wBuild Documentation

Release 1.8.0

Leonhard Wachutka

Nov 07, 2020

Contents

Contents:

CHAPTER 1

wBuild

1.1 Overview

wBuild is all about making your day easier resolving, updating and cascading various dependencies, pipeline rules and code structs. The program lets you specify all the needed information in a YAML header right in your R code and let the automated `Snakemake` processes do the rest!

You can learn more about the features that wBuild provides either taking a look at the *features list* or *looking at the HTML output of the demo project*. Another interesting thing to take a look at could be the *installation requirements & procedure* and, in particular, *wBuild project tree structure*.

You can find functionality overview of wBuild and its relationship with Snakemake *here*.

1.2 Example

First, we *install* all the needed software requirements, including wBuild. Then, we *initialize wbuild* which creates *wBuild files* in our project.

After that, we create an R script in the and provide a *YAML header* with wBuild-supported tags:

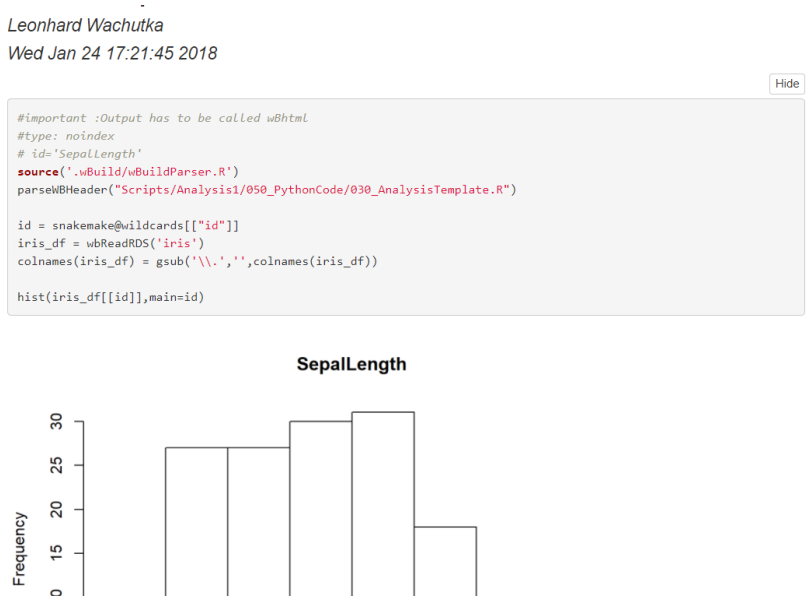
```
#'---
#' title: Basic Input Demo
#' author: Leonhard Wachutka
#' wb:
#'  input:
#'  - iris: "Data/{wbP}/iris.RDS"
#'  output:
#'  - wBhtml: "Output/html/030_AnalysisOfId_{id}.html"
#'  type: noindex
#' output:
#'  html_document:
#'   code_folding: show
#'   code_download: TRUE
#'---

source('.wBuild/wBuildParser.R')
parseWBHeader("Scripts/Analysis1/050_PythonCode/030_AnalysisTemplate.R")

id = snakemake@wildcards[["id"]]
iris_df = wbReadRDS('iris')
```

```
colnames(iris_df) = gsub('\\.','',colnames(iris_df))
hist(iris_df[[id]],main=id)
```

Running `snakemake` now in the root directory of your project will now automatically parse the parameters out of scripts headers and create an HTML output showing the results of our petal analysis - found in HTML output directory (`./Output/html` by default) along with a nice navigable HTML structure. Let's open one of the output HTML files, use the upper footer to navigate to the *needed subproject* (here *Analysis1*), and we will see a nicely rendered output of our script:



Leonhard Wachutka
Wed Jan 24 17:21:45 2018

Hide

```
#important :Output has to be called wBhtml
#type: noindex
# id='SepalLength'
source('.wBuild/wBuildParser.R')
parseWBHeader("Scripts/Analysis1/050_PythonCode/030_AnalysisTemplate.R")

id = snakemake@wildcards[["id"]]
iris_df = wbReadRDS('iris')
colnames(iris_df) = gsub('\\.','',colnames(iris_df))

hist(iris_df[[id]],main=id)
```


