JS-Doc-Toolkit RST-Template Documentation

Release 0.1

Juha Mustonen

April 21, 2016

Contents

1 Installation **3**

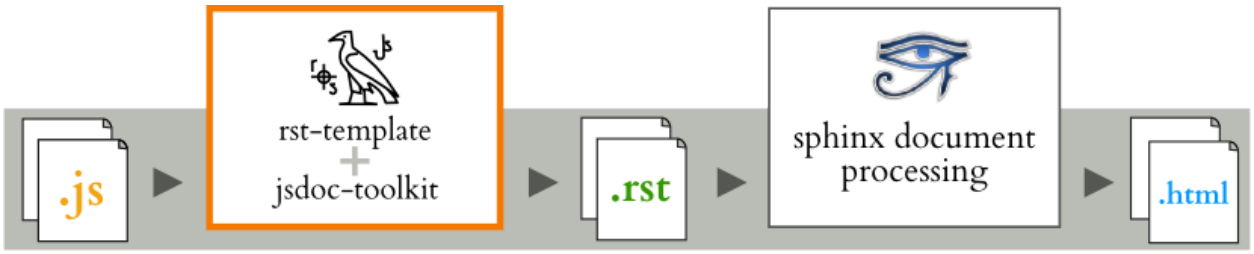
2 Usage **5**
2.1 Comment your code 5
2.2 Generate API documents 5
2.3 Write documentation 6

3 Examples **7**
3.1 API Reference 7

4 License: MIT **11**
4.1 Development 11

Project provides a RST (ReStructuredText) compliant templates for JsDoc Toolkit documentation generator, making it easy to document project's JavaScript resources with Sphinx.

With the custom rst -template, it is possible to generate rst-document pages using JsDoc Toolkit. From there, the generated pages can be included as part of documentation, similar to Sphinx autogen feature. The process is shown in the illustration:



CHAPTER 1

Installation

Suggested installation steps (unless you already have some of the apps/modules installed)

1. Install Java and Ant:
   - Download Java and extract it for example to */opt/java/*
   - Download Ant and extract it for example to */opt/apache-ant/*
2. Install JsDoc Toolkit:
   - Download release: http://jsdoc-toolkit.googlecode.com/files/jsdoc_toolkit-2.4.0.zip
   - Extract package for example to: `/opt/jsdoc-toolkit/`
3. Setup JsDoc Toolkit RST Template
   - Download the package (not yet available) or checkout the sources:
   - Open `build.properties` for editing and set the directories:
     - Where to find JsDoc Toolkit
     - Where to read javascript sources
     - Where to generate rst documents

```
# Directory where the jsdoc-toolkit is installed
jsdoc-toolkit.dir=/opt/jsdoc-toolkit

# Directory where to find javascript sources
js.src.dir=src/js

# Directory where to generate rst files
js.rst.dir=dist/rst
```

Tip: You can also pass these values as a parameter to Ant:

```
ant -Djs.src.dir=somewhere/else build
```

4. Test the setup to ensure everything works:
   - List Ant tasks:

```
ant -p
```

   - Try out the js conversion by building the .js -sources:

```
ant build
```

- See the outcode directory defined by `js.rst.dir`

Note: This document only describes how to generate RST-files from JavaScript -sources. Please follow the Sphinx -official documentation for how to write and configure documentation.

CHAPTER 2

Usage

As it can be seen from the illustration, generating the JavaScript -source API into Sphinx powered documentation, it requires a tool chain:

- *Comment your code*
- *Generate API documents*
- *Write documentation*

2.1 Comment your code

Javascript source code, commented using JsDoc Toolkit commenting conventions.

```
/**
* @class
* Comment block
*
* @param {string} name Unique name for the app
*/
var App = function (name) {
  // @default "Anonymous"
  this.name = name || 'Anonymous';
};

/**
 * Runs the app
 * @returns {App} itself bac
 */
App.run = function() {
  return this;
}
```

2.2 Generate API documents

Build RST documents from source code with either directly with JSDoc Toolkit or Ant script:

```
ant -Djs.src.dir=src/myapp -Djs.rst.dir=doc/api/myapp  build
```

After generating the source code with custom template, the outcome is something like (in this case, the file name is `api/myapp/symbols/App.rst`):

```
.. js:class:: App (name)

    Comment block

    :param string name:

       Unique name for the app

    .. js:function:: App.run ()

       Runs the app
```

Note: You may edit the generated RST API documents, if you like. However, that prevents you re-generating the documents from the sources again (unless you are willing to do some manual merging). Which approach you should use, depends on needed documentation.

