Deliverable 7.7 Contribution guidelines for future development of Spine Toolbox

Revision.................................... 1
Preparation date ....................... 2021-09-01 (M48)
Due date ................................. 2021-09-30 (M48)
Lead contractor......................... VTT

Authors:
Erkka Rinne ......................... VTT
Pekka T Savolainen .............. VTT
Manuel Marin......................... KTH
Antti Soininen ..................... VTT
Maren Ihlemann ..................... KUL

This work is licensed under the Creative Commons Attribution-ShareAlike 4.0 International License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/4.0/.
### Deliverable administration

<table>
<thead>
<tr>
<th>No &amp; name</th>
<th>D7.7 Contribution guidelines for future development of Spine Toolbox</th>
</tr>
</thead>
<tbody>
<tr>
<td>Status</td>
<td>Final</td>
</tr>
<tr>
<td>Due</td>
<td>M48</td>
</tr>
<tr>
<td>Date</td>
<td>2021-09-17</td>
</tr>
<tr>
<td>Author(s)</td>
<td>Erkka Rinne VTT</td>
</tr>
<tr>
<td></td>
<td>Pekka T Savolainen VTT</td>
</tr>
<tr>
<td></td>
<td>Manuel Marin KTH</td>
</tr>
<tr>
<td></td>
<td>Antti Soininen VTT</td>
</tr>
<tr>
<td></td>
<td>Maren Ihlemann KUL</td>
</tr>
<tr>
<td>Reviewer(s)</td>
<td>Juha Kivivuoma VTT</td>
</tr>
<tr>
<td></td>
<td>Iasonas Kouveliotis-Lysikatos KTH</td>
</tr>
</tbody>
</table>

### Description of the related task and the deliverable

**Extract from DoW**

**T7.3 Exploitation and open sourcing planning (M1–M48)**

Task leader: VTT; Participants: All

Central element of Task 7.3 is the deployment of the tools to potential users. Through the training and dissemination activities, the aim is also to build an active user community that will continue to use and develop the model also in the future. We will organize training workshops as side events of energy related conferences and workshops such as Wind and Solar Integration Workshops, EERA workshops and IEEE events. In addition, a workshop will be organized in Brussels in M45 as part of one the Spine workshops. Training is also provided through webinars.

Spine project will also produce exploitable results. The task leader together with the relevant project participants will process the case study results into concise conclusions that can be communicated to the stakeholders according to the planned communications (section 2.2.2 of the proposal). The task will also be responsible for carrying out the communications that the exploitation plan has set out. Discussions with policy makers, regulators, transmission system operators, energy companies and toolbox users. The task will also produce and publish replies to the questions posed by Spine Toolbox users. These efforts will be shared between the project partners depending on the expertise needed for each exploitation support.

The utilisation of Spine Toolbox after the project will be ensured through an open source community. The community will consist of Spine project members and other interested parties. The exploitation of Spine Toolbox will continue after the project through an open source community that will consist of Spine project members and all other interested parties. In order to secure and manage the continuous development effort, this task will setup a non-profit organization to oversee the development. All project partners will join the non-profit as initial members. The task will write a deliverable that establishes the governing rules for the non-profit and the rules of entry for new members. In order to assist the future development and maintenance of the model, contribution guidelines will be developed as a deliverable of this task by all project partners together. These will govern how new contributions are to be designed, documented and tested in order to be accepted to the master branch of Spine Toolbox.

All partners of WP7 will participate in deploying the open source tools whenever it concerns the work they are involved with. The open sourcing will include the following elements:

- Deploy documentation and rules for contributions to the open source tools
- Deploy test systems and contribution validation
- Make examples how Spine Toolbox can be used in energy system modelling in collaboration with some of the case studies – and disseminate them
- Generate charts showing how to access model dimensions together with WP3 – and deploy them


− Place code and case studies in public servers (also conversion tools and primary data when possible)

This task will establish a non-profit organization that will oversee the future development of Spine Toolbox and Spine Model [now SpineOpt].

**D7.7: Contribution guidelines for future development of Spine Toolbox:** The report will define the guidelines for future contributions to the Spine model [SpineOpt] and Spine Toolbox.

