DAPT Documentation

Release 0.9.1

Ben Duggan

Jun 02, 2020

Contents:

A library to assist with running parameter sets across multiple systems. The goal of this library is to provide a tool set and pipeline that make organizing, running and analyzing a large amount of parameter easier. Some of the highlights include:

- Provide an easy way to run paramater sets.
- Protocol for allowing teams to run parameter sets concurrently.
- Use Google Sheets as a database to host and manage paramater sets.
- Access to the Box API which allows files to be uploaded to box.

CHAPTER 1

Overview

When working on a project with or without access to high performance computing (HPC), there is often a need to perform large parameter sweeps. Before developing DAPT, there were several problems the ECM team in Dr. Paul Macklin's research lab identified. First, it was difficult to manage a large number of parameter sets with a large number of parameters. Second, it would be nice to use Google Sheets to run the parameters for easier collaboration and management. Third, only one person in the group would be running all the parameters, making their computer useless for the duration of the runs. Finally, we needed to upload the data to Box for permanent storage and to allow the rest of the team to view the data.

DAPT was written to solve these problems. A "database" (CSV or Google Sheet) is used to store a list of parameter sets. This database is managed by the *Param* class and provides methods to interact with and manage parameter sets. the *Box* class allows data to be uploaded to Box.com. Sensitive API credentials can be stored in a config file (via the *Config* class) which can also be accessed by users to get other variables.

Future versions of the project will work to improve documentation, add examples, cleanup current functionality and add more features. While most of the *dapt* module is documented, the intended way of using each method is not clearly explained. There are examples given for the main features, however, again there is not a satisfactory amount of documentation. Some of the exciting new features to come will be notification and logging integration. For example, we would like to add Slack notification so teams can be notified if there is an error with a test.

1.1 Install

The easiest way to install DAPT is using pip. To do so type:

```
pip install dapt
```

Alternatively, you can download a version the project. It is recommended to download a release of the project from GitHub for improved stability. If you would like to download the most up to date version, then download the repo or clone it on your machine `git clone https://github.com/BenSDuggan/DAPT`. Once downloaded navigate to the root of the project (DAPT) and run `pip install -r requirements.txt` to install all of the dependencies. If you use this method of installation, you will need to write all of your Python scripts using DAPT in the root directory of the project. For these reasons, it's recommended to only use this method if would like to contribute to the project.

You can then test to make sure everything installed by starting a python session and then running:

```
` import dapt dapt.__version__ `
```

You should see a version looking like `0.9.*`.

DAPT is maintained for Python >= 3.6 and a full list of requirements is given in requirements.txt

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.

1.1.1 Google Sheets Installation

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

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`.

1.1.2 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.

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".

1.2 Usage

Once you have installed DAPT and have verified that it's installed correctly you can start setting it up for actual parameter runs. There are several ways to run DAPT but the basic philosophy is outlined below. You can also look at specific examples. To use DAPT, start by importing it.

```
import dapt
```

DAPT can be run with or without a configuration file. The code is easier to use with a config file but it is not strictly necessary. If you would like to create a config file, you should consult the Config class documentation. Assuming you have created a config file called `config.json`, you can create a Config object.

```
config = dapt.Config(path='config.json')
```

Next, you need to pick a Database. A Database is a class that allows you get access a list of parameter sets. There are currently two Databases: a Delimited file and Google sheets. Below shows how to create the database objects.

```
db = data.Delimited_file('csv_file.csv', delimiter=',') # Create a Delimited file DB
# or
db = data.Sheet(config=config) # Create a Sheet DB with a config file
# or
spreedsheet_id = 'xxxxxx' # Google Sheet spreedsheet id
creds = 'credentials.json' # Path to Google Sheets API credentials
db = data.Sheet(spreedsheet_id=spreedsheet_id, creds=creds) # Create a Sheet DB with␣
↪a config file
```

Now you can create the Param object to start processing parameters. Create a Param object with the code below.

```
param = dapt.Param(db, config=config)
```

You can now use the methods in the Param class to get the next parameter set and manage the parameter set.