You can read more about *publishing the output HTML to your common server* or try launching demonstration yourself as follows:

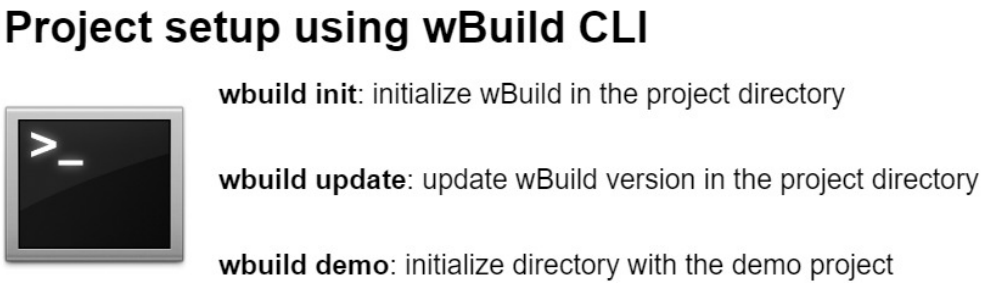
1.2.1 Running demo

- Install wBuild. You can learn more about the installation process *here*.
- Navigate to an *empty* directory.
- Run `wbuild demo`. This will create a wBuild demo project with various examples.
- Explore the files in `Scripts/`
- Run `snakemake` on the root directory to let Snakemake do its thing (see below) and compile the project.
- Open `Output/html/index.html` in your web browser. From there, you can browse through sites showing and describing *basic features* of wBuild on an example analysis.

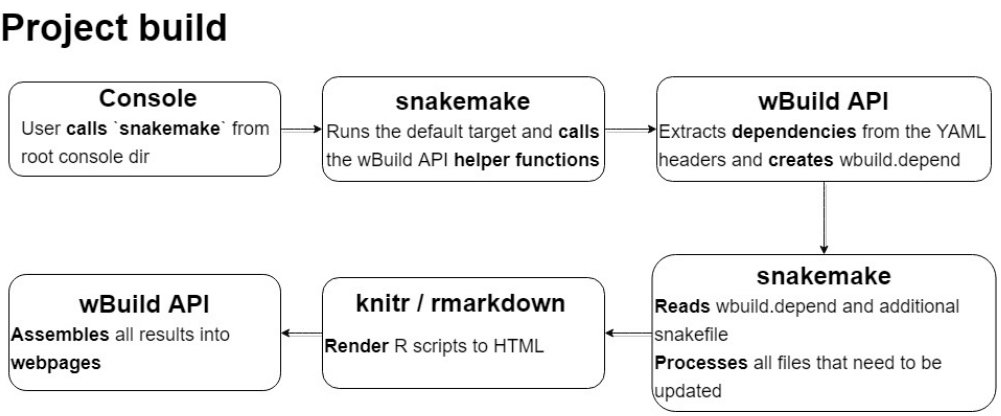
1.3 Functionality & workflow

wBuild is *not really a standalone application*, much more **a plugin and "code generator" for the later use of Snakemake**, which is *inevitable* part of a workflow involving wBuild: this way, you run `snakemake` CLI each time you want to build and render your project!

First, wBuild executes the initial, *setup* part of the workflow:



After that comes the project build phase. During this step, snakemake with a help of wBuild does process your project and creates the results. Following diagram shows the process:



As you see, *Snakemake* actually takes the **main** role in a typical wBuild workflow, so every user is *encouraged* to learn more about Snakemake - for instance in its official documentation. You are also welcome to take a look at the more *technial features* that wBuild provides.

A small overview of the functionality that wBuild provides:

- wBuild enables reproducible research by appending every R-markdown script to the global analysis pipeline written in snakemake
- All R scripts using R-markdown are compiled via knitr/Rmarkdown and rendered in a navigable web-page
- This is achieved by writing the snakemake rules directly in the header of your R scripts

- Headers allow the same flexibility (i.e. usage of python) as in the traditional Snakefile, but do not separate dependencies from the code where they are actually used.

