Contents

Good practices

Data Intuitive Tuesday - January 26, 2021

> blabla

We will introduce testing using the same components we used earlier to introduce the viash approach:

- `convert_plot`
- `combine_plots`

convert_plot

`convert_plot` converts a PDF (map) into a `.png` version.

The viash configuration

We covered the functionality of this component already in the previous sections. In this section, we show how to add (unit) tests to the component. Let see what the directory structure of the (updated) component looks like. We put the components in the `civ6_save_renderer` namespace now that we know how this works:

```
> tree src/civ6_save_renderer/convert_plot
src/civ6_save_renderer/convert_plot
├── config.vsh.yaml
├── script.sh
└── test
    └── run_test.sh

1 directory, 3 files
```

Just like in the viash primer (of the previous section) there is a viash config (`config.vsh.yaml`) and a script (`script.sh`). Let us take a closer look at both of these:

`src/civ6_save_renderer/convert_plot/config.vsh.yaml`:

```yaml
functionality:
  name: convert_plot
  namespace: civ6_save_renderer
  description: Convert a plot from pdf to png.
  version: "1.0"
  authors:
    - name: Robrecht Cannoodt
      email: rcannood@gmail.com
      roles: [maintainer, author]
      props: {github: rcannood, orcid: 0000-0003-3641-729X}
  arguments:
    - name: "--input"
      alternatives: [-i]
```

```
      type: file
      required: true
      default: "input.pdf"
      must_exist: true
      description: "A PDF input file."
    - name: "--output"
      alternatives: [-o]
      type: file
      required: true
      default: "output.png"
      direction: output
      description: "Output path."
  resources:
    - type: bash_script
      path: script.sh
  tests:
    - type: bash_script
      path: test/run_test.sh
    - path: https://www.w3.org/WAI/ER/tests/xhtml/testfiles/resources/pdf/dummy.pdf
platforms:
  - type: docker
    image: dpokidov/imagemagick
    setup:
      - type: apt
        packages: [ "tesseract-ocr" ]
  - type: native
```

src/civ6_save_renderer/convert_plot/test/run_test.sh:

```
#!/usr/bin/env bash

set -ex

# Run our component
convert_plot \
  -i dummy.pdf \
  -o dummy.png

[[ ! -f dummy.png ]] && echo "No output generated!" && exit 1

# Run OCR on the png file
tesseract dummy.png dummy-ocr

[[ ! `grep Dummy dummy-ocr.txt` ]] && echo "Not the correct content" && exit 1

echo ">>> Test finished successfully"
```

The only differences with before are:

1. The addition of the `tests`
2. An extra `apt` package to be installed for running the tests (see later).

Tests

Specifying the tests is not different from specifying the `resources` in the viash configuration. In this case, we have two resources: one is the script that contains the test code and one is a dummy PDF file that is fetched from the web during testing. We could also add a PDF file to the repository and point to that instead.

The test script itself defines two tests:

1. A test to see if an output file is effectively created by our component
2. A test that extracts the text from the resulting `png` file in order to verify the content is still the same as the original.

In order to run the second step, we install a package `tesseract` that performs the OCR.

Platforms

The only difference with the `platforms` definition earlier is the installation of an additional package in the container.

Running the tests

In order to run the tests using the default platform (`docker` in our current example), we can simply run:

```
> viash test src/civ6_save_renderer/convert_plot/config.vsh.yaml
Running tests in temporary directory: '<...>/workspace/viash_temp/viash_test_convert_plot106216272
============================================================================
+<...>/workspace/viash_temp/viash_test_convert_plot106216272389067370/build_executable/convert_plot
> docker build -t civ6_save_renderer/convert_plot:1.0 <...>/workspace/viash_temp/viashsetupdocker-c
Sending build context to Docker daemon  17.41kB

Step 1/2 : FROM dpokidov/imagemagick
 ---> 0ce61e775be8
Step 2/2 : RUN apt-get update &&   apt-get install -y tesseract-ocr &&   rm -rf /var/lib/apt/lists/
 ---> Using cache
 ---> a92be5886a41
Successfully built a92be5886a41
Successfully tagged civ6_save_renderer/convert_plot:1.0
============================================================================
+<...>/workspace/viash_temp/viash_test_convert_plot106216272389067370/test_run_test.sh/run_test.sh
+ convert_plot -i dummy.pdf -o dummy.png
convert: profile 'icc': 'RGB ': RGB color space not permitted on grayscale PNG `dummy.png' @ warnin
+ [[ ! -f dummy.png ]]
+ tesseract dummy.png dummy-ocr
Tesseract Open Source OCR Engine v4.0.0 with Leptonica
Warning: Invalid resolution 0 dpi. Using 70 instead.
Estimating resolution as 157
```

```
++ grep Dummy dummy-ocr.txt
+ [[ ! -n Dummy PDF file ]]
>>> Test finished successfully
+ echo '>>> Test finished successfully'
========================================================================
SUCCESS! All 1 out of 1 test scripts succeeded!
Cleaning up temporary directory
```