1.3 Examples

Examples of DAPT are kept in the examples folder. There are basic examples of the main features of DAPT including using a delimited file, Google Sheets and Box. There is also an example of how DAPT can be used with PhysiCell. It is recommended that you start with the csv_example.py script as it is the simplest to use and only requires DAPT to be installed. The other scripts require API keys to be generated.

1.4 DAPT Documentation

1.4.1 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

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.

1.4.2 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.

Fields

There are several standard fields (keys) that are used by DAPT for credentials and parameter settings.

Fields	Description
`last-test` (str)	The last test id that was run.
`user-name` (str)	The box username of the user.
`sheets-spreedsheet-id` (str)	The Google spreedsheet ID being used.
`sheets-creds-path` (str)	The Google Sheets credentials file path.
`sheets-worksheet-id` (str)	The Google Sheets worksheet id. Sheets are indexed at 0.
`sheets-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.
`remove-zip` (bool)	Have tools.data_cleanup() remove zip files if true.
`remove-movie` (bool)	Have tools.data_cleanup() remove mp4 files if true.

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

get_value(*key*, *recursive=False*)
Get the first value of the given key or return `None` if one doesn't exist.

Parameters

- **key** (*str or list*) – the key (given as a string) or List containing the path to the value
- **recursive** (*bool*) – recursively look through the config for the given key. False by default. If recursive is set to True then key must be a string.

Returns The value associated to the given key or None if the key is not in the dictionary.

has_value(*key*)
Checks to see if the config contains the key and a value other than None.

Parameters **key** (*str*) – The key to determine if it has a value

Returns True if the key has a value, False otherwise.

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(*key=None*, *value=None*)
Given a key and associated value, updated the config file. Alternatively, you can give no arguments and the config dict will be saved. You can also do both.

Parameters

- **key** (*str*) – the key to use. If none is given then nothing will be updated in the dictionary.
- **value** (*str*) – the value associated ot the key.

Returns Dictionary of config file

1.4.3 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. You should overide these methods when making a class that inherits `Database`. You shouldn't expect that any other method will be called by the `Param` class, the main class that uses databases. It may be benifitial to add helper methods though (e.g. `get_worksheet()` in `Sheets`).

Databases should give key-value pairs, where the keys are the "ids" of the table and the values are the values in that given row. When getting the table, the result should be an array of dictionaries that contain the contents of the row.

Indexing

The database should use indexing starting from 0.

1.4.4 Delimited file

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

1.4.5 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.

1.4.6 Box

Class that allows for access to the box API and methods to directly upload files. If you wish to use the Box API you should view the install.

Authentication

In order for the Box API to work, it needs to get a user specific access and refresh token. Box provides access tokens to users which are a session key. They remain active for one hour at which time they must be refreshed using the refresh token. Once a new access and refresh token has been given, the old one will no longer work.

The tokens can be provided in three ways. First, you can run `Box(...).connect()` which will start a flask webserver. You can then proceed to 127.0.0.1:5000 and log in with your Box username and password. This is done securely through Box and you username and password cannot be extracted. Second, you can insert the access and refresh token in the config file. Then the Box class will use these tokens. The final way to provide the tokens is by directly passign them to `Box(...).connect(access_token=<your access token>, refresh_token=<your refresh token>)`.

On a server, where you have no access to a web browser, you will need to get the tokens using a computer which has a web browser. You can then place those tokens in the config file or directly pass them to the `connect()` method.

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

1.5 Contribute

If you would like to contribute please fork the repo and make a pull request explaining what you added/fixed and why you added it. When you write a new feature please write tests in the `test` directory and documentation in the `docs` folder.

1.5.1 Documentation

Documentation is performed using Sphinx. The docs folder holds all of the resources to document the code. If you're not familiar with Sphinx you can read this Medium tutorial for an introduction. Google docstrings are used for inline commenting inside each file.

You can compile the docs by running `sphinx-build -M html . _build`, assuming you have sphinx installed. This will create the html documentation in /docs/_build/html.

1.5.2 Tests

Tests are located in the tests folder and written using pytest. You can run the tests locally by running `python3 -m pytest` in the root DAPT directory. This assumes that you have a configuration file named `test_config.json` in the root directory. The convention used is to name all files and functions in the test directory `test_x`, where x is the name/description of the test.

Python Module Index

d

Index