1.4 wBuild project structure

Assert `ROOT` is the root directory of your wBuild project.

ROOT/.wBuild Is a directory with static wBuild files that *is not to be changed unless necessary*. There are all the service files located.

ROOT/wbuild.yaml It is a configuration file. Necessary but editable by the user. See *configuration file*.

ROOT/.wBuild.depend File *autogenerated* by wBuild. Contains rule information for the *Snakemake pipeline*.

ROOT/Scripts Is a directory where all your scripts should be located within **subdirectories**. Organizing your code directly into subdirectories helps you to flexibly structure your project, to present them divided by tabs in the HTML output or to *apply the same script to the various equally structured data*

ROOT/Output Is a default output directory. It's subdirectories, `html` and `ProcessedData`, are paths for *published html* and processed data by default (*you can change it*).

CHAPTER 2

Installation

2.1 Requirements

2.1.1 1. pip

`pip` is a Python package manager that makes it much easier to download and install Python packages, as a part of such `wbuild` and `snakemake`. If you don't have pip installed, this Python installation guide can guide you through the process.

Advanced users who want to get the Python packages elsehow can skip this step.

2.1.2 2. snakemake

Snakemake is essential to wBuild workflow (learn why in the *Overview of functionality*). You can get Snakemake either using `pip` or building from sources. By *installing wBuild from pip* you automatically install the Snakemake, should it be absent.

2.1.3 3. R and packages

The original purpose of wBuild's work is to let you put additional build/dependencies info in your R scripts, so we suppose you have already installed R. Now, the very important step for installation is to install various **R packages**:

- *knitr*
- *rmarkdown*
- *pandoc*

as Snakemake **inevitably uses them** while working with wBuild.

You can install packages for R with `install.packages(packagename)` directive. Find out more for instance here.

2.1.4 Additionally for *Windows* users

You need to add `R_HOME` and pandoc home variables to your Path for Snakemake CLI to run correctly. See p.2 in this instruction. Now, you are all ready and set up to install the *wBuild* itself.

2.2 Stable release

Please make sure you've read *Requirements* first.

To install wBuild, run this command in your terminal:

```
$ pip install wbuild
```

2.3 From sources

The sources for wBuild can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/wachutka/wbuild
```

Or download the tarball:

```
$ curl  -OL https://github.com/wachutka/wbuild/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Features

3.1 Overview

We all know that doing data analysis day-to-day could easily turn into routine work and it is often hard to have fully reproducible code. Can you say for sure that you can redo your whole analysis only provided the raw data and your code? wBuild is designed to reduce the amount of time you spend to *publish the output of your script*, *declare the needed input files*, *run Py code as a part of work pipeline*, *use placeholders to structure your Snakemake job*, *map your project's scripts together* and many more.

3.2 Demo project

It is highly recommended to see all of the *examples* of using the features *in the demo project*. There you also have additional documentation that explains the features and working with them!

3.3 Command-line interface

The command-line interface of wBuild is responsible *only* for preparing a project directory to be processed by snakemake and wBuild. There are three instructions, also shortly documented under `wbuild -h`

`wbuild demo` Run *demo project*.

`wbuild init` Initialize *wBuild* in an already existing project. This command prepares all important wrappers and files for Snakemake.

`wbuild update` To be called on an already initialized project. Updates `.wbuild` directory to the latest version using *installed* Python `wbuild` package.

All these commands should be executed from the **root directory of the project**.

3.3.1 Snakemake CLI

Most of the job of building your project is done by Snakemake, *as explained here*. There are also several special Snakemake rules that wBuild provides. The most important include:

snakemake mapScripts Do *script mapping*

snakemake publish Publish your html output pages to your *projectWebDir*

snakemake clean Deletes html output, generated dependencies file and Python cache.

snakemake restoreModDate Restore previous modification date of all the files. Comes handy for pulling changes from VCS, where all the mod.dates get changed.

