DrWatson.jl

Or how to organize your simulations better.

Meet the team

DrWatson.jl

Package

What is DrWatson?

In their own words, "DrWatson is a **scientific project assistant** software".

But what does this mean?

- High level project organization/navigation
- Parameter collection and distribution
- Safe result saving
- Easy and safe data aggregation

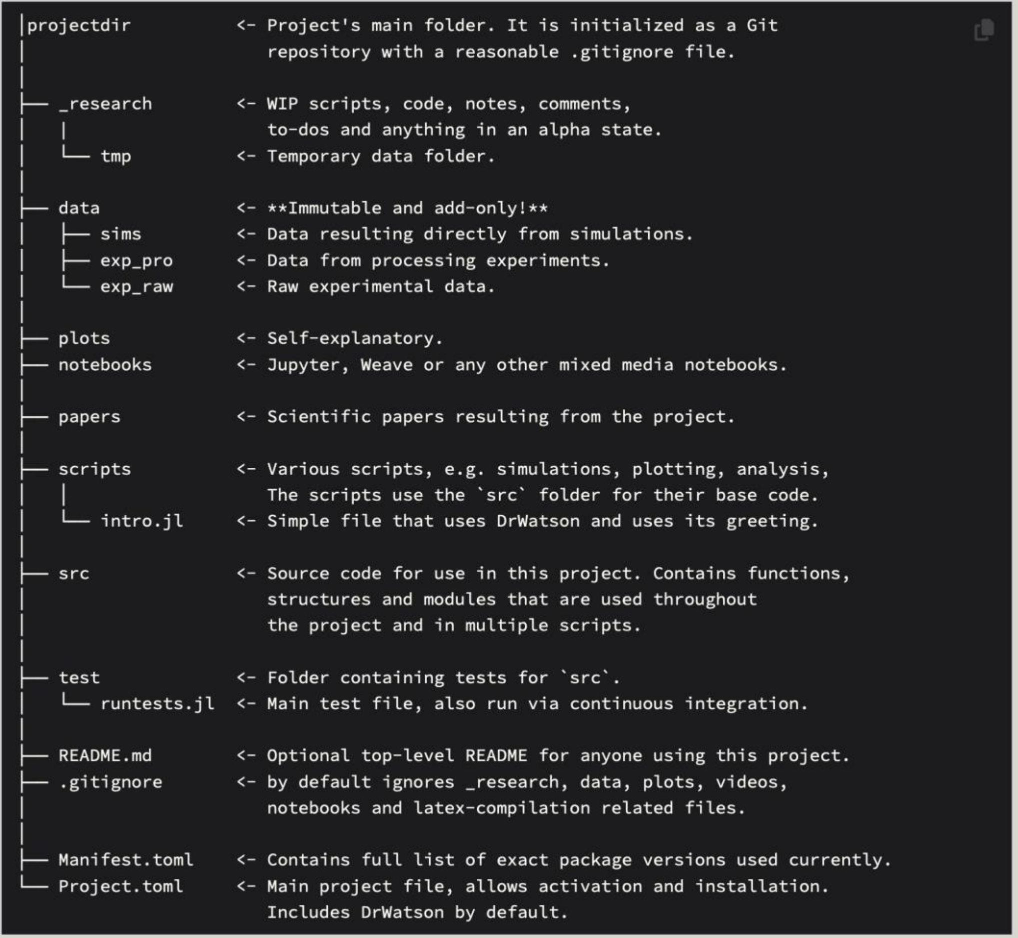
Project Organization

initialize_project

- Creates the folder layout for the project, we see the default here
- Can add Git/Test/Doc structures
- Does NOT add a special *project.jl* that can easily distribute source code