<table>
<thead>
<tr>
<th>Planned resources PM of T7.3</th>
<th>VTT</th>
<th>UCD</th>
<th>KUL</th>
<th>KTH</th>
<th>ER</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>VTT</td>
<td>4.0</td>
<td>4.0</td>
<td>4.0</td>
<td>4.0</td>
<td>1.0</td>
<td>17.0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Comments</th>
<th>V</th>
<th>Date</th>
<th>Authors</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0.12</td>
<td>2021-09-10</td>
<td>VTT</td>
<td>First draft for internal review</td>
</tr>
<tr>
<td></td>
<td>1.0</td>
<td>2021-09-17</td>
<td>VTT</td>
<td>Review comments, submitted version</td>
</tr>
</tbody>
</table>

**Disclaimer**

The information in this document is provided as is and no guarantee or warranty is given that the information is fit for any particular purpose. The user thereof uses the information as its sole risk and liability. The document reflects only the author’s views and the Community is not liable for any use that may be made of the information contained therein.
# Table of contents

1. Introduction..................................................................................................................5

   2.1 Reporting Bugs...........................................................................................................5
   2.1.1 Due Diligence........................................................................................................5
   2.1.2 What to Put in Your Bug Report ...........................................................................5
   2.2 Feature Requests.......................................................................................................6
   2.3 Submitting features/bugfixes ....................................................................................6
   2.3.1 Coding Style..........................................................................................................6
   2.3.2 Commit messages ..................................................................................................6
   2.3.3 Contributing to the User Guide ............................................................................7
   2.3.4 Contributing to the Spine Toolbox Graphical User Interface ...............................7
   2.3.5 Version Control Branching ..................................................................................7
   2.3.6 Test-driven development is your friend...............................................................8
   2.3.7 Full example .......................................................................................................8

3. SpineOpt contributing guidelines ..................................................................................9
   3.1 Reporting Issues .....................................................................................................9
   3.2 Contributing.............................................................................................................9
   3.3 Style Guidelines ......................................................................................................9

References .........................................................................................................................10
1. INTRODUCTION

This document describes the guidelines for both internal and external contributions to Spine Toolbox as well as SpineOpt. Other contributions than writing software are also possible and highly welcome. These include supporting new users, reporting bugs, identifying requirements, providing graphics and web design, assisting with project infrastructure and writing documentation. More details about possible modes of commitment are included in the governing rules of Spine [D7.6]. Interested contributors should use regular practices of open-source development (regarding software code) or contact project members for other types of contributions.

The next two sections contain contribution guidelines, from the software development point of view, for Spine Toolbox (including related software components) and SpineOpt (and related other packages), respectively. The latest versions of these guidelines are available online.¹

2. CONTRIBUTION GUIDE FOR SPINE TOOLBOX

All are welcome to contribute! This guide is based on a set of best practices for open source projects [JF18].

2.1 Reporting Bugs

2.1.1 Due Diligence

Before submitting a bug report, please do the following:

Perform basic troubleshooting steps.

1. **Make sure you’re on the latest version.** If you’re not on the most recent version, your problem may have been solved already! Upgrading is always the best first step.

2. **Try older versions.** If you’re already on the latest release, try rolling back a few minor versions (e.g. if on 1.7, try 1.5 or 1.6) and see if the problem goes away. This will help the developers narrow down when the problem first arose in the commit log.

3. **Try switching up dependency versions.** If you think the problem may be due to a problem with a dependency (other libraries, etc.). Try upgrading/downgrading those as well.

4. **Search the project’s bug/issue tracker to make sure it’s not a known issue.** If you don’t find a pre-existing issue, consider checking with the maintainers in case the problem is non-bug-related. The Spine Toolbox issue tracker is here: https://github.com/Spine-project/Spine-Toolbox/issues.

2.1.2 What to Put in Your Bug Report

Make sure your report gets the attention it deserves: bug reports with missing information may be ignored or punted back to you, delaying a fix. The below constitutes a bare minimum; more info is almost always better:

1. What version of the Python interpreter are you using? E.g. Python 2.7.3, Python 3.6?


2. What operating system are you on? Windows? (Vista, 7, 8, 8.1, 10). 32-bit or 64-bit? Mac OS X? (e.g. 10.7.4, 10.9.0) Linux (Which distro? Which version of that distro? 32 or 64 bits?) Again, more detail is better.

3. Which version or versions of the software are you using? If you have forked the project from Git, which branch and which commit? Otherwise, supply the application version number (Help → About in the menu bar). Also, ideally, you have followed the advice above and have ruled out (or verified that the problem exists in) a few different versions.