See more about this down the page.

3.4 Parsing YAML headers

In following, we present a basic YAML header:

```
#'---
#' title: Basic Input Demo
#' author: Leonhard Wachutka
#' wb:
#'  input:
#'  - iris: "Data/{wbP}/iris.RDS"
#'  output:
#'  - pca: " {wbPD_P}/pca.RDS"
#' type: script
#'---
```

wBuild requires users to define information of the scripts in RMarkdown YAML-format header. wBuild scans it and outputs rules for Snakemake. `wb` block is a "wBuild-own" one. Important tags here are input and output. These are used to *costruct the snakemake pipeline*, and *render the script into an HTML format*.

Tags that can be provided mainly follow the logic of Snakemake and partially that of wbuild.

Please note: YAML tags have a strict format that they should follow - e.g. there should be *no tabs*, **only spaces!** You can read more about the YAML syntax.

If you want to access information from the header of a script from within the script (code self-reflection), need to **source** `.wBuild\wBuildParser.R` and **call** `parseWBHeader()` with the path to your script as an argument.

3.4.1 Tags

To make working with R projects even more comfortable, there are a few additional YAML tags that wBuild provides. They are:

input Specify any input files you would like to use. You can later access them from the R code using `snakemake@input[[<input_file_var>]]`.

output The same as input - accessed using `snakemake@output`.

py This tag allows you to run some Python code during parsing of the header - a good example of how this feature can be extremely helpful is in the *demo*. Don't forget the **YAML pipe operator** for the proper functionality!

type Tag describing the type of the file. Can be: `script` for R Scripts, `noindex` for Markdown and `empty` for the rest.

The information stated under this tags is later synchronised with Snakemake. One can also state Snakemake options in "wb" block of the YAML header and even *refer to them in this R script later* using `snakemake@`. Here, we mark that we will use 10 threads when executing this script:

```
#' wb:
#'  input:
#'  - iris: "Data/iris_downloaded.data"
#'  threads: 10
```

The specified thread variable can then be refered to by name in our R script: `snakemake@threads`

3.5 Snakemake special features

Use following addenda to `snakemake` CLI:

--dag Construct the directed acyclic graph of the current snakemake workflow and display as svg.

There are also some special rules that are not getting executed as a part of the usual workflow which can be run separately. Consult `.wBuild/wBuild.snakefile` in your project to find out more.

3.6 Publishing the output

Snakemake renders your project, including script text and their outputs, to a nice viewable *structure of HTML files*. You can specify the output path by putting/changing the htmlOutputPath value inside the *configuration* file found in the root directory of your wBuild-initiated project. Your HTML gets output to `Output/html` by default.

There is also a way to automatically **fetch your output to a webserver**: typing `snakemake publish` copies the whole HTML output directory to the directory specified in **projectWebDir** parameter in the *configuration file*.

3.7 Markdown

No need to create a separate Markdown file to describe the analysis - with wBuild you can do it right in your render output using `#'` at the beginning of the line, an then just usual MD syntax!

3.8 Configuration file

`wbuild.yaml` file that is found in the root directory of the project stands for the configuration file of wBuild. In this file you can adjust various properties of wBuild workflow:

htmlOutputPath This value specifies the *relative* path where your HTML output will land. *More precisely*, it is a *prefix to output file* of any Snakemake rule that is generated by wBuild. Default is `Output/html`.

processedDataPath *Relative* path to the data output directory. Default is `Output/ProcessedData`

scriptsPath *Relative* path to the root Scripts directory.

projectWebDir Path to the output directory for `snakemake publish`.

IMPORTANT: Please, do not remove any key-value pairs from it or move this file *unless you know what you are doing*.