@quickactivate

- Activates the *Project.toml* for only this project
- Brings in source code for project into scope
- Allows for use of project file traversal functions



```
│projectdir          <- Project's main folder. It is initialized as a Git
│                       repository with a reasonable .gitignore file.
│
├── _research        <- WIP scripts, code, notes, comments,
│   │                   to-dos and anything in an alpha state.
│   └── tmp          <- Temporary data folder.
│
├── data             <- **Immutable and add-only!**
│   ├── sims         <- Data resulting directly from simulations.
│   ├── exp_pro      <- Data from processing experiments.
│   └── exp_raw      <- Raw experimental data.
│
├── plots            <- Self-explanatory.
├── notebooks        <- Jupyter, Weave or any other mixed media notebooks.
│
├── papers           <- Scientific papers resulting from the project.
│
├── scripts          <- Various scripts, e.g. simulations, plotting, analysis,
│   │                   The scripts use the `src` folder for their base code.
│   └── intro.jl     <- Simple file that uses DrWatson and uses its greeting.
│
├── src              <- Source code for use in this project. Contains functions,
│                       structures and modules that are used throughout
│                       the project and in multiple scripts.
│
├── test             <- Folder containing tests for `src`.
│   └── runtests.jl  <- Main test file, also run via continuous integration.
│
├── README.md        <- Optional top-level README for anyone using this project.
├── .gitignore       <- by default ignores _research, data, plots, videos,
│                       notebooks and latex-compilation related files.
│
├── Manifest.toml    <- Contains full list of exact package versions used currently.
└── Project.toml     <- Main project file, allows activation and installation.
                        Includes DrWatson by default.
```

Project Navigation

projectdir(args...)

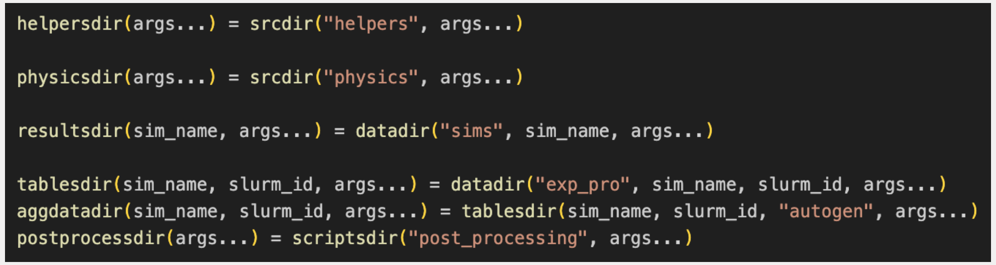
This function returns the directory of the currently activated project.

Extremely useful when you want to easily read files from other folders (which happens a lot)

datadir, srcdir, scriptsdir...

Derivatives of *projectdir* that return paths into default DrWatson folders

Good practice is to make your own *dir* functions by using *projectdir* to allow for easy/safe reading into certain folders.

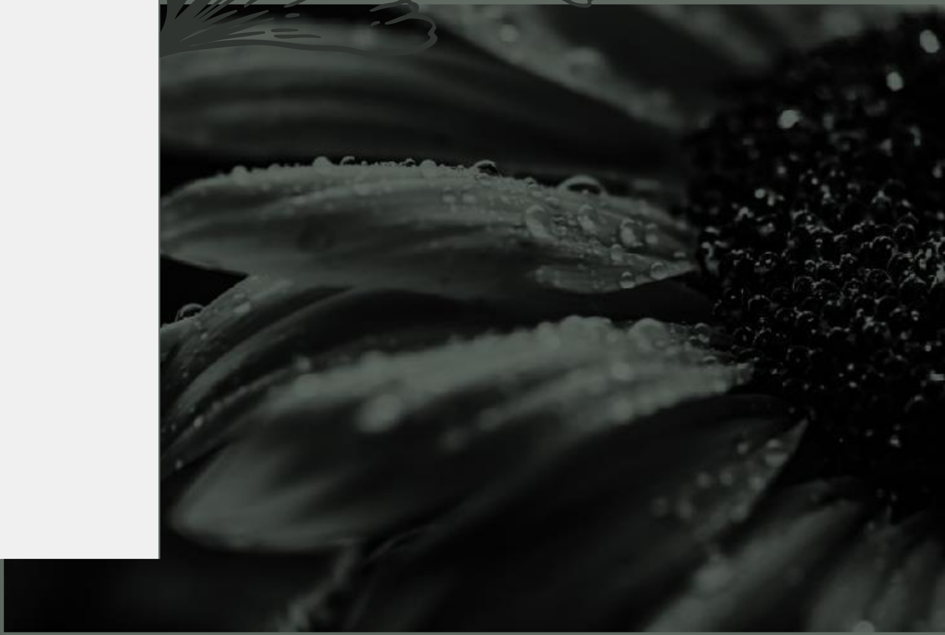


```
helpersdir(args...) = srcdir("helpers", args...)

physicsdir(args...) = srcdir("physics", args...)

resultsdir(sim_name, args...) = datadir("sims", sim_name, args...)

tablesdir(sim_name, slurm_id, args...) = datadir("exp_pro", sim_name, slurm_id, args...)
aggdatadir(sim_name, slurm_id, args...) = tablesdir(sim_name, slurm_id, "autogen", args...)
postprocessdir(args...) = scriptsdir("post_processing", args...)
```



