Contents

Creating components

With the information from the previous section, we will tackle two from the components in detail in this section:

- `convert_plot`
- `combine_plots`

Both are explained in [section 1] above.

convert_plot

`convert_plot` should convert a PDF map into a `.png` version. [ImageMagick] is a suite of command line tools for UNIX-like systems that can achieve this simply by running

```
convert input.pdf -flatten output.png
```

Additional arguments can be provided, but are not required since [ImageMagick] is pretty good at getting the defaults right. [ImageMagick] will probably not be on everyone's machine as a locally installed tool, however. We would to enable the conversion from pdf to png in a seamless way. Let's use viash for this…

The viash configuration

First of all, we will store all files related to one *component* in a separate directory and give it the name of the component:

```
> ls src/convert_plot
config.vsh.yaml
script.sh
```

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/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
platforms:
  - type: docker
    image: dpokidov/imagemagick
  - type: native
```

src/convert_plot/script.sh:

```
#!/bin/bash

convert "$par_input" -flatten "$par_output"
```

Let us dissect these two files step by step.

Arguments

The script is not so much different from the CLI example we gave above. The only difference is that 2 variables are used: `$par_input` and `$par_output`. We use double quotes around the variables, this is a good policy in general.

The argument `--input` defined in the config is automatically associated with `$par_input` and likewise for `--output`. This makes it easy to write scripts and immediately get a command-line parser for free when using viash.

If the script is more complicated than just this one instruction (it usually is), it is possible to set default values for those parameters in the script itself. This way, the script can be developed on its own without requiring viash directly. This can be achieved by including the following code block at the top of the file. This syntax is similar but slightly different depending on the scripting lanuage used.

```
## VIASH START
par_input=input.pdf
par_output=output.png
## VIASH END
```

If we focus on `--input` for a second, we notice the following attributes:

- `-i` is a (short) alternative for the longer `--input`
- The value for this argument is of type `file` which means it's either a file or a directory.
- With `required: true` we make this argument a mandatory one
- The default value for the argument is `input.pdf`
- For argument of type file like this one, we can ask viash to check if the file/directory exists prior to running.
- The `description` attribute contains a human-readable description of this argument/parameter.

Similar attributes can be found for `--output` with one difference:

- `direction: output` denotes that this argument denotes an output file/option.

In fact, `--input` also has a (hidden) `direction: input` associated to it by default.

Resources

We've covered how to specify resources earlier in the previous section. Suffice to say here that we point to a bash script that contains the actual command-line instruction.

Platforms

Two platforms are defined in the present case: a Docker one and a native one. We point the Docker platform to an existing Docker image available on Docker Hub.

Building the executable

Building an executable can be done just like before. We assume ImageMagick is not installed on the local system and thus build the Docker version:

```
> viash build src/convert_plot/config.vsh.yaml -o bin/ -p docker
```

We specify the `docker` platform explicitly although that is not really necessary because of the order of the platforms in the viash config. The resulting script is stored under `bin` relative to the current working directory.

We ask the generated executable to run the necessary setup. In this case, it means *pulling* the appropriate docker image from Docker Hub.

```
> bin/convert_plot ---setup
> docker pull dpokidov/imagemagick
Using default tag: latest
latest: Pulling from dpokidov/imagemagick
```

```
Digest: sha256:6749db04ffa5eac1cbe77566af02463f040028fef525b767dc98e06023e6cdf8
Status: Image is up to date for dpokidov/imagemagick:latest
docker.io/dpokidov/imagemagick:latest
```

We can retrieve the *help*

```
> bin/convert_plot -h
Convert a plot from pdf to png.

Options:
    -i file, --input=file
        type: file, required parameter, default: input.pdf
        A PDF input file.