3.9 Placeholders

Placeholders provide the ability to refer to your current position in your system's filepath with a pair of letters instead of absolute, relative paths. It's best shown in an example:

```
#' wb:
#'  input:
#'  - iris: "Data/{wbP}/iris.RDS"
#'  output:
#'  - pca: " {wbPD_P}/pca.RDS"
```

Here, we use `wbP` for the name of the current project (say, Analysis01) and `wbPD_P` for the name of the output directory for processed data slash project name, say `Output/ProcessedData/Analysis01`.

Here is the conscise list of the placeholders:

wbPD <output directory for processed data>, e.g. `Output/ProcessedData`

wbP <current project>, e.g. `Analysis1`

wbPP <subfolder name>, e.g. `020_InputOutput`

wbPD_P <output directory for processed data>/<current project>, e.g. `Output/ProcessedData/Analysis1`

wbPD_PP <output directory for processed data>/<current project>/<subfolder name>, e.g. `Output/ProcessedData/Analysis1/020_InputOutput`

3.10 Script mapping

This advanced feature allows you to use the same script to analyse the similarly structured data as a part of various subprojects.

It all begins with a configure file `scriptsMapping.wb` in the root directory of your project. There, you put a YAML *list of* YAML formatted **dictionaries** with two keys:

src A **YAML list** of *file* paths to create links from.

dst A **YAML list** of **directories** paths to put file links *into*.

Running `snakemake mapScripts` then creates symbolic links for *all the 'src' files* in any of *'dst' directories*.

Below is an example of a proper `scriptsMapping.wb` file:

```
- src:
  - _Template/preprocessData.R
  - _Template/PCAoutliers.R
  dst:
  - Principal_Analysis/allIntensities
  - Principal_Analysis/withoutFamilies
  - Principal_Analysis/withoutReplicates
  - Principal_Analysis/withoutReplicatesAndFamilies
```

Here, we map two scripts, `preprocessData.R` and `PCAoutliers.R`, to be in each of the four projects of `Principal_Analysis`. *Placeholders* then do their thing to speak to the right `ProcessedData` sub-directories, based on the current subproject.

3.11 HTML Subindex

For subdirectories under the `Scripts/` directory you can also create a separate HTML index file. This is particularly useful when you have a larger, more modular workflow and you want to view the results of one module as soon as they have successfully finished.

In order to create a subindex, you need to create a new rule in your `Snakefile`.

Note: The subdirectory path has to be within the script directory so that all HTML pages get rendered correctly.

Here is an example from the Demo project.

```
from wbuild.createIndex import createIndexRule, ci

subdir = "Scripts/Analysis1/010_BasicInput/"
index_name = "Analysis1_BasicInput"
input, index_file, graph_file, _ = createIndexRule(scriptsPath=subdir, index_
↪name=index_name)

rule subIndex:
    input: input
    output:
        index = index_file,
        graph = graph_file
    run:
        # 1. create the index file
        ci(subdir, index_name)
        # 2. create the dependency graph
        shell("snakemake --rulegraph {output.index} | dot -Tsvg -Grankdir=LR >
↪{output.graph}")
```

The `wbuild.createIndex.createIndexRule()` function takes in the relative subdirectory path and an index name, which is prepended to the index HTML file. In this example, the HTML index file is called `Analysis1_BasicInput_index.html` under the `htmlOutputPath`. The function returns a list of all HTML output files, the index file name, the dependency graph file name and the readme HTML file name.

Using this information, you can assemble your rule, where the HTML file list is the input and the output is the index file name. You need to call the `wbuild.createIndex.ci()` function to write the index HTML file. You should also include the instructions to generate your dependency graph file. The standard way is to use the snakemake option `--rulegraph` to create a graph of all dependencies of the index file. This gives you a `graphviz` output that you can pipe into an the dependency graph file that you obtained from `wbuild.createIndex.createIndexRule()`. Optionally, you can also use the `--dag` option, which gives you the complete job graph.

CHAPTER 4

Contributing

4.1 Welcome!

Thanks for your interest in the technical side of our project. Contributions to wBuild are a great way to level up your skill, tackle the problem you are facing with the program faster and receive some nice and fun Open Source experience! Last but not least, contributions are very welcome from our side, and they are greatly appreciated! You'll help anybody using wBuild, and we'll surely give a credit for you in *contributors*.

