Folder Structure & Content of GPars Documentation Website

The Whole GPars Team <gpars-developers@googlegroups.com>

Version 1.2.1, 2015-12-21

Table of Contents

Documentation

Goals

- To convert original **GPars** documentation from `mark-down` syntax to **Asciidoctor**
- To create a new look
- To combine a variety of **GPars** information into a single place (guides,references,tutorials.etc.)
- To re-arrange file structures as found in gpars.github.io and dis-continue `Jekyll`
- To maintain the static nature of the website; only a few dynamic parts were needed, but more later as features grow

For our documents, we use the **Asciidoctor** tool and **Gradle** plugin for **Asciidoctor**

For hosting, this implementation uses a CloudFoundry PaaS provider like IBM BlueMix, Pivotal, CloudBees, Anynines.com

Introduction

A knowledge of **Gradle**, **GitHub**, and both static and dynamic **web servlets** is assumed as are a familiarity with website folder structure, html, etc.

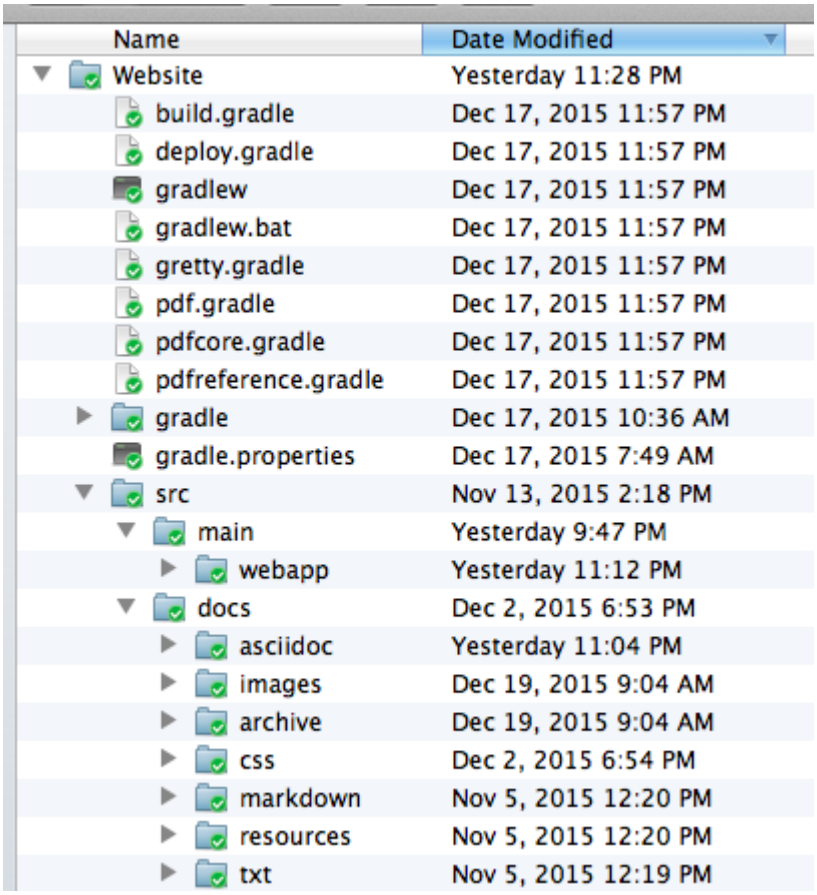
The groovy-based build tool called **Gradle** is used to construct this website. Several different build scripts are used to implement this website. Each script is plain text UTF-8 encoded. Each script serves a different purpose.

Original **GPars** documents were stored as text files using the `mark-down` syntax. These files have been translated into a more robust **Asciidoctor** syntax. Each text file is plain text UTF-8 encoded and resides in the `./Website/src/docs/asciidoc` folder plus sub-folders for each topic.

All translated files retain the original mark-down filename but now sport an **.adoc** suffix.

Gradle Build Tool Overview

Gradle Components



This build tool allows us to include a *wrapper* holding components used by **Gradle**.

After cloning this repository, these components are all you need to build and manage this app, even if **Gradle** is not installed on your system. This avoids installing **Gradle** on your system.

