# datazimmer

## Social Science Computing Unit Budapest

**Feb 04, 2024**

# CONTENTS:

CHAPTER

# ONE

# TO CREATE A NEW PROJECT

- make sure that `python` points to `python>=3.8` and you have `pip` and `git` then `pip install datazimmer`
- run `dz init project-name`
  - pulls project-template
- add a remote
  - both to git and dvc (can run `dz build-meta` to see available dvc remotes)
  - git remote can be given with `dz init`
- create, register and document steps in a pipeline you will run in different *environments*
- build metadata to exportable and serialized format with `dz build-meta`
  - if you defined importable data from other artifacts in the config, you can import them with `load-external-data`
  - ensure that you import envs that are served from sources you have access to
- build and run pipeline steps by running `dz run`
- validate that the data matches the *datascript* description with `dz validate`

## 1.1 Scheduling

- a project as a whole has a cron expression in `zimmer.yaml` to determine the schedule of reruns
- additionally, aswan projects within the dz project can have different cron expressions for scheduling new runs of the aswan projects

## 1.2 Test projects

TODO: document dogshow and everything else much better here

CHAPTER

# TWO

# LOOKAHEAD

- overlapping names convention
- resolve naming confusion with colassigner, colaccessor and table feature / composite type / index base classes
- abstract composite type + subclass of entity class
  - import ACT, inherit from it and specify
  - importing composite type is impossible now if it contains foreign key :(
- add option to infer data type of assigned feature
  - can be problematic b/c pandas int/float/nan issue
- create similar sets of features in a dry way
- overlapping in entities
  - detect / signal the same type of entity
- exports: postgres, postgis , superset

## 2.1 W3C compliancy plan

- test suite for compliance: https://w3c.github.io/csvw/publishing-snapshots/PR-earl/earl.html
- https://github.com/w3c/csvw
  - https://www.w3.org/TR/2015/REC-tabular-data-model-20151217/
  - https://www.w3.org/TR/tabular-metadata/

```
@article{tennison2015model,
  title={Model for tabular data and metadata on the web},
  author={Tennison, Jeni and Kellogg, Gregg and Herman, Ivan},
  year={2015}
}
```

```
@article{pollock2015metadata,
  title={Metadata vocabulary for tabular data},
  author={Pollock, Rufus and Tennison, Jeni and Kellogg, Gregg and Herman, Ivan},
  journal={W3C Recommendation},
  volume={17},
  year={2015}
}
```

CHAPTER

THREE

# INSTALLATION:

## 3.1 using pip

```
pip install datazimmer
```

CHAPTER

FOUR

# GLOSSARY

## 4.1 Namespace

The atomic unit of the knowledge system containing data and metadata

- defines
  - tables
  - composite types
  - entity classes
  - code to build data based on these
- represented by
  - a module in a data project (as datascript) - nested right below the main (src) module
  - a set of YAML files in `{namespace name}/**.yaml` as serialized metadata in the released sdist
    - automatically generated from the code
  - an exported .py file with basic datascript in `{namespace name}/__init__.py` in the released sdist
- can import other namespaces, either to use
  - data (even for foreign keys in tables)
  - defined composite types / entity classes

## 4.2 Data Project

A versioned set of interconnected namespaces with metadata and different environments

- defines
  - namespaces
  - different environments where (usually) the same code runs for different data
- represented by
  - a git repository
    - is a DVC repository
    - based on a template
    - has fixed form tags representing the releases and data versions

## 4.3 Registry

A repository containing data about the releases and dependencies of projects to make importing namespaces straightforward

- represented by
  - a git repository (either local or remote)
  - write access needed to the repo to release to it
- contains data about
  - (named) projects
    - URI
    - versions
    - environment->dvc remote mapping
- contains sdist forms of metadata of projects release there
  - to set up a special PyPI index so that installation and dependency resolution is outsourced

## 4.4 Metadata

Information about the data contained in projects

- defines
  - for each namespace
    - tables
    - composite types
    - entity classes
- represented
  - in a project repository
    - defined in code (datascript object)
      - scrutable
      - entitybase
      - compositetypebase
    - serialized (generated from code)
      - YAML files
  - in runtime
    - converted as soon as possible to dataclasses in bedrock module
  - some even in data output in parquet

