project.name

Release *project.version*

February 17, 2015

The *sphinx-maven* plugin is a Maven site plugin that uses Sphinx to generate the main documentation. Sphinx itself was origially created by the Python community for the new Python documentation. It uses a plain text format called reStructured Text which it then compiles into a variety of documentation formats such as HTML, LaTeX (for PDF), epub. reStructured Text is similar to Markdown but - at least via Sphinx - has better support for multi-page documentation.

The *sphinx-maven* plugin is BSD licensed just as Sphinx itself is.

CHAPTER

ONE

CREATING THE DOCUMENTATION

First, create a folder `src/site/sphinx`. This folder will contain the reStructured Text source files plus any additional things like themes and configuration. The name of the folder can be changed via options should you want a different folder.

Next, add the documentation. The Sphinx first steps tutorial gives a good introduction into the required tasks. Basically what you need is

- A configuration file called conf.py that defines the theme and other options (such as which output formats etc.)
- The documentation files in reStructured Text format.
- Additional files such as static files (images etc.), usually in a `_static` sub directory.
- Optionally, a customized theme in a sub directory called `_theme`

For good examples of documentation, see Sphinx' examples page. The documentation for this plugin itself is based on the documentation for Werkzeug (documentation source for it can be found on Werkzeug's github page) and Celery (documentation source can be found on Celery's github page).

CHAPTER

TWO

RUNNING AS PART OF THE `SITE` LIFECYCLE

Simply add the sphinx maven plugin to your `pom.xml`:

```
<reporting>
  <plugins>
    <plugin>
      <groupId>org.apache.maven.plugins</groupId>
      <artifactId>maven-project-info-reports-plugin</artifactId>
      <version>2.4</version>
      <reportSets>
        <reportSet>
          <reports></reports>
        </reportSet>
      </reportSets>
    </plugin>
    <plugin>
      <groupId>org.tomdz.maven</groupId>
      <artifactId>sphinx-maven-plugin</artifactId>
      <version>1.0.1</version>
    </plugin>
  </plugins>
</reporting>
```

It is important that you set the `reportSet` attribute of the `project-info-reports` plugin to an empty set of `reports`. If not then the default `about` report will be generated which conflicts with the `sphinx-maven` plugin, and in effect Sphinx will not be run.

Maven 3 changes how reporting plugins are specified. A `profile` can be used to define a `pom.xml` that can be used with both Maven 2 and Maven 3:

```
<profiles>
  <profile>
    <id>maven-3</id>
    <activation>
      <file>
         <!--  This employs that the basedir expression is only recognized by Maven 3.x (see MNG-236
        <exists>${basedir}</exists>
      </file>
    </activation>
    <build>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-site-plugin</artifactId>
          <version>3.0</version>
          <configuration>
```

```
            <reportPlugins>
              <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-project-info-reports-plugin</artifactId>
                <version>2.4</version>
                <reportSets>
                  <reportSet>
                    <reports></reports>
                  </reportSet>
                </reportSets>
              </plugin>
              <plugin>
                <groupId>org.tomdz.maven</groupId>
                <artifactId>sphinx-maven-plugin</artifactId>
                <version>1.0.1</version>
              </plugin>
            </reportPlugins>
          </configuration>
        </plugin>
      </plugins>
    </build>
  </profile>
</profiles>
```

The profile will only be activated if Maven 3 is used to generate the site. For more details about Maven 3 and the site plugin see the Maven 3 site plugin wiki page and this Maven 3 site plugin howto.

Now all you need to do is to generate the documentation:

```
mvn site
```

This will generate the documentation in the *target/site* folder.

CHAPTER

THREE

RUNNING AS PART OF THE NORMAL LIFECYCLE

You can also bind the plugin to a normal lifecycle phase. This is for instance useful if you want to generate a documentation artifact and deploy it somewhere.

The plugin configuration is pretty much the same, the only difference is that you need to add an `execution` section. It might also be useful to change the `outputDirectory` to a different folder as the plugin by default puts the generated documentation into the `target/site` folder.

