DAPT Documentation

Release 0.9.0

Ben Duggan

Mar 15, 2021

Contents:

CHAPTER 1

Install

To install DAPT you can either use a release downloaded from GitHub (recomended), or download the most recent "stable" version of the code by cloning or downloading from the master brantch. If you install using a release they can be found at https://github.com/BenSDuggan/DAPT/releases. You can also download from the GitHub page for run `git clone https://github.com/BenSDuggan/DAPT`.

Once you've downloaded the code, you can install the python libraries by running `pip3 install -r requirements.txt` in the terminal from inside the root folder. You can then open a python session and test to see if everything installed correctly:

```
import dapt
dapt.__version__
```

You should then see some version number corresponding with the version you downloaded.

You can now use the library! However, the functionality can be greatly increased by connecting some other services such as Google Sheets or Box. See the below guides on how to include this functionality.

1.1 Google Sheets Installation

Google Sheets can be used as a database to store your paramater sets. The advantage to using this "database" over a file containing the paramaters is that a team can work on the set more collabortively and update the paramater list on the fly. The bigger advantage is that the paramater list can be run dynamically. By this I mean that people, running the library simultaneously, can connect to Google Sheets and get the next paramater set in the list.

1.1.1 API Credentials

To use Google Sheets you will need to use the Google Sheets API and generate the proper credentials.

1. Start by going to the Google Developer Console and login using a Google account.
2. Create a new project and name it whatever you'd like. You can do this by selecting the down arrow next in the top left corner of the page next to the "Google API" logo and clicking "New Project".

3. Click “ENABLE APIS AND SERVICES”, search for “Google Sheets API” and click it. Then click “Enable”.
4. Click the “Credentials” tab from the menu on the left side of the page.
5. Click the dropdown at the top of the page that says “CREATE CREDENTIALS” and select “Service account”.
6. Give the service a name and click “Create”.
7. In the next section asking about service account permissions, create a role with by selecting “Project” then “Editor”. Then select “Continue”.
8. Select “CREATE KEY”, ensure the key type of “JSON” is selected and click “CREATE”. You will give DAPT the path to this JSON file when using Google Sheets. Then click “DONE”.
9. You should now be on the Credentials page of the Google Sheets API. Record the email address in the Service Account table. It should look something like `*.iam.gserviceaccount.com`.

1.2 Box Installation

Box is a cloud storage service that many universities allow students, facutie and staff to use. The advantage of box is that it allows a large amount of data to be uploaded to a common place where team memebers can observe data. In order to allow DAPT to upload to box, you must create some API credentials.

1.2.1 API Credentials

1. Start by going to the Box Development website and clicking on the blue “Console” button. Then log in.
2. Click “Create New App”. Then click “Custom App” and “Next” on the next page.
3. On the “Authentication Method” page click “Standard OAuth 2.0 (User Authentication)” and name your project. Then click “View Your App”.
4. Scroll down to the “OAuth 2.0 Credentials” section and record the Client ID and Secret. You will pass these to the DAPT Box class to allow the Box SDK to work.
5. Lastely, scroll down to the “OAuth 2.0 Credentials” section and change the url to `http://127.0.0.1:5000/return`. Then click “Save Changes”.

CHAPTER 2

Usage

Once you have installed DAPT and have verified that it's installed correctly you can start setting it up for actual parameter runs. Below are links showing how to use the project.

CHAPTER 3

Example scripts

This still needs to be completed.

3.1 CSV Example

Heres how

Example of how to use DAPT with a CSV file

`examples.csv_example.`**`create_file`**`()`
Reset csv. This is just used update the csv and reset it so interesting things happen.

CHAPTER 4

DAPT Documentation

4.1 Param

This is the main class which interact with the database to get and manage parameter sets. The `Param` class links all the other classes together enabling a paramater set to be run

4.1.1 Database

In order to get the paramaters, the `Param` class needs to be given a `Database` instance (e.g. `Sheets` or `Delimited_file`). Regardless of the which database option is used, it must be set up in a particular way as shown below.

Fields

A field is the key (or identifier) used to get the value when a parameter set is returned. Each database is required to have and `id` and `status` field. There are many optional fields which can be used to give additionally information about the run such as start time and who performed the run. Below are the fields that are used with parsing parameter sets. Required parameters are marked with an astrict(*).

Fields	Description
`id`* (str)	Unique parameter set installed
`status`* (str)	The current status of the parameter set. Blank values are unran (default), `finished` have finished and `failed` have failed.
`start-time` (str)	The time that the parameter set began. Times are in UTC time format.
`end-time` (str)	The time that the parameter set finished. Times are in UTC time format.
`performed-by` (str)	The username of the person that ran the parameter set.
`comments` (str)	Any comments such as error messages relating to the parameter set.

class dapt.param.**Param**(*database*, *config=None*)
Create a Param instance with a database and optional config file.