Let us break down what happens here:

1. viash creates a temporary directory (configurable via `$VIASH_TEMP`)
2. The setup of the appropriate platform is executed
3. The executable for the component is built in the temporary directory
4. The test script is run

If tests are successful, the temporary directory is removed (unless `--keep` is provided as an option to `viash test`).

This is a quick way to run a test on a component.

combine_plots

We do something similar for the component that combines different `png` (map) files into one `webm` video. Let us see how we can do something similar as before so that a test can run on its own.

We refer to an article that discussed the generation of an animation from `png` image sources and does this using … ImageMagic. We use a selection of the images stored on Github.

`src/civ6_save_renderer/combine_plots/config.vsh.yaml`:

```yaml
functionality:
  name: combine_plots
  namespace: civ6_save_renderer
  description: Combine multiple images into a movie using ffmpeg.
  version: "1.0"
  authors:
    - name: Robrecht Cannoodt
      email: rcannood@gmail.com
      roles: [maintainer, author]
      props: {github: rcannood, orcid: 0000-0003-3641-729X}
  arguments:
    - name: "--input"
      alternatives: [-i]
      type: file
      required: true
      default: "plot1.png:plot2.png"
      must_exist: true
      multiple: true
      description: A list of images.
    - name: "--output"
      alternatives: [-o]
      type: file
      required: true
      default: "output.webm"
      direction: output
      description: A path to output the movie to.
    - name: "--framerate"
      alternatives: [-f]
      type: integer
```

```
      default: 4
      description: Number of frames per second.
  resources:
    - type: bash_script
      path: script.sh
  tests:
    - type: bash_script
      path: test/run_test.sh
    - path: https://github.com/hplgit/animate/raw/master/doc/src/animate/src-animate/testfiles/fram
    - path: https://github.com/hplgit/animate/raw/master/doc/src/animate/src-animate/testfiles/fram
    - path: https://github.com/hplgit/animate/raw/master/doc/src/animate/src-animate/testfiles/fram
    - path: https://github.com/hplgit/animate/raw/master/doc/src/animate/src-animate/testfiles/fram
    - path: https://github.com/hplgit/animate/raw/master/doc/src/animate/src-animate/testfiles/fram
    - path: https://github.com/hplgit/animate/raw/master/doc/src/animate/src-animate/testfiles/fram
    - path: https://github.com/hplgit/animate/raw/master/doc/src/animate/src-animate/testfiles/fram
    - path: https://github.com/hplgit/animate/raw/master/doc/src/animate/src-animate/testfiles/fram
    - path: https://github.com/hplgit/animate/raw/master/doc/src/animate/src-animate/testfiles/fram
platforms:
  - type: docker
    image: jrottenberg/ffmpeg
  - type: native
```

src/civ6_save_renderer/combine_plots/test/run_test.sh:

```
#!/usr/bin/env bash

set -e

# Run our component
png_files=`ls *.png`
png_args=""
for i in "$png_files"; do
  png_args="$png_args $i"
done
png_args_parsed=`echo -n $png_args | sed 's/ /:/g'`

combine_plots -i "$png_args_parsed" -o output.webm --framerate 1

[[ ! -f output.webm ]] && echo "No output generated!" && exit 1

echo ">>> Test finished successfully"
```