## 4.5 Config

- defines
  - name
  - version (this is the metadata version, the data version is determined at release)
  - default-environment name (the first environment in envs config by default)
  - validation-environments (the default-environment by default)
  - registry address (the SSCUB registry by default) TODO: link
  - imported_projects
    - either a list of project names to be imported, where other than name, all default values are used
    - or a dictionary, where the key is the project name (in the registry), and values are:
      - version (metadata version)
      - data_namespaces - the namespaces where loading the data is required
  - in `envs` for each environment (one empty env named complete by default)
    - params for all local namespaces and global params (namespace params default to these if not defined)
      - logged to DVC from here
    - environments of imported projects (where data is needed)
    - specific DVC remote
      - where to push data generated as outputs of running the code from namespaces (TODO - find a proper name, e.g. namespace processor) - identified by the name of a remote defined in DVC config
    - parent env (default-environment by default)
      - all missing keys of parameters or imported ns
- represented as `zimmer.yaml` in project root

## 4.6 Environment

A complete run of the code in an project with its values for parameters and environments for imported data

- defined by config
- …

## 4.7 Tabular data related phrases

- feature: *A named set of columns in a table*
  - can be a primitive feature, foreign key or composite feature
- the subject of records: entity class that is represented in a table

CHAPTER

FIVE

# MOCK PROJECTS

The *dogshow standard* projects and explorer are intended to showcase all features of the package.

They have their test registry (TODO: should have sscu-budapest GitHub repo address)

- dog-show
- dograce
- dogsuccess
- dogcombine

In the code, chenges from versions v0.0, v0.1 and v1.0 are noted with comments - `# add in v0.1:` at the beginning of a line or `# remove in v1.0` at the end

- to test for
- rebuilding the code with the same version
- trying (and failing) to publish a different codebase for a previously published version
- test for changed dependencies

CHAPTER

SIX

# NAMING CONVENTIONS AND RESTRICTIONS

- project names:
  - lower case letters and non-duplicated dashes (-) not at either end
- environment and namespace names:
  - lower case letters and non-duplicated underscores (_) not at either end

## 6.1 Metadata

- `[a-b]*_table` table names based on singular form of entity e.g. `dog_table`
- see all in dogshow standard

CHAPTER

SEVEN

# RULES

Notes of rules that are necessary for operation. Too strict or stupid rules need to change!

- Only one registered function per namespace
  - unless for separate environments, like with the helper function of data loaders and environment creation
- usually, an env run of a namespace processor (is it called this? TODO) reads and writes to one env, the one it corresponds to, but:
  - can only write to its env
  - can read from a different one, but then can only read from that one
  - this allows for registering a function that creates its environment from a base/complete set
- TableNameFeatures should be the name of table feature classes if the table name is to be inferred
- No composite features with the same prefix in the same table
- Feature name can't contain __ (dunder)

CHAPTER

# EIGHT

# CLI (WIP)

- csv format default
- core namespace default
- datazimmer registry repository default registry

```
datazimmer pull dog-show --env top_comps datazimmer pull dogssuccess/sex_matches --parquet
datazimmer pull something --registry uri://sg-registry/...
```

CHAPTER

# NINE

# API

## 9.1 datazimmer Package

sscu-budapest utilities for scientific data engineering

### 9.1.1 Functions

| | |
|---|---|
| *dump_dfs_to_tables*(df_structable_pairs[, ...]) | helper function to fill the detected env of a dataset |
| *get_raw_data_path*(leaf_name[, project]) | if project is None, raw data output path is given, otherwise imported |
| *parse_df*(df, entity[, verbose]) | |
| *register*([procfun, dependencies, outputs, ...]) | registers a function to the pipeline the names of parameters will matter and will be looked up in conf/envs.yaml params |
| *register_data_loader*([fun, extra_deps]) | Convenience functions to use register with typical parameters |
| *register_env_creator*([fun, extra_deps]) | Convenience functions to use register with typical parameters |

