# Datatoolbox Documentation

***Release 0.6.5***

**__authors__**

**Apr 30, 2024**

# CONTENTS

CHAPTER

ONE

# INSTALLATION

## 1.1 Installation

### 1.1.1 Software installation outside python

```
sudo apt install git
```

### 1.1.2 Datatoolbox python dependencies

See enviroment file.

## 1.2 Set up datatoolbox and connect to a database

First time you import datatoolbox, you need to set up the local database and link it to datatoolbox. Otherwise, a simple SANDBOX data structure is loaded for playing around.

### 1.2.1 1) Create local empty database folder

```
import datatoolbox as dt
dt.admin.create_empty_datashelf('/your/path/on/local/hard_disk')
```

This created an empty database that can be linked to datatoolbox. This link is one by creating you personal setting including you name and the same path to the database folder on you hard disk without quotation marks.

```
dt.admin.change_personal_config()
```

```
>>> dt.admin.change_personal_config()
Please enter your initials: MM
Please enter path to datashelf: /your/path/on/local/hard_disk
>>>
```

### 1.2.2 2) Set up remote Access

Datatoolbox allows to automatically integrate new data sets by using git + ssh connections. However, this requires having git installed on you system and ssh connection properly set up. The following example shows the outlines the required steps to access gitlab via ssh:

- Create account and apply to https://gitlab.com/climateanalytics
- set up ssh key access (https://docs.gitlab.com/ee/ssh/)

Available datasets on gitlab: https://gitlab.com/climateanalytics/datashelf

### 1.2.3 3) Import remote sources

Import of remote source (using git in the background)

```
import datatoolbox as dt
dt.core.DB.importSourceFromRemote('WDI_2020')
dt.core.DB.importSourceFromRemote('PRIMAP_2019')
```

After the import, the datasets will be available in your local database and can be accessed by functions of datatoolbox.

CHAPTER

TWO

# GETTING STARTED

## 2.1 Datatoolbox - First steps

Using datatoolbox requires to components to run in parallel. Datatoolbox itself is a python packages that contains all the functionality, structure and access to data.

The data itself is not stored or installed with datatoolbox. The data is separated in a database like file structure. This structure must be available on you local disk. To collaborate with different users the database is shared with other using git. For example on gitlab, a freely accessible version of the World development indicators is available. Once, you have a local datatbase set up and you have ssh access to the git account, datatoolbox brings the tools to automatically pull new data from a shared online account.

## 2.2 Working with datatoolbox

Each individual data that is stored in the database is identified by a table ID. This ID is unique is constructed as the based on three attributes. variable, pathway and source.

The **variable** describes the physical and societal kind of the data. It consists of the physical entity and the category (optional) and is merged as stings using the pipe "|".

**Entity** defines the physical species that is described by the data, for example for emission being 'Emissions|CO2' or 'Emissions|KYOTOGHG'. But also all other entities like 'Population', 'Primary_energy' are included. Entities are specifically defined and should follow a specific naming convention to allow comparison across multiple different data source.

**Category** provides additional separation to different sectors or categories. This can be an industrial sector "Transport" or various types 'Electricity generation' to indicate that this emission data is only related to a specific sector.

Currently datatoolbox is set up that the property variable is generated automatically. Thus, the upper examples would lead to the variable:

- Emissions|CO2|Transport
- Emissions|CO2|Electricity_generation
- Emissions|KYOTOGHG|AR4
- Population

The **pathway** is category is related to the modeling community and describes how the data is collected or produced. But unrelated to modeling, historic data would be indicated here. Pathway is constructed from the two related properties scenario and model (optional).

**Scenario** describes which underlying scenario is used for the data. In the most simple case, for historic data, the scenario is 'Historic'. Otherwise, for projected or modeled data, various indicators like the Shared Socioeconomic Pathways (e.g. 'SSP1') or other categories like 'medium fertility' can be applied.

**Model** describes optionally the model that is used for the data. For example Integrated Assessment Models like 'Message' or 'AIM' can be indicated here.

Using scenario and model, the property pathway is constructed as for example:

- SSP1-RCP2.6|MESSAGE_v2.4
- Historic
- Projection|Medium_fertility

The last component of the ID is the **source** of the data. This can be freely defined. But if you provide 'source_name' and 'source_year', the source property is automatically constructed from those two.