A sample `pom.xml` plugin section could look like this:

```
<build>
  <plugins>
    ...
    <plugin>
      <groupId>org.tomdz.maven</groupId>
      <artifactId>sphinx-maven-plugin</artifactId>
      <version>1.0.1</version>
      <configuration>
        <outputDirectory>${project.build.directory}/docs</outputDirectory>
      </configuration>
      <executions>
        <execution>
          <phase>package</phase>
          <goals>
            <goal>generate</goal>
          </goals>
        </execution>
      </executions>
    </plugin>
    ...
  </plugins>
</build>
```

CONFIGURATION

The `sphinx-maven` plugin has these configuration options:

Parameter	Description	Default value
`sourceDirectory`	The directory containing the documentation source.	`${basedir}/src/site/sphinx`
`outputDirectory`	The directory where the generated output will be placed.	`${project.reporting.outputDirectory}`
`fork`	Whether to run Sphinx in a forked JVM instance.	`false`
`jvm`	The JVM binary to use. If not set, then will use the one used to run the plugin.	
`argLine`	Additional arguments for the forked JVM instance (such as memory options).	
`forkTimeoutSec`	How long the plugin should wait for the plugin. 0 means wait forever.	`0`
`outputName`	The base name used to create the report's output file(s).	`index`
`name`	The name of the report.	`Documentation via sphinx`
`description`	The description of the report.	`Documentation via sphinx`
`builder`	The builder to use. See the Sphinx commandline documentation for a list of possible builders.	`html`
`verbose`	Whether Sphinx should generate verbose output.	`true`
`warningsAsErrors`	Whether warnings should be treated as errors.	`false`
`force`	Whether Sphinx should generate output for all files instead of only the changed ones.	`false`
`tags`	Additional tags to pass to Sphinx. See the Sphinx tag documentation for more information.	

CHAPTER

FIVE

BUILDING PDFS

The `sphinx-maven` plugin has experimental support for PDF generation. You'll turn it on by using the pdf builder, e.g.:

```
<plugin>
  <groupId>org.tomdz.maven</groupId>
  <artifactId>sphinx-maven-plugin</artifactId>
  <version>1.0.3</version>
  <configuration>
    <builder>pdf</builder>
    <outputDirectory>${project.reporting.outputDirectory}/pdf</outputDirectory>
  </configuration>
</plugin>
```

You'll likely also have to add some additional configuration options to your `conf.py` file (usually in `src/site/sphinx`) to tell the pdf builder what to do. At a minimum you'll probably need to point it to the index page by adding this to the end:

```
# -- Options for PDF output ---------------------------------------------------
pdf_documents = [
    ('index', u'<file name>', u'<document name>', u'<author>'),
]
```

For additional options see the Sphinx section of the rst2pdf manual.

Please note that alpha channels in the images (i.e. PNGs) are not supported, and will be replaced with black pixels. This is most likely not what you want, so please don't use alpha channels in the images.

CHAPTER

SIX

A NOTE ON MEMORY USAGE

Sphinx is run via Jython which will generate lots of small classes for various Python constructs. This means that the plugin will use a fair amount of memory, especially PermGen space (a moderate plugin run will likely use about 80mb of PermGen space). Therefore we suggest to either run maven with at least 256mb of heap and 128mb of PermGen space, e.g.

> MAVEN_OPTS="-Xmx256m -XX:MaxPermSize=128m" mvn site

or use the fork parameter of the plugin, e.g.:

```
<plugin>
  <groupId>org.tomdz.maven</groupId>
  <artifactId>sphinx-maven-plugin</artifactId>
  <version>1.0.3</version>
  <configuration>
    <fork>true</fork>
    <argLine>-Xmx256m -XX:MaxPermSize=128m</argLine>
  </configuration>
</plugin>
```

CHAPTER

SEVEN

UPDATING SPHINX

The project comes with a bash script which will update the embedded sphinx installation automatically. Simple invoke it like so:

```
./src/main/build/update-sphinx.sh
```

