Fable tutorial

The Fable project Lars Wirzenius Daniel Silverstone

0.2-64-g5c15f80

Contents

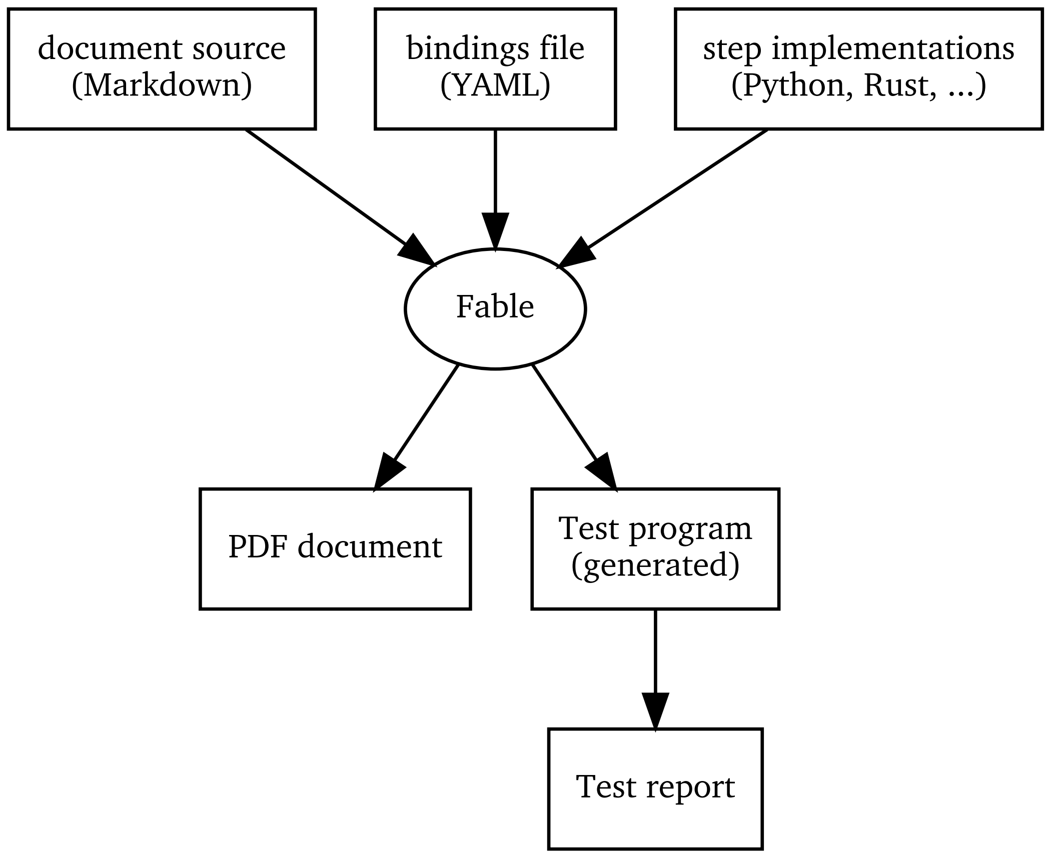
1 Introduction

Fable is a tool for **automated acceptance testing**. Acceptance tests define if a program (or other system) is accepable, from a user's or client's or employer's point of view. Basically, in a commercial consulting setting, acceptance tests determine if the client is willing to pay the bill for the software having been developed.

Acceptance tests differ from other integration tests by looking at the software *only* from the user's point of view, not the developer's. Because of this, Fable produces two things: a standalone document aimed at the user to describe the accepance tests, and a report from executing the automated tests.

The document is meant to be a vehicle of communication between the various stakeholders in a project, when specifying in detail what the acceptance criteria are. It both explains the criteria, and how they're verified, and lets the verification to be done automatically.

Fable generates the document, and a test program to execute the tests. The generated test program produces the test report.



2 Example

Here is a quick example of using Fable. Let's imagine you are in need of a new implementation of the **echo**(1) command line utility, and you want to commission the implementation. You need to specify acceptance tests. After negotations with the developers, and their project manager, and your own sales and test teams, you end up with the following requirements:

- If echo is run without arguments, it should write a newline character, and exit with a zero exit code.
- If echo is run with arguments, it should write each argument, separated by a space character, and then a newline character, and finally exit with a zero exit code.

These are specified for Fable in a Markdown document. A minimal one for echo might look like this:

~~~
No arguments
=============================================================================

Run `/bin/echo` without arguments.