Parameters

- **database** (Database) – a Database instance (such as Sheets or Delimited_file)
- **config** (Config) – a config object which allows for more features. This is optional.

failed(*id*, *err=''*)
Mark a parameter set as failed to completed.

Parameters

- **id** (*str*) – the id of the parameter set to use
- **err** (*str*) – the error message. Empty by default.

Returns The new parameter set that has been updated or False if not able to update.

next_parameters()
Get the next parameter set if one exists

Returns An OrderedDict containing the key-value pairs from that parameter set or None if there are no more to sets.

successful(*id*)
Mark a parameter set as successfully completed.

Parameters **id** (*str*) – the id of the parameter set to use

Returns The new parameter set that has been updated or False if not able to update.

update_status(*id*, *status*)
Update the status of the selected parameter. If status is not included in the parameter set keys then nothing will be updated.

Parameters **id** (*str*) – the id of the parameter set to use

Returns The new parameter set that has been updated or False if not able to update.

4.2 Config

Class that allows for reading and modification of a configuration (config) file. A config file is not required but using one will make using DAPT much easier to use and greatly increase increase it's functionality. A configuration file is simply a JSON file. There are some reserved keys but you can add your own and refer to them throughout your program.

4.2.1 Fields

Fields	Description
`last-test` (str)	The last test id that was run.
`user-name` (str)	The box username of the user.
`spreedsheet-id` (str)	The Google spreedsheet ID being used.
`sheets-creds-path` (str)	The Google Sheets credentials file path.
`sheet-worksheet-id` (str)	The Google Sheets worksheet id. Sheets are indexed at 0.
`sheet-worksheet-title` (str)	The Google Sheets worksheet title.
`client_id` (str)	Box API client ID.
`client_secret` (str)	Box API client secret.
`box-folder-id` (str)	The box folder id to use
`reset-time` (str)	The time that the box access-token needs to be refreshed.
`num-of-runs` (int)	The number of paramater sets to run.
`computer-strength` (int)	Any comments such as error messages relating to the parameter set.
`access-token` (str)	The box access token for the particular session.
`refresh-token` (str)	The box refresh token for the particular session.

class dapt.config.**Config**(*path='config.json'*)

Class which loads and allows for editing of a config file

Parameters **path** (*string*) – path to config file

static create(*path='config.json'*)

Creates a config file with the reserved keys inserted.

Parameters **path** (*string*) – path where config file will be written

read_config()

Reads the file with path set to self.path

Returns Dictionary of config file

static safe(*path='config.json'*)

Safe config file by removing accessToken and refreshToken.

Parameters **path** (*string*) – path where config file will be writen

update_config()

Change value for a given key in the config file

Returns Dictionary of config file

4.3 Database

The `Database` class is the basic interface for adding parameter set hosting services. The idea is that the core methods (`get_table`, `get_keys`, `update_row` and `update_cell`) stay the same so that the inner workings can use multiple sources to access the parameter sets. In general, there shouldn't be any more arguments added. The exception to this is `__init__`. It may be necessary or desirable to add additional, optional, arguments. This should be done by overloading the method or providing a default option to the argument.

When prepareing for a parameter sweep the collection of parameter sets can be thought of as a database where the name of each paramate is a column name, each row is a paramater set and the value at the i-th column and j-th row is the value. This is the approach of DAPT.

class dapt.database.**Database**
An interface for accessing and setting parameter set data.

get_keys()
Get the keys of the parameter set

Returns Array of strings with each element being a key (order is preserved if possible)

get_table()
Get the table from the database.

Returns An array with each element being a dictionary of the key-value pairs for the row in the database.

update_cell(*row_id*, *key*, *value*)
Get the keys of the parameter set

Parameters

- **row_id** (*int*) – the row id to replace
- **key** (*str*) – the key of the value to replace
- **value** (*str*) – the value to insert into the cell

Returns A boolean that is True if successfully inserted and False otherwise.

update_row(*row_id*, *values*)
Get the keys of the parameter set

Parameters

- **row_id** (*int*) – the row id to replace
- **values** (*OrderedDict*) – the key-value pairs that should be inserted

Returns A boolean that is True if successfully inserted and False otherwise.

