firetable Documentation

Release 0

David Seibert

Jun 18, 2016

Table of Contents

CHAPTER 1

A CLI tool and Python library for Airtable

1.1 Introduction

Airtable is the best, but sometimes (nay, *oftentimes*) a web interface is not the best for a workflow. Airtable provides a handy and comprehensive API for downloading and uploading JSON data. Firetable is meant to bridge Airtable and the command line with:

1. a git-esque command line tool that supports standard UNIX pipes' and
2. a Python library for using the API directly.

It's called Firetable to preserve the namespace rights in case the Airtable folks want to hand down a blessed airtable.py package someday, and because I hope to eventually expand this package beyond the simple API stuff into integration with pandas, reportlab, etc.

1.2 Things that are Working Right Now

1. Download a complete table
2. Download a table view
3. Download a table restricted to specified fields

1.3 Up Next

1. Add support for *sort* and *filterByFormula* parameters.
2. Add support for POST requests.
3. Editing
   (a) Download data and convert to YAML or Markdown.
   (b) Open temp file in default text editor.
   (c) Convert back to JSON and upload data.

1.4 Index

1.4.1 Firetable Commands

Airtable API command classes.

Copied fairly faithfully from the Airtable API documentation. I wouldn't have known where to begin if it weren't for nicocanali/airtable-python.

Why am I using the command design pattern? I don't know. I don't even know if I'm using it correctly. It seemed like a good idea. I was thinking something about validation methods, eventually, I think.

Some day.

The text width is set to 54, so I can see it comfortably in Pythonista for iOS.

class `firetable.commands.`**`Command`**(*endpoint*, ***kwargs*)
Bases: `object`

`execute`()

class `firetable.commands.`**`Delete`**(*endpoint*, ***kwargs*)
Bases: *`firetable.commands.Command`*

Delete command class for Airtable API.

Parameters **`endpoint`** (`Endpoint`) – An Endpoint object pointing to a record.

class `firetable.commands.`**`Get`**(*endpoint*, ***kwargs*)
Bases: *`firetable.commands.Command`*

Get command class for Airtable API.

Parameters

- **`endpoint`** (`Endpoint`) – An Endpoint object pointing to a record or a table.
- **`note`** (`.`) – record and table endpoints. However, per the Airtable API, the following parameters only apply to *table* endpoints. However again, issuing a command to a record endpoint is essentially the same as issuing a get command to the table endpoint with *offset=<recordID>* and *maxRecords=1*. So, I think I will implement it like that. The upshot, therefore, is that you can restrict the data using the *fields* parameter. You can still send the *filterByFormula*, *pageSize*, and *sort* parameters and they will (I think) have no effect. I'm not sure what will happen if you send a *view* parameter. Probably nothing if the record is included in the view and an error if it is not. We'll see.
- **`fields`** (*`Optional[List[str]]`*) – List of fields to be returned.
- **`filterByFormula`** (*`Optional[str]`*) – A formula for filtering records.
- **`maxRecords`** (*`Optional[int]`*) – Max number of records to return.
- **`pageSize`** (*`Optional[int]`*) – Max number of records to return.

  Airtable restricts this to a maximum of 100, which is also the default.
- **`sort`** (*`Optional[List[Sort]]`*) – List of sort objects specifying how the records should be sorted.

  Since we're using Python here, there's not much reason you can't do this after the fact.
- **`view`** (*`Optional[str]`*) – Name or ID of a view in the table.

  This is a useful shortcut for downloading records based on preset filters and sorts.

- **offset** (*Optional[str]*) – The record ID to start with when downloading.

  This is necessary because of the 100-record page size limit. I can't think of any other use for it. This package should be able to take care of this automatically soon.