```fable
when user runs echo without arguments
then exit code is 0
and standard output contains a newline
~~~

~~~
and standard error is empty
```

Hello, world
=============================================================================

Run `/bin/echo` to produce the output "hello, world".

```fable
when user runs echo with arguments hello, world
then exit code is 0
and standard output contains "hello, world"
and standard error is empty
```
~~~

Acceptance tests in Fable are expresses as **scenarios**, which roughly correspond to use cases and acceptance requirements. A scenario has an overall structure of given/when/then:

- **given** some initial context (setup)
- **when** some action is taken (the thing to test)
- **then** the end result looks in a specific way (checks)

These are embedded in markdown using **fenced code blocks** marked as containing `fable` text. Fable extracts these and generates a test program to execute them. The scenario steps are formulated in a way that all stakeholders in the project understand. This is crucial.

Fable is not magic artificial intelligence. Each scenario step needs to be implemented by programmers so that computers also understand them. Each step corresponds to a function, currently in Python, and a YAML file **binds** the step to the function, using a regular expression to capture relevant parts of the step. A binding file might look like this:

```
- when: user runs echo without arguments
  function: run_echo_without_args

- when: user runs echo with arguments (?P<args>.+)
  function: run_echo_with_args

- then: exit code is (?P<exit_code>\d+)
  function: exit_code_is_zero

- then: standard output contains a newline
  function: stdout_is_a_newline

- then: standard output contains "(?P<text>.*)"
  function: stdout_is_text

- then: standard error is empty
  function: stderr_is_empty
```

This means that for a step saying ‘then standard output contains “foo”’, the Python function `stdout_is_text` is called and given the string `foo` as an argument.

You, or your test team, in collaboration with the development team, need to supply the bindings and the Python functions. Fable produces the code that extracts the interesting parts of scenario steps, and calls your functions in the right order, plus any other scaffolding needed.

3 Installing Fable

To install Debian for Debian unstable, add the following to your APT sources lists:

```
deb http://ci-prod-controller.vm.liw.fi/debian unstable-ci main
```

Then run the following commands:

```
apt update
apt install fable
```

To run Fable from git, you need to clone the git repository and need Python 3, and the Markdown parser installed, as well as document typesetting tools, from Debian 10 (buster) or later:

```
sudo apt update
sudo apt install make git locales-all pandoc python3 \
  python3-pandocfilters python3-commonmark-bkrs python3-yaml \
  texlive-latex-base texlive-fonts-recommended \
  graphviz librsvg2-bin
git clone git://git.liw.fi/fable-poc
cd fable-poc
```

Note that `fable-poc` contains sample versions of the files for the echo acceptance tests (`echo.md`, `echo.yaml`, `echo.py`, and `echo-prelude.py`) so that you don’t need to type them in yourself.

4 Producing a document

To produce a PDF document of the echo requirements, for you to review, the following commands are needed:

```
./ftt-docgen --html echo.md
./ftt-docgen --pdf echo.md
```

`echo.md` is the markdown input file. `echo.yaml` is the bindings, and its name is inferred by `ftt-docgen`. `echo.pdf` and `echo.html` are the formatted documents. The input files are included in the `fable-poc` git repository.

5 Running the tests

To generate the test program, and running it to produce the test report, the following commands are needed:

```
./ftt-codegen --run echo.md
```

The output of the last command is the test report:

```
OK: No arguments
OK: Hello, world
```

`echo.yaml` and `echo.md` are again the bindings and markdown files. Also, `echo.py` has the Python functions, needed at runtime, though not referenced above. The names of `echo.py` and `echo.yaml` are inferred by `ftt-codegen`.

6 Exercise

Your mission, should you choose to accept it, is to write a Fable to write an acceptance test suite for simple uses of the **cp**(1) command. You can model it after the **echo**(1) one, and you get to specify the actual acceptance criteria yourself, but at minimum you need to check that a regular file can be copied to another name and that the end result is the same. You should try to get `ftt-docgen` produce a PDF of your fable.

7 Finally

It would great if you could give us, Lars and Daniel, feedback on Fable and this tutorial.

Pre:

- What kinds of testing do you already do?
- Who are your usual stakeholders?
- What sorts of things would you like to test better?

Post:

- What do you think of Fable as a concept?
- What's good about Fable?
- What's less good about Fable?
- Did you have any problems following the tutorial?
- Do you see yourself using Fable in the future?
- Do you see Fable as filling a gap?