4.4 Delimited file

Create a CSV instance which can be used by param to get and run param sets

class dapt.delimited_file.**Delimited_file**(*csv_file*, *delimiter=', '*)
An interface for accessing and setting paramater set data.

get_keys()
Get the keys of the paramater set

Returns Array of strings with each element being a key (order is preserved if possible)

get_row_index(*column_key*, *row_value*)
Get the row index given the column to look through and row value to match to.

Parameters

- **column_key** (*str*) – the column to use.
- **row_value** (*str*) – the row value to match with in the file and determin the row index.

Returns The index or -1 if it could not be determined

get_table()
Get the table from the database.

Returns An array with each element being a dictionary of the key-value pairs for the row in the database.

update_cell(*row_index*, *key*, *value*)
Get the keys of the paramater set

Parameters

- **row_index** (*int*) – the row id to replace
- **key** (*str*) – the key of the value to replace
- **value** (*str*) – the value to insert into the cell

Returns A boolean that is True if successfully inserted and False otherwise.

update_row(*row_index*, *values*)
Get the keys of the paramater set

Parameters

- **row_index** (*int*) – the row id to replace
- **values** (*OrderedDict*) – the key-value pairs that should be inserted

Returns A boolean that is True if successfully inserted and False otherwise.

4.5 Sheets

Class which allows for Google Sheets to be used as paramater set database.

Note: if you have data in the first row, you must have entries in some other row.

class dapt.sheets.**Sheet**(**args*, ***kwargs*)
An interface for accessing and setting paramater set data. You must either provide a Config object or client_id and client_secret.

Keyword Arguments

- **config** (Config) – A Config object which contains the client_id and client_secret.
- **spreedsheetID** (*str*) – the Google Sheets ID
- **creds_file** (*str*) – the path to the file containing the Google API credentials. Default is credentials.json.
- **sheet_id** (*int*) – the the sheet id to use (0 by default)

get_key_index(*column_key*)
Get the column index given the key.

Parameters **column_key** (*str*) – the key to find the index of

Returns The index or -1 if it could not be determined.

get_keys()
Get the keys of the paramater set

Returns Array of strings with each element being a key (order is preserved if possible)

get_row_index(*column_key*, *row_value*)
Get the row index given the column to look through and row value to match to.

Parameters

- **column_key** (*str*) – the key to find the index of
- **row_value** (*str*) – the value of the cell to fine

Returns The index or -1 if it could not be determined.

get_table()
Get the table from the database.

Returns An array with each element being a dictionary of the key-value pairs for the row in the database.

update_cell(*row_id*, *key*, *value*)
Get the keys of the paramater set

Parameters

- **row_id** (*str*) – the row id to replace
- **key** (*str*) – the key of the value to replace
- **value** (*str*) – the value to insert into the cell

Returns A boolean that is True if successfully inserted and False otherwise.

update_row(*row_id*, *values*)
Get the row of the paramater set

Parameters

- **row_id** (*int*) – the row id to replace
- **values** (*OrderedDict*) – the key-value pairs that should be inserted

Returns A boolean that is True if successfully inserted and False otherwise.

worksheet(**args*, ***kwargs*)
Get a Google Sheet object. You can give a worksheet id or title or nothing (get values from Config file). If the you give a worksheet id and title then the id will be used.

Keyword Arguments

- **id** (*str*) – the Google Sheets worksheet id
- **title** (*int*) – the Google Sheet worksheet title

Returns A Google Sheet worksheet

4.6 Box

Class that allows for access to the box API and methods to directly upload files

class dapt.box.**Box**(**args*, ***kwargs*)
Class which allows for connection to box API. You must either provide a Config object or client_id and client_secret.

Keyword Arguments

- **config** (Config) – A Config object which contains the client_id and client_secret.
- **client_id** (*str*) – The Box client ID.
- **client_secret** (*str*) – The Box client secret.

connect (*access_token=None*, *refresh_token=None*)
Tries to connect to box using arguments provided in Config and starts server for authorization if not.

Parameters

- **access_token** (*str*) – Optional argument that allows DAPT to connect to box without going through web authentification (assuming refresh_token is given and not expired).
- **refresh_token** (*str*) – Optional argument that allows DAPT to connect to box without going through web authentification (assuming access_token is given and not expired).

Returns Box client if successful

updateTokens (*access_token*)
Refresh the access and refresh token given a valid access token

Parameters **access_token** (*string*) – box access token to be refreshed

Returns Box client

uploadFile (*folderID*, *path*, *name*)
Upload a file to box using the current client

Parameters

- **folderID** (*string*) – the box folder ID that the file should be added to (found in URL)
- **path** (*string*) – the path to the file on your local machine
- **file** (*string*) – the name of the file you wish to upload

Python Module Index

d

e

Index