4. How can the developers recreate the bug? What were the steps used to invoke it? A screenshot demonstrating the bug is usually the most helpful thing you can report (if applicable). Relevant output from the Event Log or debug messages from the console of your run, should also be included.

2.2 Feature Requests

The developers of Spine Toolbox are happy to hear new ideas for features or improvements to existing functionality. There isn’t a specific format for requesting new features. Just fill out the required fields on the issue tracker and give a description of the new feature. A picture accompanying the description is a good way to get your idea into development faster. Before you make a new issue, always check that there isn’t a related idea already open in the issue tracker. If you have an idea on how to improve an existing idea, just join the conversation.

2.3 Submitting features/bugfixes

If you feel like you can fix a bug that’s been bothering you or you want to add a new feature to the application but the developers seem to be too busy with something else, please follow the instructions in the following sections on how to contribute code.

2.3.1 Coding Style

Follow the style you see used in the repository! Consistency with the rest of the project always trumps other considerations. It doesn’t matter if you have your own style or if the rest of the code breaks with the greater community - just follow along.

Spine Toolbox coding style follows PEP-8 style guide for Python code with the following variations:

- Maximum line length is 120 characters. Longer lines are acceptable for a good reason.
- Google style\(^2\) docstrings with the title and input parameters are required for all classes, functions, and methods. For small functions or methods, only the summary is necessary. Return types are highly recommended but not required if it is obvious what the function or method returns.
- Other deviations from PEP-8 can be discussed.

2.3.2 Commit messages

The commit message should tell what was changed and why. Details on how it was done can usually be left out if the code itself is self-explanatory (remember source comments too!). Separate the subject line from the body with a blank line. The subject line (max. 50 chars) should explain in condensed form what happened using imperative mood, i.e. using verbs like ‘change’, ‘fix’, or ‘add’. Start the subject line with a capital letter. Do not use the issue number

\(^2\) http://google.github.io/styleguide/pyguide.html
on the subject line, as it does not tell much to a person who’s not aware of that particular issue. For more info see Chris Beams’ ‘Seven rules of a great Git commit message’ [CB14].

A good example (inspired by [CB14]):

Fix bugs when updating parameters in foo and bar

Body of the commit message starts after a blank line. Explain here in more detail the reasons why you made the change, how things worked before and how they work now. Also explain why

You can use hyphens to make bulleted lists:
- Foo was added because of bar
- Baz was not used so it was deleted

Add references to issue tracker (if any) at the end.

Solves: #123
See also: #456, #789

2.3.3 Contributing to the User Guide

Spine Toolbox uses Sphinx to create HTML pages from restructured text (.rst) files. The .rst files are plain text files that are formatted in a way that Sphinx understands and is able to turn into HTML. Please see this brief introduction³ for more on reStructured text. You can modify the existing or create new .rst files into docs/source directory. When you are done editing, run bin/build_doc.bat on Windows or bin/build_doc.py on other systems to build the HTML pages to check the result before making a commit. The created pages are found in docs/build/html directory. After a commit, the User Guide is built automatically by readthedocs.org. The latest User Guide is available in https://spine-toolbox.readthedocs.io/en/latest/.

2.3.4 Contributing to the Spine Toolbox Graphical User Interface

If you want to change or add new widgets into the application, you need to use the bin/build_ui.bat (Windows) or bin/build_ui.py (other systems) scripts. The main design of the widgets should be done with Qt Designer (designer.exe or designer) that is included with PySide2. The files produced by Qt Designer are XML files (.ui). You can also embed graphics (e.g. icons, logos, etc.) into the application by using Qt Designer. When you are done modifying widgets in the designer, you need to run the build_ui script for the changes to take effect. This script uses tools provided in the PySide2 package to turn .ui files into Python files, in essence rebuilding the whole Spine Toolbox user interface.

Styling the widgets should be done with Qt Style Sheets⁴ in code. Please avoid using style sheets in Qt Designer.

2.3.5 Version Control Branching

Always make a new branch for your work, no matter how small. This makes it easy for others to take just that one set of changes from your repository, in case you have multiple unrelated changes floating around. A corollary: don’t submit unrelated changes in the same branch/pull request! The maintainer shouldn’t have to reject your awesome bugfix because the feature you put in with it needs more review.