7.1 How the update script works

1. It sets up a temporary working directory `target/sphinx-tmp` and cd's into it.
2. It downloads the Jython installer for version 2.5.2 from Sourceforge.
3. It downloads the `ez_setup.py` script which will setup `easy_install`.
4. It installs Jython in the temporary directory.
5. It installs `easy_install` in the temporary directory.
6. It uses `easy_install` to install `docutils`, `pygments`, `jinja2`, `sphinx`, and `rst2pdf`.
7. `rst2pdf` depends on `ReportLab`, but unfortunatly that won't install directly with `easy_install`. The reason for that is that by default it is trying to install native extensions which won't work on Jython. To workaround that, the script downloads the `ReportLab` distribution directly, unpacks it, removes the native extensions (the setup script will be fine with it), and then runs the installation manually.
8. Both `reportlab` and `rst2pdf` have bugs with Jython (see e.g. this rst2pdf bug) which we'll patch and then trigger Jython to pre-compile the modules again.
9. Finally, we create the `sphinx.jar` out of the installed modules, and move it to `src/main/resources` (which will cause it to be included as a file in the plugin).

Steps 1-8 are performed by the *setup-jython-env.sh* script which is executed by the *update-sphinx.sh* script.

CHAPTER

EIGHT

FIXING BUGS IN ONE OF THE PYTHON LIBRARIES

Occasionally there are bugs in one of the python libraries, either plain bugs or bugs when running under Jython, that need to be fixed for sphinx-maven to work. In this case, you can use the `setup-jython-env.sh` script to setup an unpacked, editable sphinx jython environment:

```
./src/main/build/setup-jython-env.sh
```

This script will create a temporary folder `target/sphinx-tmp` into which it installs Jython and all relevant libraries plus patch them as necessary (as described above).

Now you can simply use that environment directly:

```
"$SPHINX_MAVEN_DIR/target/sphinx-tmp/jython/bin/sphinx-build" -a -E -n -b pdf src/site/sphinx target/
```

where `SPHINX_MAVEN_DIR` points to where you have checked out the sphinx maven project.

The neat thing with this is that you can now edit the python code directly. The packages are under:

```
target/sphinx-tmp/jython/Lib/site-packages
```

Once you're done, simply create a patch file to an untouched sphinx + dependencies installation (see below). E.g. for `rst2pdf`

```
diff -dur -x "*.pyc" -x "*.class" -x "requires.txt" \
    /Library/Python/2.5/site-packages/rst2pdf-0.92-py2.5.egg \
    target/sphinx-tmp/jython/Lib/site-packages/rst2pdf-0.92-py2.5.egg \
    > src/main/build/rst2pdf.patch
```

And for `reportlab`

```
diff -dur -x "*.pyc" -x "*.class" -x "hyphen.mashed" \
    /Library/Python/2.5/site-packages/reportlab-2.5-py2.5-macosx-10.7-x86_64.egg/reportlab \
    target/sphinx-tmp/jython/Lib/site-packages/reportlab \
    > src/main/build/reportlab.patch
```

CHAPTER

NINE

RUNNING NORMAL SPHINX

If you want to compare to the normal sphinx, install it like this:

```
easy_install-2.5 sphinx rst2pdf
```

Note that this uses Python version 2.5 which is what Jython 2.x provides. Now run Sphinx like so in your project's root directory:

```
sphinx-build -v -a -E -n -b html src/site/sphinx target/site
sphinx-build -a -E -n -b pdf src/site/sphinx target/site/pdf
```

CHAPTER

TEN

FREQUENTLY ASKED QUESTIONS

When I put documentation generated by Sphinx on Github pages, then it looks all mangled and broken. Why is that ?

> Github pages by default uses a system called Jekyll to generate the documentation. This usually works just fine with already generated content. However Jekyll regards directory names beginning with an underscore to be special and won't make them available on the website. Sphinx however uses directories like *_source* and *_static* for the site which means it won't work properly. Fortunately the fix is easy. In your *gh-pages* branch of your project on github, simply create a file called *.nojekyll* in the root folder. Once you pushed that file, Jekyll will be turned off for the site and the documentation should look as intended. If you are using the Github maven plugins to deploy your site, then you can tell the *site-maven-plugin* to automatically maintain the *.nojekyll* file by adding:
>
> ```
> <noJekyll>true</noJekyll>
> ```
>
> to the configuration of the *site-maven-plugin* plugin (since version 0.5 of the plugin).