We added the `tests` and point to the frames explicitly. The test script basically generates a command line instructions (list of `png` files) based on the images that have been downloaded as resources.

```
> viash test src/civ6_save_renderer/combine_plots/config.vsh.yaml
Running tests in temporary directory: '<...>/workspace/viash_temp/viash_test_combine_plots49212700
====================================================================
+<...>/workspace/viash_temp/viash_test_combine_plots4921270052339625520/build_executable/combine_p
> docker pull jrottenberg/ffmpeg
```

```
Using default tag: latest
latest: Pulling from jrottenberg/ffmpeg
Digest: sha256:21eb739725c43bd7187982e5fa4b5371b495d1d1f6f61ae1719ca794817f8641
Status: Image is up to date for jrottenberg/ffmpeg:latest
docker.io/jrottenberg/ffmpeg:latest
================================================================
+<...>/workspace/viash_temp/viash_test_combine_plots4921270052339625520/test_run_test.sh/run_test.s
ffmpeg version 4.1 Copyright (c) 2000-2018 the FFmpeg developers
  built with gcc 5.4.0 (Ubuntu 5.4.0-6ubuntu1~16.04.11) 20160609
  configuration: --disable-debug --disable-doc --disable-ffplay --enable-shared --enable-avresample
  libavutil      56. 22.100 / 56. 22.100
  libavcodec     58. 35.100 / 58. 35.100
  libavformat    58. 20.100 / 58. 20.100
  libavdevice    58.  5.100 / 58.  5.100
  libavfilter     7. 40.101 /  7. 40.101
  libavresample   4.  0.  0 /  4.  0.  0
  libswscale      5.  3.100 /  5.  3.100
  libswresample   3.  3.100 /  3.  3.100
  libpostproc    55.  3.100 / 55.  3.100
Input #0, png_pipe, from 'concat:frame_0000.png|frame_0001.png|frame_0002.png|frame_0003.png|frame_
  Duration: N/A, bitrate: N/A
    Stream #0:0: Video: png, rgba(pc), 812x612 [SAR 3937:3937 DAR 203:153], 1 fps, 1 tbr, 1 tbn, 1
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> vp9 (libvpx-vp9))
Press [q] to stop, [?] for help
[libvpx-vp9 @ 0x14e4580] v1.8.0
Output #0, webm, to 'output.webm':
  Metadata:
    encoder         : Lavf58.20.100
    Stream #0:0: Video: vp9 (libvpx-vp9), yuva420p, 812x612 [SAR 1:1 DAR 203:153], q=-1--1, 200 kb/
    Metadata:
      encoder         : Lavc58.35.100 libvpx-vp9
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: -1
frame=    9 fps=0.0 q=0.0 Lsize=      13kB time=00:00:08.00 bitrate=  13.2kbits/s speed=17.4x
video:12kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 11.627710%
>>> Test finished successfully
================================================================
SUCCESS! All 1 out of 1 test scripts succeeded!
Cleaning up temporary directory
```

In order to avoid viash deleting the directory when a test succeeds, the `-k` option can be used.

Testing a namespace

In the previous examples we tested individual components, but we can test a suite of components as well. Since we stored the 2 components above in the (namespace) `civ6_save_renderer` again, we can do the following:

```
> viash ns test -p docker --parallel --tsv /tmp/report.tsv
          namespace        functionality             platform           test_name exit_code durat
  civ6_save_renderer        combine_plots               docker               start
  civ6_save_renderer         convert_plot               docker               start
  civ6_save_renderer         convert_plot               docker     build_executable         0
  civ6_save_renderer         convert_plot               docker          run_test.sh         0
  civ6_save_renderer        combine_plots               docker     build_executable         0
  civ6_save_renderer        combine_plots               docker          run_test.sh         0
```

With the `--parallel` option multiple tests are run in parallel (depending on your setup and the way Docker is configured).

The contents of (the optional) `report.tsv` contains a report of the test run:

namespace	functionality	platform	test_name	exit_code	duration	resul
civ6_save_renderer	convert_plot	docker	build_executable	0	0	SUCC
civ6_save_renderer	convert_plot	docker	run_test.sh	0	4	SUCC
civ6_save_renderer	combine_plots	docker	build_executable	0	1	SUCC
civ6_save_renderer	combine_plots	docker	run_test.sh	0	4	SUCC

For each component, you see the 2 steps from above: 1) build the executable and 2) run the actual test.

It should be noted that the tests are still running in their respective containers.