    -o file, --output=file
        type: file, required parameter, default: output.png
        Output path.
```

Running the executable

Now that everything is up and running, we can start converting images. Let us first generate a simple PDF file with the help of viash. We start by preparing a directory to store the data:

```
> mkdir -p data/
```

And then use a simple mechanism to create a very basis PDF from the viash help output:

```
> viash -h | groff -mom -T pdf > data/viash.pdf
```

This generates `data/viash.pdf` in order to verify if our conversion step works:

```
> bin/convert_plot -i data/viash.pdf -o data/viash.png
convert: profile 'icc': 'RGB ': RGB color space not permitted on grayscale PNG `/viash_automount<.
```

Ok, so this should work now:

Without viash

Please note that in the above example, the input file and output file reside on the host image while the conversion process is running inside a Docker container. If we would want to achieve this without viash, we would need something like this:

```
docker run -i -v `pwd`:/mount dpokidov/ImageMagick /mount/data/viash.pdf -flatten /mount/data/viash
```

This requires some mental bookkeeping to understand the difference between the host's file system and the one inside the container. It also requires one to know how the container's commands are parsed. In this

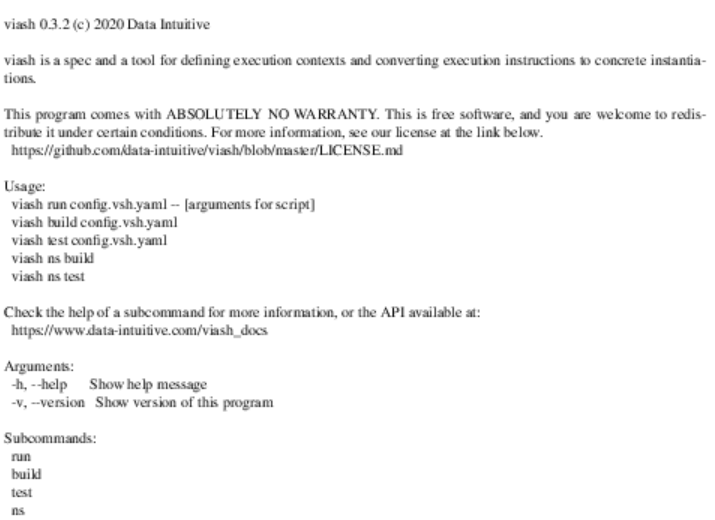


Figure 1: PNG from PDF using convert_plot

case the `convert` command from ImageMagick is automatically called with the options we provide. But that may be different for every container and depends on the contents of the `Dockerfile`.

Also, while the above explicit `docker` command achieves our aim, it does not fully cover the use-case that we tackle using viash. For a correct comparison, we would have to run our custom script in the container. But then, we would have to make a few updates:

1. Include command-line argument parsing in the `script.sh` file so that we can provide input and output parameters to it.
2. *Install* the modified `script.sh` file inside the container, or somehow *mount* the location of the executable inside the container such that it can be found.

In other words:

> Using viash all this is greatly simplified and wrapped in one executable, command-line parsing comes for free.

combine_plots

This *component* combines a number of `png` files in to one single movie (`webm` format).

The viash configuration

First of all, we will store all files related to one *component* in a separate directory and give it the name of the component:

```
> ls src/combine_plots
config.vsh.yaml
script.sh
```

Again, there is a viash config (`config.vsh.yaml`) and a script (`script.sh`). Let us take a closer look at both of these:

`src/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
platforms:
  - type: docker
    image: jrottenberg/ffmpeg
  - type: native
```

src/combine_plots/script.sh:

```
#!/bin/bash

inputs=$(echo $par_input | tr ':' '|')
ffmpeg -framerate $par_framerate -i "concat:$inputs" -c:v libvpx-vp9 -pix_fmt yuva420p -y "$par_out
```

Arguments

This component is similar to the one above with one major difference: we need to specify *multiple* input file names. This can easily be done with viash by specifying `multiple: true` in the configuration of the `--input` argument. By default viash will pass the value of this option to the wrapped script depending on the type of script. In the case of `bash`, this is simply a string with a delimiter for the individual values (`:` by default). For `Python` and `R` etc., it is passed as a simple collection (list, array).

The script that is run converts the following value `--input`:

```
path1:path2:path3
```

into

```
-i path1 -i path2 -o path3
```

by means of the `sed` instruction.

The rest is only a matter of getting the command line parameters for `ffmpeg` right.