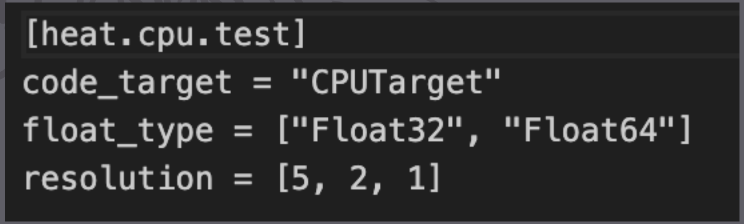
Parameter Collecting

@dict, @strdict, @ntuple

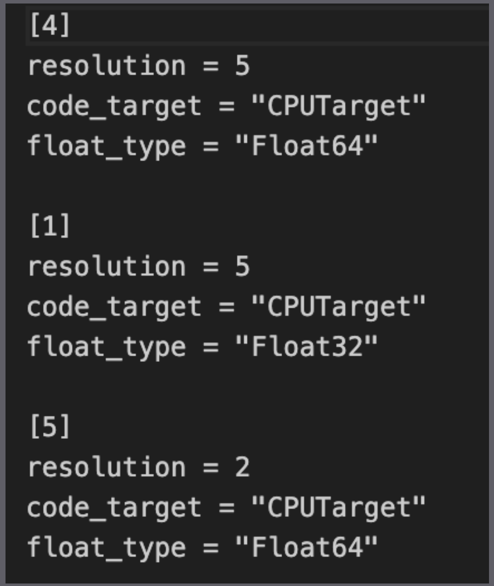
- All these functions can be used to collect variables into neat data structures
- There are also functions that can be used to convert between these types easily
- Even more, there are also functions to take structs to these above data types

@dict_list

- Wonderful function that can easily distribute different parameter values
- Takes a dictionary of single values or lists of values
- Returns a list of dictionaries with those lists expanded out