There are several ways to contribute:

4.2 Ways of contributing to wBuild

4.2.1 Bug reports

Report bugs at https://github.com/gagneurlab/wBuild/issues.

If you are reporting a bug, please include:

```
#### Environment

    Your operating system, version of wBuild, version of Snakemake; any further␣
↪details of your particular local setup
    that could be relevant

#### Issue description

    Generally describe the issue.

#### Steps to reproduce the issue

    Describe what did you do in the context of program before the bug came out.
    For example:
        1. Initiate wBuild in project
        2. Remove wBuild.depend
        3. Launch snakemake publish rule
        ....

#### What's the expected result?

    Describe the result of your actions that you have expected.
```

```
#### What's the actual result?

    Describe the result of your actions that you have faced.


#### Additional details / screenshot

    Include any additional details that you consider relevant.

- ![Screenshot]()
-
```

4.2.2 Bug fixes

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open for your work. Initiative bug fixes are also *highly welcome!*

4.2.3 Implement features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

4.2.4 Write documentation

wBuild could always use more documentation, whether as part of the official wBuild docs, in docstrings, or even on the web in blog posts, articles, and such.

4.2.5 Request/propose a feature

The best way to send feedback is to file an issue at https://github.com/gagneurlab/wBuild/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

4.3 Working with wBuild code

4.3.1 Prepare

Please make sure you've read the user *overview* to understand the basics of wBuild - *wBuild position in the Snakemake workflow*, *demo project* as well as *features list* could be especially interesting here.

4.3.2 Setting up the development environment

Ready to contribute? Here's how to set up *wbuild* for local development.

1. Fork the *wbuild* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/wbuild.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv wbuild
$ cd wbuild/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 wbuild tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

If the pull request adds functionality, we kindly ask you to also update the docs, telling about this new cool stuff! Put your new functionality into a function with a docstring, and add this feature to the list in README.rst.

4.4 Code documentation

The code of wBuild is well-documented, and it would be nice to keep it that way. Apart from looking in the code, here you find the documentation for the functions of wBuild:

4.4.1 CLI (`wbuild.cli`)

CLI interface to wbuild.

`wbuild.cli.`**`setup_paths`**`()`
Setup the wbuild paths

4.4.2 Files scanning (wbuild.scanFiles)

wbuild.scanFiles.**dumpSMRule**(*ruleInfos*, *outputFile*, *inputFile*)
Write the rule to the file.

Parameters

- **ruleInfos** – dictionary containing all the rule's data
- **outputFile** – file to print the rule to
- **inputFile** – the object of the rule

wbuild.scanFiles.**escapeSMString**(*item*)
Convert item to the appropriate string representation.

Parameters **item** – string or dict

Returns "key = 'value"' (dict), "'value"' (string) or '' (other type)

wbuild.scanFiles.**insertPlaceholders**(*dest*, *source*)
Infer placeholders' substitutions.

Parameters

- **dest** – string to replace placeholders in
- **source** – file; from its path we infer the placeholders values

Returns dest with replaced placeholders

wbuild.scanFiles.**joinEmpty**(*string_list*)

Parameters **string_list** –

Returns sting representation of a list without the blank elements.

wbuild.scanFiles.**writeDependencyFile**()
Entry point for writing .wBuild.depend.

wbuild.scanFiles.**writeIndexRule**(*wbData*, *mdData*, *file*, *ignoreMD=False*, *dump=False*)
Write the rule of mapping the R and md wbData to the index.html.

Parameters

- **wbRRows** – info dict parsed from R wB files
- **wbMDrows** – info dict parsed from MD wB files
- **file** – file to print the index rule to

wbuild.scanFiles.**writeMdRule**(*ruleInfos*, *file*)

Parameters

- **ruleInfos** –
- **file** – file to write the rule to

wbuild.scanFiles.**writeRule**(*r*, *file*, *dump=False*)
Write Snakemake rule from the parsed WB header informations.