1. **gradlew** - a Linux/Apple bash `wrapper` script to execute the build tool
2. **gradlew.bat** - the windows equivalent of the prior script
3. **gradle/** - a folder of build tool components
4. **gradle.properties** - settings to influence the execution of this build tool

1. **build.gradle** - executes `defaultTasks` to setup the job, run the **Asciidoctor** task, then builds a servlet war file.
2. **gretty.gradle** - used for internal purposes to run this site as a local webservice; useful to review changes prior to server upload; from a command line, do this :
   ```
   cd ./Website
   ./gradlew -b gretty.gradle appRun
   ```
   then from a browser address line, use: `http://localhost:8080` - *NOTE: may not work on Java JVM9*
3. **deploy.gradle** - uploads our servlet war file to target CloudFoundry (or jetty or tomcat) service; see notes within this script for more details; *NOTE: enter credentials prior to first use*; to run this script,

try :

```
cd ./Website
./gradlew -b deploy.gradle
```

Default tasks in this script run several steps to login to the remote CloudFoundry API, and push an existing **.war** file to it. This script posts the **.war** file that was built in the **build.gradle** script from `./build/libs` folder to the target.

Root Folder Layout

Input / Output Folder Designations of Website Root and Sub-Folders - see **build.gradle**

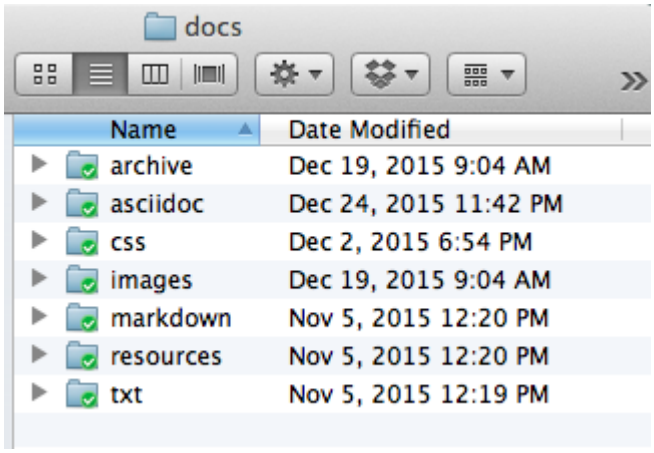
```
asciidoctor {
    sourceDir = file('src/docs/asciidoc') ①
    outputDir = file('src/main/webapp')   ②
}
```

① declares the root input folder of all *.adoc files

② declares the output folder for generated HTML documents

Input Source

For input, **Asciidoctor** reads each *.adoc file in the `./Website/src/docs/asciidoc` folder and any sub-folders.



Input Sub-Folder Layout

Topical areas of discussion are often broken into pieces, typically segregated physically as folders-within-folders (i.e. sub-folders).

Our website follows this pattern. Here's a list of our current input sub-folders and their purpose as of

Dec.2015. Note others maybe added later.

1. **archive** - original stuff no longer needed but saved 'just-in-case'
2. **asciidoc** - contains the converted markdown documents as **.adoc** files plus sub-folders by topic
3. **css** - styling components;
4. **images** - variety of stuff, some current, some obsolete and due for a tidy-up
5. **markdown** - original markdown files saved here from the github repo - keep or delete ? No harm for the moment
6. **resources** - to be used for zips of older releases
7. **txt** - by-products of search for stale/obsolete URLs

HTML Generation Process

HTML generation is provided by a single task within the **build.gradle** script. The **Asciidoctor** task has this privilege.

All document files have their original *mark-down* filename plus an **.adoc** suffix.

The **Asciidoctor** plugin in **build.gradle** reads each **.adoc** file from the `./Website/src/docs/asciidoc` folder. It renders an **.html** equivalent and writes this to the `./Website/src/main/webapp` folder, or sub-folder.

Folder structure is preserved from the source folder.