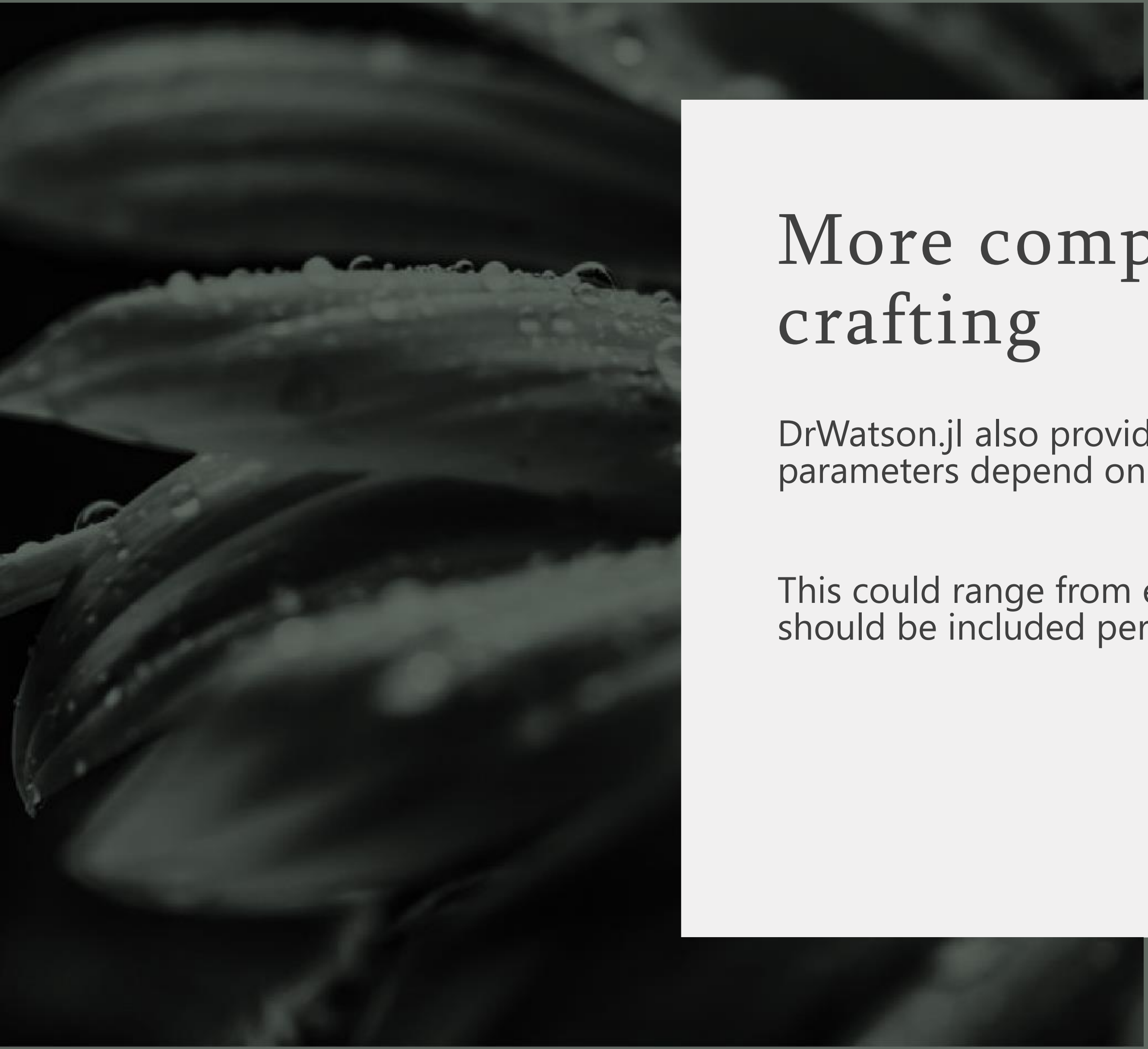
```
[heat.cpu.test]
code_target = "CPUTarget"
float_type = ["Float32", "Float64"]
resolution = [5, 2, 1]
```



```
[4]
resolution = 5
code_target = "CPUTarget"
float_type = "Float64"

[1]
resolution = 5
code_target = "CPUTarget"
float_type = "Float32"

[5]
resolution = 2
code_target = "CPUTarget"
float_type = "Float64"
```



More complicated parameter crafting

DrWatson.jl also provides the ability to have certain parameters depend on other parameters.

This could range from effecting the value to determining if it should be included period.