Parameters

- **r** – parsed WB data dictionary entry
- **file** – to write the rule to

wbuild.scanFiles.**writeWBParseDependencyFile**(*filename*)
Entry point for writing .wBuild.depend. for the wbParseFunction in R

4.4.3 Service functions (wbuild.utils)

wbuild.utils.**checkFilename**(*filename*)

Parameters **filename** – to check

Returns has appropriate name?

Raises ValueError if the name is inappropriate

wbuild.utils.**findFilesRecursive**(*startingPath*, *patterns*)

Parameters

- **startingPath** – root path of the search
- **patterns** – patterns to search file names for

Returns paths to files matching the patterns

wbuild.utils.**hasYAMLHeader**(*filepath*)

Parameters **filepath** – path to the file

Returns file contains YAML header?

wbuild.utils.**linuxify**(*winSepStr*, *doubleBackslash=False*)
Convert windows (path) string to the linux format.

Parameters

- **winSepStr** – (path) string with windows-like “” separators
- **doubleBackslash** – if the slashes in the winSepStr are double (happens when you read a macro string raw. Ex.: “C:Program Filesa.txt”

Returns str with substituted “” -> “/”

wbuild.utils.**parseMDFiles**(*script_dir='Scripts'*, *htmlPath='Output/html'*, *readmePath=None*)

Parameters

- **script_dir** – Relative path to the Scripts directory
- **htmlPath** – Relative path to the html output path
- **readmePath** – Relative path to the readme

Returns a list of dictionaries with fields: - file - what is the input .md file - outputFile - there to put the output html file - param - parsed yaml header - always an empty list

wbuild.utils.**parseWBInfosFromRFile**(*filename*, *htmlPath='Output/html'*)

Parameters

- **filename** – Relative path to the Scripts directory
- **htmlPath** – Relative path to the html output path

Returns a list of dictionaries with fields: - filen - what is the input R file - outputFile - there to put the output html file - param - parsed yaml params

wbuild.utils.**parseWBInfosFromRFiles**(*script_dir='Scripts'*, *htmlPath='Output/html'*)

Parameters

- **script_dir** – Relative path to the Scripts directory
- **htmlPath** – Relative path to the html output path

Returns a list of dictionaries with fields: - file - what is the input R file - outputFile - there to put the output html file - param - parsed yaml params

wbuild.utils.**parseYAMLHeader**(*filepath*)

Parameters **filepath** – path to the file

Returns String representation of the YAML header in the file, including inter-document framing ("—")

wbuild.utils.**parseYamlParams**(*header*, *f*)

Parameters

- **header** – String form of YAML header
- **f** – Filename of a file from where the header was parsed

Returns Parameters dictionary parsed from the header; None if parsing errors occured

wbuild.utils.**pathsepsToUnderscore**(*systemPath*, *dotsToUnderscore=False*, *trimPrefix=True*)
Convert all system path separators and dots to underscores. Product is used as a unique ID for rules in scanFiles.py or the output HTML files :param systemPath: path to convert in :param dotsToUnderscore: if the dot should be converted as well. Defaults to false :return: path string with converted separators

wbuild.utils.**wbuildVersionIsCurrent**()
Read wBuild version from .wBuild/.version and compare it to wbuild module version from pckg mngr. :return: True if wBuild up-to-date, False if not

wbuild.utils.**writeWbuildVersion**()
Write wBuild version to .wBuild/.version

4.4.4 HTML output index creation (wbuild.createIndex)

wbuild.createIndex.**ci**(*scriptsPath=None*, *index_name=None*)
Write HTML index file

Parameters

- **scriptsPath** – relative scripts path. If not specified, use the default scripts path from the config.
- **index_name** – prefix of the index file name <*index_name*>*_index.html*

wbuild.createIndex.**createIndexRule**(*scriptsPath=None*, *index_name=None*, *wbData=None*, *mdData=None*)
Create the input and output files necessary for an HTML index rule.