#### dump_dfs_to_tables

datazimmer.**dump_dfs_to_tables**(*df_structable_pairs: list[tuple[pd.DataFrame,* ScruTable*]]*, *parse=True*, *skip_empty=False*, ***kwargs*)

helper function to fill the detected env of a dataset

#### get_raw_data_path

datazimmer.**get_raw_data_path**(*leaf_name: str*, *project: str | None = None*) → Path

if project is None, raw data output path is given, otherwise imported

#### parse_df

datazimmer.**parse_df**(*df: DataFrame*, *entity:* AbstractEntity, *verbose=False*)

#### register

datazimmer.**register**(*procfun=None*, ***, *dependencies: list | None = None*, *outputs: list | None = None*, *outputs_nocache: list | None = None*, *outputs_persist: list | None = None*)

registers a function to the pipeline the names of parameters will matter and will be looked up in conf/envs.yaml params

#### register_data_loader

datazimmer.**register_data_loader**(*fun=None*, ***, *extra_deps=None*)

Convenience functions to use register with typical parameters

#### register_env_creator

datazimmer.**register_env_creator**(*fun=None*, ***, *extra_deps=None*)

Convenience functions to use register with typical parameters

### 9.1.2 Classes

| | |
|---|---|
| *AbstractEntity*() | |
| *CompositeTypeBase*() | |
| *DzAswan*([global_run]) | |
| *EntityClass*(name, identifiers, ...) | |
| *Index*() | |
| *Nullable*(dtype) | |
| *PersistentState*() | |
| *ReportFile*(filename) | |
| *ScruTable*(entity[, entity_key_table_map, ...]) | |
| *SourceUrl*(_) | |

### AbstractEntity

**class** datazimmer.**AbstractEntity**

Bases: ColAssigner

### CompositeTypeBase

**class** datazimmer.**CompositeTypeBase**

Bases: ColAssigner

### DzAswan

**class** datazimmer.**DzAswan**(*global_run=False*)

Bases: object

#### Attributes Summary

| | |
|---|---|
| *cron* | |
| *name* | |
| *starters* | |

#### Methods Summary

| | |
|---|---|
| *get_all_events*(handler[, only_latest]) | |
| *get_aswan_status*() | |
| *get_unprocessed_events*(handler[, only_latest]) | |
| *prepare_run*() | this runs prior to running the project |
| *run*() | |

#### Attributes Documentation

**cron: str = ''**

**name: str = None**

**starters: dict[type[ANY_HANDLER_T], list[str]] = {}**

### Methods Documentation

**get_all_events**(*handler: ANY_HANDLER_T*, *only_latest=True*)

**get_aswan_status**()

**get_unprocessed_events**(*handler: ANY_HANDLER_T*, *only_latest=False*)

**prepare_run**()

this runs prior to running the project

**run**()

## EntityClass

class datazimmer.**EntityClass**(*name: str*, *identifiers: List[Union[datazimmer.metadata.atoms.PrimitiveFeature, datazimmer.metadata.atoms.CompositeFeature, datazimmer.metadata.atoms.ObjectProperty]] = <factory>*, *properties: List[Union[datazimmer.metadata.atoms.PrimitiveFeature, datazimmer.metadata.atoms.CompositeFeature, datazimmer.metadata.atoms.ObjectProperty]] = <factory>*, *parents: List[ForwardRef('EntityClass')] = <factory>*, *description: Optional[str] = None*)

Bases: _AtomBase

### Attributes Summary

| |
|---|
| *description* |
| *table_all_columns* |
| *table_feature_cols* |
| *table_feature_dt_map* |
| *table_full_dt_map* |
| *table_index_cols* |
| *table_index_dt_map* |