Root HTML Generation Commands

- Manual - when used manually from a command line to run a single task in **build.gradle**, do this :
  ```
  cd ./Website
  ./gradlew asciidoctor
  ```
- Automatic - runs tasks in this script declared as default tasks. These are `'clean','asciidoctor', 'build', 'war'`.
  Do this without any task names after gradlew:
  ```
  cd ./Website
  ./gradlew
  ```

PDF Generation Scripts

PDF generation is provided by identical gradle scripts. The only difference between them is the declarations for input and output folders. These appear around lines 50-55 in each script. They look like the following logic that produces a PDF document in our **core** folder for each `*.adoc` file :

Sample Input / Output Folder Designations for **Core** *PDFs*

```
sourceDir = file('src/docs/asciidoc/core')  ①
outputDir = file('src/main/webapp/core')    ②
```

① declares the input folder of *.adoc files

② declares the output folder for the generated PDF

1. **pdf.gradle** - generates a single page PDF for the full user guide as defined in `index.adoc`; this index uses the `include` syntax to insert pieces of text from the *.txt files.
2. **pdfcore.gradle** - generates a single page PDF for each ./core topic file ending with `.adoc`
3. **pdfreference.gradle** - generates a single page PDF for the reference manual defined in `index.adoc`; this index uses the `include` syntax to insert pieces of text from the *.txt files.
4. **pdfstructure.gradle** - generates a single page PDF describing the workings of this website, it's folders, scripts,etc.

Output Generation

The `./Website/src/main/webapp` folder is used as the output capture folder for our website. **Asciidoctor** writes **.html** (or .PDF) output here. Sub-folders are duplicated, if necessary, to preserve the integrity of the source.

Output Sub-Folder Layout

Topical areas of discussion are often broken into pieces, typically segregated physically as folders-within-folders (i.e. sub-folders).

Our website follows this pattern. Here's a list of our current output sub-folders and their purpose as of Dec.2015. Note others maybe added later.

1. **api** - groovydoc and javadoc compiler output copied over from latest release; **groovy-overview-summary.html** and **java-overview-summary.html** are re-built by build.gradle logic
2. **core** - describes the primary mechanisms of **GPars**
3. **css** - styling components; `css3menu1` sub-dir is for the site navigation bar
4. **font-awesome** - used by asciidoctor to generate admonition icons
5. **fonts** - **Asciidoctor** usage
6. **guide** - everything to construct our **.html** user guide and companion **.pdf**
7. **images** - variety of stuff, some current, some obsolete and due for a tidy-up
8. **img** - used in landing page (**index.html**)
9. **JonKerridgeBook** - chapters from his material and his PDF series
10. **js** - for landing page and JQuery support
11. **quickstart** - the short reference manual for both **.html** and it's one-page PDF version

Our landing page **index.html** is not an **Asciidoctor** artifact, and must **NOT** be lost or deleted

WEB-INF Review

WEB-INF is our deployment folder for java servlets. It holds traditional support and configuration files.

Here's a list of components, sub-folders and their purpose as of Dec.2015.

1. **groovy** - dynamic components written as groovy scripts (plus a lot of obsolete stuff)
2. **includes** - fragments of html stored as groovy templates for **include** targets
3. **lib** - jars of runtime logic to support
   1) additional servlet processing in `caelyf-1.3.3.jar` and
   2) live **Asciidoctor** translation from any *.adoc file directly to servlet response stream in `Doctor-all-1.0.jar`
4. **pages** - groovy templates (**.gtpl**) add further text to a servlet response
5. **logging.properties** - adjust log level as needed
6. **routes.groovy** - adds additional mappings to our own code for browser addresses. For example :
   `get "/datetime",forward:"/datetime.groovy", cache:2.minutes`
   where **get**, **post**, etc HTTP requests go to a specific groovy script; if **Redis** service is running, response is copied to cache for 2 min.s before reconstructing it

   In this example, a request from browser address: `http://localhost:8080/datetime` forwards to `datetime.groovy` script in `/Website/src/main/webapp/WEB-INF/groovy` folder. It gets system date then forwards request to a template at `/Website/src/main/webapp/WEB-INF/pages` for final response resolution.
7. **web.xml** - configures servlet container (jetty,tomcat,etc.) by mapping file suffixes to servlets; rarely changes.

Sample Workflow

To be added soon