Safe Result Saving

wsave

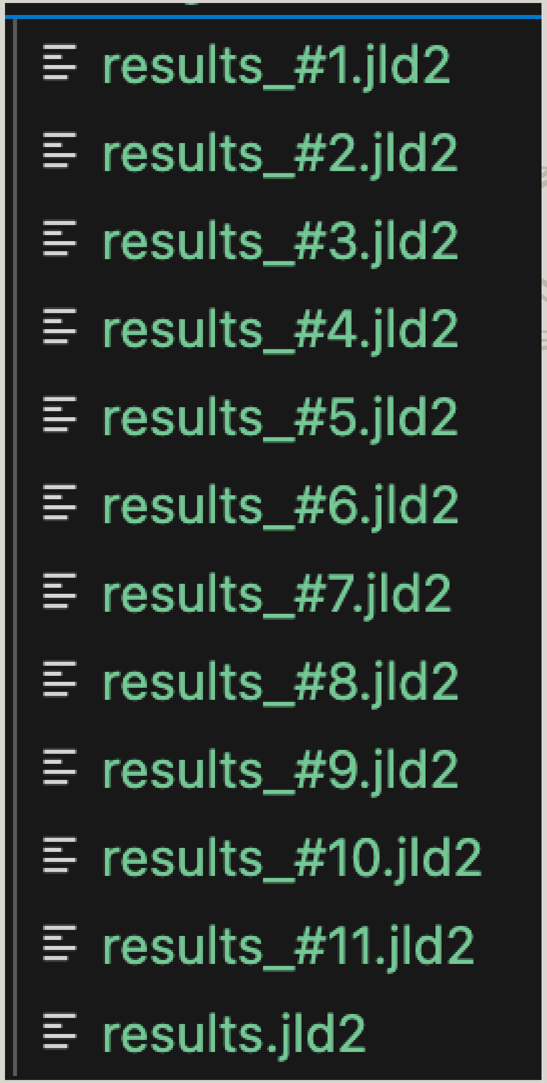
- Calls *mkpath* to ensure that intended save directory exists
- Works only for JLD2 files
- Good to prevent loss of results

safesave

- Prevents the deletion of files with the same name
- Works on top of *wsave*
- Great for preventing loss of results

tagsave

- Works just like *safesave* but also tags the file using Git



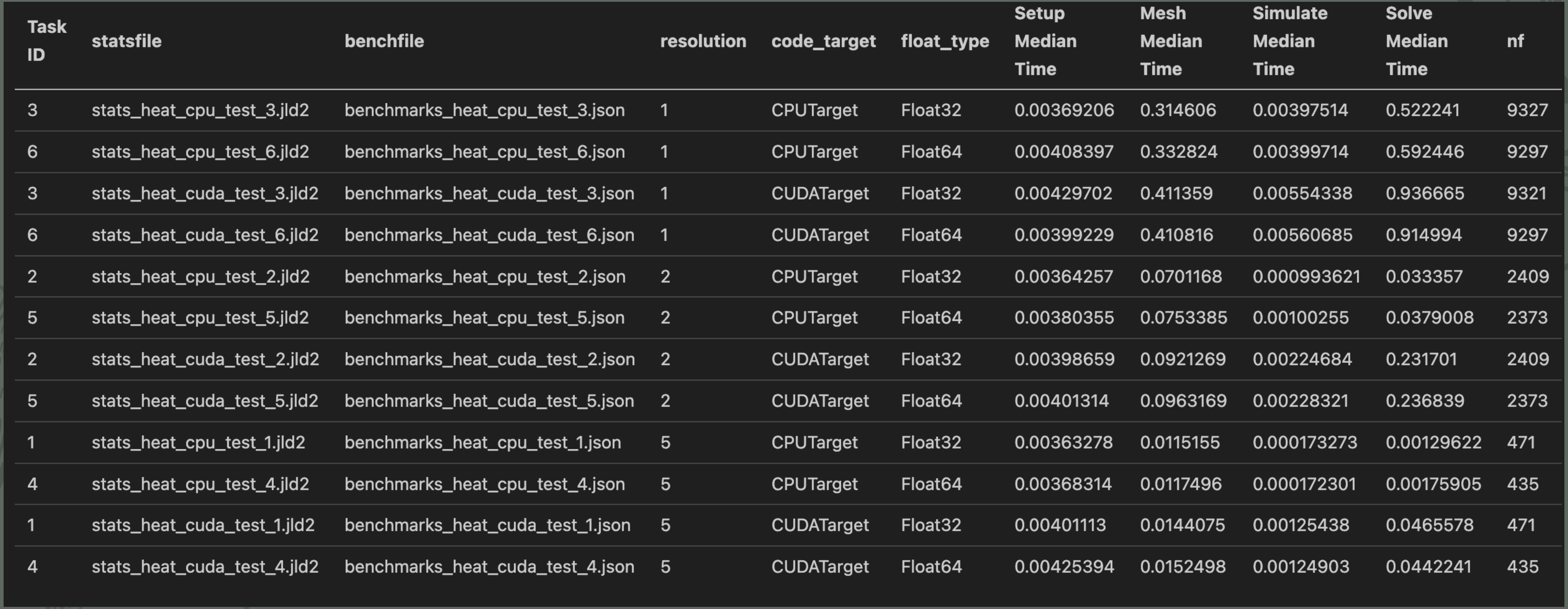
```
results_#1.jld2
results_#2.jld2
results_#3.jld2
results_#4.jld2
results_#5.jld2
results_#6.jld2
results_#7.jld2
results_#8.jld2
results_#9.jld2
results_#10.jld2
results_#11.jld2
results.jld2
```