#### Attributes Documentation

**description: str | None = None**

**table_all_columns**

**table_feature_cols**

**table_feature_dt_map**

**table_full_dt_map**

**table_index_cols**

**table_index_dt_map**

### Index

**class** datazimmer.**Index**

Bases: object

### Nullable

**class** datazimmer.**Nullable**(*dtype*)

Bases: type

### PersistentState

**class** datazimmer.**PersistentState**

Bases: object

#### Methods Summary

| |
|---|
| *get_conf*() |
| *get_full_name*() |
| *load*() |
| *save*() |

### Methods Documentation

**classmethod get_conf**()

**classmethod get_full_name**()

**classmethod load**()

**save**()

## ReportFile

**class** datazimmer.**ReportFile**(*filename: str*)

Bases: object

### Attributes Summary

| |
|---|
| *current_path* |

### Methods Summary

| |
|---|
| *env_path*(env) |
| *env_posix*(env) |
| *write_bytes*(blob) |
| *write_text*(text) |

### Attributes Documentation

**current_path**

### Methods Documentation

**env_path**(*env*)

**env_posix**(*env*)

**write_bytes**(*blob*)

**write_text**(*text*)

### ScruTable

class datazimmer.**ScruTable**(*entity: type[AbstractEntity]*, *entity_key_table_map: dict[str, ScruTable] | None = None*, *partitioning_cols: list[str] | None = None*, *max_partition_size: int | None = None*)

Bases: `object`

#### Attributes Summary

| |
|---|
| *dfs* |
| *paths* |

#### Methods Summary

| |
|---|
| *env_ctx*(env) |
| *get_partition_paths*(partition_col[, env]) |
| *purge*() |

#### Attributes Documentation

**dfs**

**paths**

#### Methods Documentation

**env_ctx**(*env*)

**get_partition_paths**(*partition_col*, *env=None*)

**purge**()

### SourceUrl

class datazimmer.**SourceUrl**(_)

Bases: `str`

### 9.1.3 Class Inheritance Diagram

CHAPTER

TEN

# RELEASE NOTES

## 10.1 v0.1.0

start datazimmer from sscutils

## 10.2 v0.1.1

fix build - add some cron logic

## 10.3 v0.1.2

revert build

## 10.4 v0.1.3

## 10.5 v0.1.4

## 10.6 v0.1.5

## 10.7 v0.2.1

init explorer frame

## 10.8 v0.2.2

auth improvement

## 10.9 v0.2.3

tighter explorer

## 10.10 v0.2.4

tighter explorer

## 10.11 v0.2.5

multitable dec

## 10.12 v0.2.6

minimal to dataset level

## 10.13 v0.2.7

scrutable extension

## 10.14 v0.3.0

simplification and unification

## 10.15 v0.3.1

incremental fixes

## 10.16 v0.3.10

flat index and improved secrets

## 10.17 v0.3.2

requirement and explorer updates

## 10.18 v0.3.3

init explorer and os fix

## 10.19 v0.3.4

nb requirement fix

## 10.20 v0.3.5

title detection fix

## 10.21 v0.3.6

explorer upgrades

## 10.22 v0.3.7

freeze jb version

## 10.23 v0.3.8

flexible pulling

## 10.24 v0.3.9

sql loader extension

## 10.25 v0.4.0

aswan integration init

## 10.26 v0.4.1

here goes nothing

## 10.27 v0.4.10

dvc realign, minor extensions and reorg

## 10.28 v0.4.11

stage non cached outputs

## 10.29 v0.4.12

private zenodo

## 10.30 v0.4.13

aswan reset plus scrutable pickle fix

## 10.31 v0.4.14

minimal alignment, pre-dependency fix

## 10.32 v0.4.15

move import

## 10.33 v0.4.2

collect dependency fix

## 10.34 v0.4.3

minor iterations, redundancy removal

## 10.35 v0.4.4

git fixes, unifications

## 10.36 v0.4.5

aswan alignment

## 10.37 v0.4.6

aswan align

## 10.38 v0.4.7

simplifications and raw-data introduction

## 10.39 v0.4.8

properly citeable

## 10.40 v0.4.9

init zenodo integration

## 10.41 v0.5.0

fix dependencies and drop explorer

## 10.42 v0.5.1

many many bugs fixed

## 10.43 v0.5.2

external dvc zimmauth

## 10.44 v0.5.3

properly rm dvc as dependency

## 10.45 v0.5.4

metaclass def bugfix

CHAPTER

ELEVEN

# INDICES AND TABLES

- genindex
- modindex
- search

# PYTHON MODULE INDEX

## d

# INDEX

# W