAA content, AAAA content

BBBB content, BBBB content, BBBB content, BBBB content
BBBB content, BBBB content, BBBB content, BBBB content

We can apply some basic formats to the text we are typing just by enclosing between special characters.

```
// asterisks for bold: *AAAA AAAA* content
// underscores for italic: _BBBB BBBB_ content
// pluses for underline: +CCCC CCCC+ content
// minuses for strikeout: -DDDD DDDD- content
```

Will be rendered the following way:

asterisks for bold: content **AAAA AAAA** content
underscores for italic: *BBBB BBBB* content
pluses for underline: CCCC CCCC content
minuses for strikeout: ~~DDDD DDDD~~ content

As the greater and less than symbols have special meaning in HTML we are going to need to escape them. If you want to use a < or a > you need to precede them with one backslash: \< and or \>. You can also use the HTML entity codes: < and >

```
// \<
// \>
```

Will be rendered the following way:

```
<
>
```

As the ampersand has special meaning when coding HTML entities we are going to need to escape them if we want to use them. If you want to use a & you need to precede it with one backslash: \& You can always use the HTML entity: &

```
// \&
```

Will be rendered the following way:

```
&
```

Scriba also provides the ability to escape any character in case it is useful:

```
// \\
// \A
```

Will be rendered the following way:

```
\
A
```

We can also create a special Scriba section that will contain code taken directly from the application source. We mark the place where we want to start dumping code with the special key "scrb: code begin" and it will dump everything until we find "scrb: code end".

```
// scrb: code begin
real application code
// scrb: code end
```

This will be rendered the following way:

```
if($bInCommentsCode) {
    if($sCleanLine =~ /^\s*$refhParameters->{Scriba_ID}\s*code\s*end/) {
        $bInCommentsCode = 0;
        # replacing some special characters
        $sCleanCode =~ s/</</g;
        $sCleanCode =~ s/>/>/g;
        $sCleanCode =~ s/\n/<br \/>/g;
        # adding the code within code sections
        $refhDocStructure->{$sDocument}{$sChapter}{CONTENT} .= '<pre class="code source_code">';
        $refhDocStructure->{$sDocument}{$sChapter}{CONTENT} .= $sCleanCode;
        $refhDocStructure->{$sDocument}{$sChapter}{CONTENT} .= '</pre>';
    }
    else {
        # removing as many initial blank spaces as the first
        # initial line has
        if($sCleanCode eq "") {
            if($sLine =~ /^(\s+)/) {
                $sCleanCodeSpaces = $1;
            }
        }
        $sLine =~ s/^$sCleanCodeSpaces//;
        $sCleanCode .= $sLine . "\n";
    }
}
```

2.2.3. Linking with JSDoc

One of the features we can activate is the linking of the Scriba generated documents with JSDoc. What will happen is that is we process some source code files that have properly encoded JSDoc tags Scriba is able to interpret them and then it can create links to the right JSDoc documentation.
You can activate this option with the *-jsdoc* command line option where you can pass the URL to use as base direction for the links pointing to the JSDoc HTML page.
Scriba will parse the content of each comment section looking for matching class names and

functions and if found it will blindly create the link (meaning that Scriba will not check neither warn if the target URL is wrong or does not exist).
The way to reference modules and functions to point to JSDoc documentation is the same used in JSDoc:

```
<Module>[(.#)<Method>]
```

2.3. Statistical Information

As Scriba processes all files on your project directory looking for comments and its contents, it can provide some information about the number of lines and number of comments found that might be of help.
If you run the script with the *-info* switch at the end of the run it will print on the console something similar to this:

```
Building the documentation into "../doc" path
Creating '../doc/DeveloperGuide.html'
Creating '../doc/UserGuide.html'

lines: 11 comments: 6 / 54.55% Scriba comments: 0 / 0.00%
./build_documentation.sh
lines: 1100 comments: 596 / 54.18% Scriba comments: 436 / 73.15%
./scriba.pl

TOTAL FILES: 2
lines: 1111 comments: 602 / 54.19% Scriba comments: 436 / 72.43%
```

This report tells you, first for each file and then as a total:

- The number of *lines* on the files (not lines of code, simply lines)
- The number of comment lines we have detected. And a percentage based on the above number of lines.
- The number of Scriba comment lines. And a percentage based on the above number of comments.