2.3 Write documentation

Now, both generated and manually written documentation can be used together. The documentation structure may be as follows:

```
conf.py
index.rst
api/
  index.rst
  symbols/
    _global_.rst
    App.rst
```

To include API documents in Sphinx document tree, the suggested method is to set `api/index` -toctree somewhere in master document:

```
.. toctree::

   api/index
```

Then, you may refer documented classes, functions and other JavaScript elements by using Sphinx notation:

```
The application is implemented in class :js:class:`App`,
where as the actual processing is done in :js:func:`App.run`.
To see the complete API, see :ref:`separate API document <api>`.
```

CHAPTER 3

Examples

Following documents are generated using the RST template:

3.1 API Reference

generator JsDoc Toolkit

3.1.1 Android (class)

class **Android**()

Android is a subclass of Bot

Android.**talk**()

Makes the Android talk

Returns what Android said, in format: "Android said: <message>"

Return type String

See Bot#talk

#talk

Android.**walk**(*direction*)

Arguments

- **direction** (Direction) – X and Y coordiates

This function really does the thing.

```
var bot = new Bot('bot');
bot.walk({x:123, y:2});
```

Documentation generated by jsdoc-toolkit 2.4.0 using jsdoc-toolkit-rst-template

3.1.2 App (class)

class **App**(*name*)

Constructs the Bot

Arguments

- **name** (*string*) – Unique name for the bot

Documentation generated by jsdoc-toolkit 2.4.0 using jsdoc-toolkit-rst-template

3.1.3 Bot (class)

class Bot (*name*)

This is a class with no other purpose but being an example and for testing.

Second paragraph just to show off:

•One

•Two

As for the HTML elements, template has a limited support for transforming them into **bold** and *italic* elements.

Arguments

- **name** (*string*) – Unique name for the bot

Author Juha Mustonen

```
var android = new Bot("Android");
android.speak();
```

```
// Extend bot
subapp = new Bot;
subapp.prototype.walk = function(direction) {
  // ...
};
```

Bot.**talk**()

This function really does the thing.

```
var bot = new Bot('bot');
bot.talk({x:123, y:2});
```

Bot.**walk**(*direction*)

Arguments

- **direction** (Direction) – X and Y coordiates

This function really does the thing.

```
var bot = new Bot('bot');
bot.walk({x:123, y:2});
```

Documentation generated by jsdoc-toolkit 2.4.0 using jsdoc-toolkit-rst-template

3.1.4 Direction (class)

Simple class for modelling a direction

class Direction (*x*, *y*)

Arguments

- **x** (*int*) – X coordinate
- **y** (*int*) – Y coordinate

Deprecated Since version 0.5. You should now plain Object instead: `{x:234, y:123}`

`Direction.x`

Type number

X value

`Direction.y`

Type number

Y value

Documentation generated by jsdoc-toolkit 2.4.0 using jsdoc-toolkit-rst-template

3.1.5 Hangar (data)

Hangar

Hangar

Hangar is a singleton object that is responsible for fixing the bots

```
var mybot = Bot("test");
mybot = Hangar.repair(mybot);
```

See Bot

`Hangar.repair(bot)`

Arguments

- **bot** (Bot) – Bot to repair

Returns Fixed bot back

Return type Bot

Documentation generated by jsdoc-toolkit 2.4.0 using jsdoc-toolkit-rst-template

3.1.6 _global_ (data)

global

`_global_.ACCEL`

Accelerator value: 9.80665

setup (*hangar*)

Arguments

- **hangar** (Hangar) –

Set ups the hangar

Documentation generated by jsdoc-toolkit 2.4.0 using jsdoc-toolkit-rst-template

Documentation generated by jsdoc-toolkit 2.4.0 using jsdoc-toolkit-rst-template

CHAPTER 4

License: MIT

This piece of software is MIT licensed. It means you can freely take it, *hack it* and break it - both in personal and commercial use.

This also means *your contribution is welcome*

4.1 Development

You're contribution is welcome. If you improve the RST templates, please contribute it back to project.

- *Templates*
- *Testing*

4.1.1 Templates

TBD

4.1.2 Testing

TBD

Symbols

global (global variable or constant), 9
global.ACCEL (_global_ attribute), 9

A

Android() (class), 7
Android.talk() (Android method), 7
Android.walk() (Android method), 7
App() (class), 7

B

Bot() (class), 8
Bot.talk() (Bot method), 8
Bot.walk() (Bot method), 8

D

Direction() (class), 8
Direction.x (Direction attribute), 9
Direction.y (Direction attribute), 9

H

Hangar (global variable or constant), 9
Hangar.repair() (Hangar method), 9

S

setup() (built-in function), 9