Examples

```
>>> from endpoints import TableEndpoint
```

```
>>> e = TableEndpoint(
...     default_base,
...     'Restaurants',
...     )
```

```
>>> g = Get(e)
```

```
>>> g
...
Get(TableEndpoint(base='app...', table='Restaurants'))
```

```
>>> print g
...
Method: GET
URL: https://api.airtable.com/v0/app.../Restaurants
Headers: {'Authorization': 'Bearer key...'}
Params: {}
```

```
>>> g()
...
[{u'records': [{u'createdTime': ...}]}]
```

class firetable.commands.**Patch**(*endpoint*, ***kwargs*)
Bases: *firetable.commands.Command*

Patch command class for Airtable API.

Must be sent to a record endpoint. Updates selected fields on a record. Any field you don't include will be left alone. To update a field that links to other records, get the original list of recordIDs, add/remove as needed, and send the new list.

Parameters

- **endpoint** (Endpoint) – An Endpoint object pointing to a record.
- **fields** (*[Dict]*) – The updated values for the new record.

class firetable.commands.**Post**(*endpoint*, ***kwargs*)
Bases: *firetable.commands.Command*

POST Command for the Airtable API.

Parameters

- **endpoint** (Endpoint) – An Endpoint object pointing to a table.
- **note** (.) – table endpoint. They create a new record in that table.
- **fields** (*Optional[Dict]*) – The initial values for the new record. You can include all, some or none of the fields.
- **typecast** (*Optional[bool]*) – Tell Airtable to do some automatic data conversion. I don't understand the implications yet, but pass it if you wish. Disabled by default.

Examples

```
>>> from endpoints import TableEndpoint
```

```
>>> e = TableEndpoint(
...     default_base,
...     'Restaurants',
...     )
```

```
>>> post = Post(e, fields={
...     'Name': 'Five Guys',
...     'Cost': '$',
...     'Notes': 'The small fry is more than plenty.'
...             })
```

```
>>> post
...
Post(TableEndpoint(base='app...', table='Restaurants'))
```

```
>>> print post.request.url
...
Method: POST
URL: https://api.airtable.com/v0/app.../Restaurants
Headers: {'Content-Length': '0', 'Authorization': 'Bearer key...'}
Params: {'fields': {'Notes': ..., 'Cost': ..., 'Name': ...}}
```

```
>>> try:
...     post()
... except Exception as e:
...     print "Is the internet down?"
...
```

class firetable.commands.**Put**(*endpoint*, ***kwargs*)

Bases: *firetable.commands.Command*

Put command class for Airtable API.

Parameters

- **endpoint** (Endpoint) – An Endpoint object pointing to a record.
- **fields** (*[Dict]*) – The updated values for the record. (All other fields will be obliterated!)
- **typecast** (*Optional[bool]*) – Tell Airtable to do some automatic data conversion. I don't understand the implications yet, but pass it if you wish. Disabled by default.

firetable.commands.**main**()

1.4.2 endpoints

Endpoints classes for Airtable API.

An endpoint is reference to a table or a record.

Creating a Table Endpoint

```
>>> BASE = 'app0123456789abcd'
```

```
>>> contacts = TableEndpoint(
...     base=BASE,
...     table='Contacts'
...     )
```

```
>>> contacts
...
TableEndpoint(base='app0123456789abcd',
    table='Contacts')
```

```
>>> contacts.url
'https://api.airtable.com/v0/app0123456789abcd/Contacts'
```

Creating a Record Endpoint

```
>>> john = RecordEndpoint(
...     base=BASE,
...     table='Contacts',
...     record='rec0123456789abcd'
...     )
```

```
>>> john
...
RecordEndpoint(base='app0123456789abcd',
    table='Contacts',
    record='rec0123456789abcd')
```

```
>>> john.url
'https://api.airtable.com/v0/app0123456789abcd/Contacts/rec0123456789abcd'
```

Sugar

Getting URLs through Endpoint.__str__()

You can also get an endpoint's URL by getting its str representation:

```
>>> print john
https://api.airtable.com/v0/app0123456789abcd/Contacts/rec0123456789abcd
```

