Subplot website tutorial

The Subplot project

2022-08-13 05:27

Contents

1 Introduction

This is a tutorial for the Subplot software. It is aimed at people who want to see what using Subplot is like.

This tutorial walks you through how to use Subplot to verify that the Subplot web site, `subplot.liw.fi`, works and has the right kind of content. The full source code of the tutorial is part of the Subplot source tree, and is checked every time the software changes.

The source code for this tutorial is at https://gitlab.com/subplot/subplot/-/tree/main/examples/website[1]. You need to read the source code to understand everything.

[1] `https://gitlab.com/subplot/subplot/-/tree/main/examples/website`

2 Installing Subplot

First, you need to install[2] Subplot. The web site has instructions.

3 Acceptance criteria

The fundamental acceptance criteria for the Subplot web site is that it exists, talks about Subplot, and doesn't talk about Obnam, the backup program.

To use Subplot to verify that you need to write a document that explains everything to all the people involved in the project. The document will contain semi-formal *scenarios* that document how a program would verify the site works. For this tutorial, that document is this document, website.md[3]. It uses website.yaml[4] for bindings, and website.py[5] for step implementations. Consult the source code for how to mark up scenarios and the other files.

Each scenario is placed under its own heading. This makes it easy to refer to each scenario.

3.1 Web page talks about Subplot

This scenario verifies the Subplot home page exists and talks about Subplot.

given website **https://subplot.liw.fi**
when I look at the front page
then it mentions "**Subplot**"
and it mentions "**acceptance criteria**"
and it mentions "**verified**"

3.2 Doesn't talk about Obnam

This scenario verifies the Subplot home page doesn't talk about Obnam, the backup program.

given website **https://subplot.liw.fi**
when I look at the front page
then it doesn't contain "**Obnam**"

[2]`https://subplot.liw.fi/download/`
[3]`https://gitlab.com/subplot/subplot/-/tree/main/examples/website/website.md`
[4]`https://gitlab.com/subplot/subplot/-/tree/main/examples/website/website.yaml`
[5]`https://gitlab.com/subplot/subplot/-/tree/main/examples/website/website.py`

4 Using Subplot

In order to follow along with this tutorial, you should download website.md[6], website.yaml[7], and website.py[8].

4.1 Generate typeset documents

To generate typeset versions of this document, run the following commands:

```
$ subplot docgen website.md -o website.html
$ subplot docgen website.md -o website.pdf
```

Open up the files to see what they look like.

4.2 Generate and run test program

To generate and run a test program from this document, run the following command:

```
$ subplot codegen website.md -o test.py
$ python3 test.py --log test.log
srcdir /home/liw/pers/subplot/git/examples/website
datadir /tmp/tmp3Ohj6kvb
scenario: Doesn't talk about Obnam
  step: given website https://subplot.liw.fi
  step: when I look at the front page
  step: then it doesn't contain "Obnam"
scenario: Web page talks about Subplot
  step: given website https://subplot.liw.fi
  step: when I look at the front page
  step: then it mentions "Subplot"
  step: then it mentions "acceptance criteria"
  step: then it mentions "verified"
OK, all scenarios finished successfully
$
```

Assuming you get similar output, then you know that everything is working.

Open `test.py` and `test.log`. The log file is useful if a scenario fails.

4.3 Some explanations

The website.yaml[9] file has a list of bindings. Each binding uses one of the key words (given, when, then) and a pattern, and also a function name. The pattern

[6]https://gitlab.com/subplot/subplot/-/tree/main/examples/website/website.md
[7]https://gitlab.com/subplot/subplot/-/tree/main/examples/website/website.yaml
[8]https://gitlab.com/subplot/subplot/-/tree/main/examples/website/website.py
[9]https://gitlab.com/subplot/subplot/-/tree/main/examples/website/website.yaml

can be fixed text, or it can extract parts of the scenario step and pass those to the function.

Patterns which capture and extract parts of the scenario steps have the form `{name}` for single words, or `{name:text}` for multiple words. This allows for an easy way to, for example, use the same step for different web sites.

The generated test program will call each function, with extracted values, in the appropriate order. If the function finishes successfully, the step succeeds. If the function raises an exception, the step fails.

The `assert` statement is typically used to verify things in a step function. For more useful error messages, Subplot provides the functions `assert_eq` and `assert_ne` that can be used as well.

Extracted parts of steps are passed to functions as keyword arguments. Such arguments must exist, which is why you will typically see them added as `name=None` in the function definition.

Each step function also gets a dict-like "context" variable. A new, empty context is created for each scenario. This allows data to be passed conveniently between step functions.

5 Changing the document

We now have minimal verification of the Subplot web site, but it can be improved. One possible improvement would be to verify that there is a sub-page with contact information for the project, and a blog. To achieve this, you should make the following changes:

- change the "given a website" step to be called "given a web page", by changing the bindings file, and the markdown file; then re-run the commands to generate documentation and test program, and re-run the test program: it should still pass
- also change the "when I look at the front page" to say "when I look at the page"; then re-run all the commands again; these changes are not functional, but it's important to make sure everything is easily understood by all the people involved
- add a scenario to verify that there is a page `https://subplot.liw.fi/contact` and that it contains "Lars" and "Daniel"
- add a new step that verifies the page content type is `text/html`: add to the relevant scenario a step "then it has a content type text/html", and the corresponding binding to the YAML file, and a new function to the Python file
  - you can get the content type from the return value of `requests.get` as `r.headers["content-type"]`