Platforms

Two platforms are again defined in the present case: a Docker one and a native one. We point the Docker platform to an existing Docker image available on Docker Hub.

Running the executable

Let us create one additional `png` file by using the same *technique* as before, this time creating a PDF file with just the output of an empty viash run.

```
> viash | groff -mom -T pdf > data/viash1.pdf
```

This generates `data/viash.pdf` in order to verify if our conversion step works:

```
> bin/convert_plot -i data/viash1.pdf -o data/viash1.png
convert: profile 'icc': 'RGB ': RGB color space not permitted on grayscale PNG `/viash_automount<.
```

Ok, so this should work now:

We now have 2 `png` files and should be able to run our `combine_plots` component. But first, we'll build the executable:

```
> viash build src/combine_plots/config.vsh.yaml -o bin/ -p docker
```

Make sure the container is present:

```
> bin/combine_plots ---setup
> 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
```

And than executing the executable:

```
> bin/combine_plots --input data/viash.png:data/viash1.png --output data/output.webm --framerate 1
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:/viash_automount<...>/workspace/di/viash_workshop_1/130-CreatingC
```

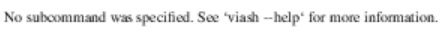
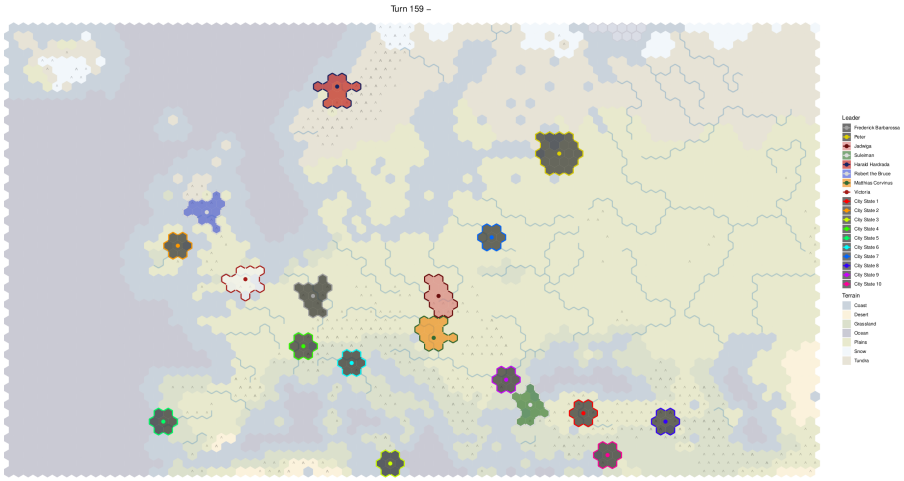


Figure 2: PNG from PDF using convert_plot

```
  Duration: N/A, bitrate: N/A
    Stream #0:0: Video: png, gray(pc), 612x792 [SAR 72:72 DAR 17:22], 1 tbr, 1 tbn, 1 tbc
Stream mapping:
  Stream #0:0 -> #0:0 (png (native) -> vp9 (libvpx-vp9))
Press [q] to stop, [?] for help
[libvpx-vp9 @ 0x1aa51c0] v1.8.0
Output #0, webm, to '/viash_automount<...>/workspace/di/viash_workshop_1/130-CreatingComponents/dat
  Metadata:
    encoder         : Lavf58.20.100
    Stream #0:0: Video: vp9 (libvpx-vp9), yuva420p, 612x792 [SAR 1:1 DAR 17:22], q=-1--1, 200 kb/s,
    Metadata:
      encoder         : Lavc58.35.100 libvpx-vp9
    Side data:
      cpb: bitrate max/min/avg: 0/0/0 buffer size: 0 vbv_delay: -1
frame=    2 fps=0.0 q=0.0 Lsize=      19kB time=00:00:01.00 bitrate= 151.7kbits/s speed=4.78x
video:18kB audio:0kB subtitle:0kB other streams:0kB global headers:0kB muxing overhead: 4.843163%
```

The result this simple *video* that can hardly be called a video:



In the next section, we will cover *all* the components of the postgame pipeline one by one.