**source_name** defines the name of the organization or individual that provided the data.

**source_year** is the year of publishing

Finally, the the table ID is constructed by merging the property strings variable, pathway and source using double lower score "**". Thus, the final ID could look like:**

- Population|Total__Historic__Worldbank_2020
- Emissions|CO2|Transport__SSP1|MESSAGE_v2.4__IAMC15_2019

figures/ID_meta_data.svg.png

## 2.3 Data structure:

### 2.3.1 Inventory

The inventory is pandas DataFrame containing the following meta data:'variable', 'entity', 'category', 'pathway', 'scenario', 'model', 'source', 'source_name', 'source_year', 'unit'

Each row represents one data table and the index gives the unique tableIDs. The inventory can be accessed by : dt.inventory()

Normal pandas command can be used:

- Show columns: print(dt.inventory().columns)
- Show head of table: print(dt.inventory().head())

### 2.3.2 Find tables

Datatables that contain the search string

```
dt.findp(variable ='', entity='', category='', scenario ='', model='' source='')
```

Two find functions are currently available:

- dt.findp : Finds mathces according to patterns
- dt.findc : Finds matches that contain the given strings

#%% List all data sources

```
sources = list(dt.findp().source.unique())
sources.sort()
print(sources)
```

#%% List all scenarios within a source

```
res = dt.findp(source='PRIMAP_2019')
print(res.scenario.unique())
```

#%% List all variables within a source

```
print(res.entity.unique())
```

#%% List all Emissions|KYOTOGHG data tables

```
res = dt.findp(entity = "Emissions|KYOTOGHG", source='PRIMAP_2019')
print(res.entity.unique())
```

```
## Handling sources:

Import a new source form remote (e.g. gitlab)
```

dt.import_new_source_from_remote(source_id)

```

Export a new source to the remote repository
```

dt.export_new_source_to_remote(source_Id)

```

Update local existing source:
```

dt.pull_source_from_remote(source_id)

```

Update remote source with your new local data
```

dt.push_source_to_remote(source_id)

```

(continues on next page)
```

(continued from previous page)

```
## Access data tables

Tables are accessed by their ID, given by the inventory index returning a Datatable
```

table = dt.getTable(tableID)

```
or given by the result dataframe
```

table = dt.getTable(res.index[100])

```
Multiple tables can be loaded at once and returned in a tableSet (being a dictionary+)
```

tables = dt.getTables(res.index[:10])

```
#%% Datatable
```

print(type(table))

```
A Datatable is a pandas Dataframe with addition restrictions and functionalities

- columns are integer years
- the index only consists of varied region identifiers
- the data is numeric
- each table as meta data attached with some required variables (see config.py)
- each tables only consists of one variable with the same unit which
- allows for easy unit conversion (see unit_conversion.py as tutorial)
- each table is stored as a csv file each inf the dedicated source folder
```

print(dt.core.DB._getTableFilePath(table.ID))

```
## Handling and changing tables

Creating a new table
```

new_table = dt.Datatable(columns = range(2000,2011), index = ['IDN','FRA'] meta = {'entity' : 'Population', 'scenario' : 'Historic', 'source' : 'ORGANISATION_20XX', 'unit' : 'thousands'} )

```

Commit table to database (storing to the database locally)
```

#one table dt.commitTable(table, message ='commit message for git')

#multiple tables tables = [table1, table2] dt.commitTables(tables, message= "Commiting nutiple tables")

```



## DataSet
```

print(type(tables))

```
A TableSet is a dictionary with minor additional functionalities
```

(continues on next page)

(continued from previous page)

```
## Interfaces to other python data packages

### Excel
```

tables.to_excel('output_test.xlsx')

```
### PyIAM
```

iamDataFrame = tables.to_IamDataFrame()

```
### XArray

### Emission module PIK



###
```

CHAPTER

# THREE

# PACKAGE CONTENT:

## 3.1 Data structures

## 3.2 Core

## 3.3 CSV-based Database

### 3.3.1 Database Class

### 3.3.2 Git Repository Manager Class

## 3.4 Datatoolbox - tools

### 3.4.1 Excel tools

### 3.4.2 Matplotlib tools

### 3.4.3 Pandas tools

### 3.4.4 Xarray tools

## 3.5 License

OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

## 3.6 Help

Any questions please contact: andreas.geiges@climateanalytics.org

CHAPTER

FOUR

# INDICES AND TABLES

- genindex
- modindex
- search