Parameters

- **scriptsPath** – relative scripts path. If not specified, use the default scripts path from the config.
- **index_name** – prefix of the index file name <*index_name*>*_index.html*
- **wbData** – wBuild rule data from parsed R scripts
- **mdData** – wBuild rule data from parsed markdown files

Returns

- **inputFiles** - list of output HTML files from the *scriptsPath*, comprising the input of the index rule
- **indexPath** - index html file name, equates to the output
- **graphPath** - dependency graph image file name for HTML template (graph is needs to be written to this file)
- **readmePath** - readme HTML name for HTML template (takes readme from the *scriptsPath*)

wbuild.createIndex.**getRecentMenu**()
Support recently edited files list to the HTML web output.

Returns HTML string: "Recently viewed" menu contents

wbuild.createIndex.**writeDepSVG**(*graphPath=None*)
Search for rule graph. If path not specified in config, take default dep.svg in snakeroot path

wbuild.createIndex.**writeIndexHTMLMenu**(*scriptsPath=None*, *index_name=None*)
Scan for files involved in the current HTML rendering and fill the HTML quick access toolbar correspondingly

wbuild.createIndex.**writeReadme**(*readmePath*)
Create readme file output for html template readmePath: html readme path

wbuild.createIndex.**writeSubMenu**(*top*, *wbData*, *level*)
Recursive call to construct the dropdown list and hover-over side-menus in it adhereing to a "top" toolbar category.

Parameters

- **top** – "top" toolbar directory to be appointed to
- **wbData** – wb relevant data of all scanned files
- **level** – deepness of the current submenu (first dropdown list, then hover-over side-menus in the html)

Returns deeply constructed dropdown list of the top toolbar category as an HTML string

4.4.5 Script mapping (wbuild.autolink)

See also *the overview of this feature*

CHAPTER 5

Frequently asked questions

Q: I've modified my project files and now I'm running snakemake, but it prints there's nothing to be done.

A: Probably your pipeline is broken. Snakemake is configured to run only if `all.done` file in the ProcessedData directory(`{wbPD}/..`) is out of date and the whole pipline can be run. Therefore, please remove it manually first if you want to run the whole pipeline again! You can also try to launch `snakemake Index -f` to force a recreation of the index page. Normally you will then be pointed to the place where your pipeline is broken.

Q: I want to remove a file from the pipeline:

A: Just move it into a folder starting with an underscore _.

Q:Can I use my input/output variables defined in header in the code afterwards?

A: Of course you can! See *tags section* for more information how.

Q:Hey, but I don't run snakemake now, and still would like to have something to debug!

A:You are not alone! See the bottom of *information about in-script headers* - there is a special callable for that!

Q:All the time I push/pull from my VCS (e.g. git), the modification times of the files get updated. Why, and how to avoid it?

A: (All) VCS work this way. Unluckily, there's a little we can do about it on our side, since algorithm of comparing timestamps in builds comes from Snakemake. But you can take care about it yourself relatively easily, e.g. using `touch -r` to restore the modification date of file(s). See e.g. https://stackoverflow.com/questions/2458042/restore-a-files-modification-time-in-git for more information. There is also an *in-built wBuild rule* that does it recursively to all the project files.

CHAPTER 6

Credits

6.1 Development Team

- Leonhard Wachutka <leonhard@wachutka.eu> - idea of the tool, architecture, code
- Stefan Dvoretskii <stefan.dvoretskii@gmail.com> - code, documentation

6.2 Contributors

- Jun Cheng <chengju@in.tum.de> - code, discussion
- Žiga Avsec <avsec@in.tum.de> - code, discussion
- Felix Brechtmann <Felix.Brechtmann@in.tum.de> - beta-testing, documentation

CHAPTER 7

History

7.1 0.1.0 (2017-06-23)

- First release on PyPI.

7.2 1.0 (2017-12-15)

- Improved Demo
- Many fixes

CHAPTER 8

Search the documentation

search

W

C

D

E

F

G

H

I

J

L

P

S

W