Name your new branch descriptively, e.g. issue#XXX-fixing-a-serious-bug or issue#ZZZ-cool-new-feature. New branches should in general be based on the latest master branch. In case you want to include a new feature still in development, you can also start working from its branch. The developers will backport any relevant bugfixes to previous or upcoming releases under preparation.

If you need to use code from an upstream branch, please use `git-rebase` if you have not shared your work with others yet. For example: you started working on an issue, but now the upstream branch (master) has some new commits you would like to have in your branch too. If you have not yet pushed your branch, you can now rebase your changes on top of the upstream branch:

```
$ git pull origin master:master
$ git checkout my_branch
$ git rebase master
```

Avoid merging the upstream branch to your issue branch if it’s not necessary. This will lead to a more linear and cleaner history.

Finally, make a pull request from your branch so that the developers can review your changes. You might be asked to make additional changes or clarifications or add tests to prove the new feature works as intended.

### 2.3.6 Test-driven development is your friend

Any bug fix that doesn’t include a test proving the existence of the bug being fixed, may be suspect. Ditto for new features that can’t prove they actually work.

It is recommended to use test-first development as it really helps make features better designed and identifies potential edge cases earlier. Writing tests before the implementation is strongly encouraged.

See [Unit testing guidelines](https://spine-toolbox.readthedocs.io/en/latest/unit_testing_guidelines.html) for more information.

### 2.3.7 Full example

Here’s an example workflow. Your username is `yourname` and you’re submitting a basic bugfix.

**Preparing your Fork**

1. Click [Fork](https://github.com) on Github, creating a new repository e.g. `yourname/Spine-Toolbox`
2. Clone your project: `git clone git@github.com:yourname/Spine-Toolbox`
3. `cd Spine-Toolbox`
4. Create a virtual environment and install requirements
5. Create a branch: `git checkout -b foo-the-bars master`

**Making your Changes**

1. Add an entry to `CHANGELOG.md`.
2. Write tests expecting the correct/fixed functionality; make sure they fail.
3. Hack, hack, hack.

---


4. Run tests again, making sure they pass.
5. Commit your changes: `git commit -m "Foo the bars"

Creating Pull Requests

1. Push your commit to get it back up to your fork: `git push origin HEAD`
2. Visit Github, click the handy **Pull request** button that it will make upon noticing your new branch.
3. In the description field, write down the issue number (if submitting code fixing an existing issue) or describe the issue + your fix (if submitting a wholly new bugfix).
4. Hit **Submit**! And please be patient - the maintainers will get back to you when they can.

3. **SpineOpt Contributing Guidelines**

Thanks for taking the plunge!

3.1 Reporting Issues

- It's always good to start with a quick search for an existing issue to post on, or related issues for context, before opening a new issue.
- Including minimal examples is greatly appreciated.
- If it's a bug, or unexpected behaviour, reproducing it on the latest development version (`Pkg.checkout("SpineOpt")`) is a good gut check and can streamline the process, along with including the first two lines of output from `versioninfo()`.

3.2 Contributing

- Feel free to open, or comment on, an issue and solicit feedback early on, especially if you're unsure about aligning with design goals and direction, or if relevant historical comments are ambiguous.
- Pair new functionality with tests, and bug fixes with tests that fail pre-fix. Increasing test coverage as you go is always nice.
- Aim for atomic commits, if possible, e.g. ‘Change foo behaviour like so’ or ‘Bar handles such and such corner case’, rather than ‘Update foo and bar’ or ‘Fix typo’ or ‘Fix bar better’.
- Pull requests will be tested against the release and development branches of Julia, so using `Pkg.test("SpineOpt")` as you develop can be helpful.
- The style guidelines outlined below are not the personal style of most contributors, but for consistency throughout the project, we should adopt them.

3.3 Style Guidelines

- Include spaces:
  - after commas,
  - around operators: ‘=’, ‘<’, comparison operators, and generally around others,
  - but not after opening parentheses or before closing parentheses.
- Use four spaces for indentation (test data files and Makefiles excepted).
- Don't leave trailing whitespace at the end of lines.
- Don't go over the 119 per-line character limit.
- Avoid squashing code blocks onto one line, e.g. `for foo in bar; baz += qux(foo); end`.
- Don't explicitly parameterize types unless it's necessary.
- Never leave things without type qualifications. Use an explicit `::Any`.
- Order method definitions from most specific to least specific type constraints.

REFERENCES

https://chris.beams.io/posts/git-commit/  