Collecting Results

collect_results

- A powerful function that reads folders of result files and automatically collects that data into a single table
- Requires DataFrames but organizes data into a ready-to-go table
- If data in a result file is only partially filled, then *collect_results* will automatically fill-in the data as missing, instead of erroring
- Since this produces a DataFrame, a user/script can process that DataFrame to refine results

Example Markdown Output



Task ID	statsfile	benchfile	resolution	code_target	float_type	Setup Median Time	Mesh Median Time	Simulate Median Time	Solve Median Time	nf
3	stats_heat_cpu_test_3.jld2	benchmarks_heat_cpu_test_3.json	1	CPUTarget	Float32	0.00369206	0.314606	0.00397514	0.522241	9327
6	stats_heat_cpu_test_6.jld2	benchmarks_heat_cpu_test_6.json	1	CPUTarget	Float64	0.00408397	0.332824	0.00399714	0.592446	9297
3	stats_heat_cuda_test_3.jld2	benchmarks_heat_cuda_test_3.json	1	CUDATarget	Float32	0.00429702	0.411359	0.00554338	0.936665	9321
6	stats_heat_cuda_test_6.jld2	benchmarks_heat_cuda_test_6.json	1	CUDATarget	Float64	0.00399229	0.410816	0.00560685	0.914994	9297
2	stats_heat_cpu_test_2.jld2	benchmarks_heat_cpu_test_2.json	2	CPUTarget	Float32	0.00364257	0.0701168	0.000993621	0.033357	2409
5	stats_heat_cpu_test_5.jld2	benchmarks_heat_cpu_test_5.json	2	CPUTarget	Float64	0.00380355	0.0753385	0.00100255	0.0379008	2373
2	stats_heat_cuda_test_2.jld2	benchmarks_heat_cuda_test_2.json	2	CUDATarget	Float32	0.00398659	0.0921269	0.00224684	0.231701	2409
5	stats_heat_cuda_test_5.jld2	benchmarks_heat_cuda_test_5.json	2	CUDATarget	Float64	0.00401314	0.0963169	0.00228321	0.236839	2373
1	stats_heat_cpu_test_1.jld2	benchmarks_heat_cpu_test_1.json	5	CPUTarget	Float32	0.00363278	0.0115155	0.000173273	0.00129622	471
4	stats_heat_cpu_test_4.jld2	benchmarks_heat_cpu_test_4.json	5	CPUTarget	Float64	0.00368314	0.0117496	0.000172301	0.00175905	435
1	stats_heat_cuda_test_1.jld2	benchmarks_heat_cuda_test_1.json	5	CUDATarget	Float32	0.00401113	0.0144075	0.00125438	0.0465578	471
4	stats_heat_cuda_test_4.jld2	benchmarks_heat_cuda_test_4.json	5	CUDATarget	Float64	0.00425394	0.0152498	0.00124903	0.0442241	435

Thank you

Questions?

Benchmarks demo?