Dynamic Endpoint Creation

Instead of instantiating endpoints directly, you can use make_endpoint to get an endpoint whose type is based on the arguments you provide. Combine it with the star notation and endpoint creation becomes very dynamic:

```
>>> a = (BASE, 'Cakes')
```

```
>>> b = (BASE, 'Cakes', 'rec0123456789abcd')
```

```
>>> make_endpoint(*a)
TableEndpoint(base='app0123456789abcd', table='Cakes')
```

```
>>> make_endpoint(*b)
RecordEndpoint(base='app0123456789abcd', table='Cakes', record='rec0123456789abcd')
```

class firetable.endpoints.**Endpoint**
Bases: object

get()

url

class firetable.endpoints.**RecordEndpoint**
Bases: *firetable.endpoints.Endpoint*, *firetable.endpoints.RecordEndpoint*

get()

class firetable.endpoints.**TableEndpoint**
Bases: *firetable.endpoints.Endpoint*, *firetable.endpoints.TableEndpoint*

get(***kwargs*)

firetable.endpoints.**make_endpoint**(*base*, *table*, *record=None*)

firetable.endpoints.**testmod**()

1.4.3 controller

firetable.controller.**delete**()

firetable.controller.**get**(*table='Restaurants'*, ***kwargs*)

firetable.controller.**patch**()

firetable.controller.**post**()

firetable.controller.**put**()

1.4.4 CLI

CLI

Command Line Interface based on click.

Copied from http://chase-seibert.github.io/blog /2014/03/21/python-multilevel-argparse.html. No relation, I don't think. ;)

class firetable.cli.**CLI**(*raw_args*)
Bases: object

delete()

get(*raw_args*)

patch()

post()

put()

firetable.cli.**fire**()

class firetable.cli.**CLI**(*raw_args*)

Bases: object

delete()

get(*raw_args*)

patch()

post()

put()

1.4.5 Airtable API Reference

GET Record

To retrieve an existing record in a table, issue a GET request to the record endpoint. Any "empty" fields (e.g. "", [], or false) in the record will not be returned.

GET Table

fields

[str+]: *list* of *str* objects. Only data for fields whose names are in this list will be included in the records. If you don't need every field, you can use this parameter to reduce the amount of data transferred.

filterByFormula

str: A formula used to filter records. Only records which satisfy the formula will be returned. If combined with view, only records in that view which satisfy the formula will be returned.

maxRecords

Int: The maximum total number of records that will be returned.

pageSize

Int: The number of records returned in each request. Must be less than or equal to 100. Default is 100.

sort

[{field: str, direction: "desc" | "asc"}+]: A list of sort objects that specifies how the records will be ordered. Each sort object must have a field key specifying the name of the field to sort on, and an optional direction key that is either "asc" or "desc". The default direction is "asc". If you set the *view* parameter, the returned records in that view will be sorted by these fields.

view

str: The *name* or *ID* of a view in the Counties table. If set, only the records in that view will be returned. The records will be sorted according to the order of the view.

offset

str: The recordID to start with. The server returns one page of records at a time. Each page will contain pageSize records, which is 100 by default. If there are more records, the response will contain an offset. To fetch the next page of records, include offset in the next request's parameters

Post

To create a new record, issue a POST request to the table endpoint.

You can include all, some, or none of the field values. Returns the created record object if the call succeeded, including a record ID which will uniquely identify the record within Heritage Legal.

Airtable API performs automatic data conversion from string values if typecast parameter is passed in (click for example). Automatic conversion is disabled by default to ensure data integrity, but it may be helpful for integrating with 3rd party data sources.

Patch

To update some (but not all) fields of a record, issue a PATCH request to the record endpoint. Any fields that are not included will not be updated.

To link to new records, add new linked record IDs to the existing array. Be sure to include all existing linked record IDs that you wish to retain. To unlink records, include the existing array of record IDs, excluding any that you wish to unlink.

Delete

To delete a record, issue DELETE request to the record endpoint.

Example Response

```
{
    "deleted": true,
    "id": "rec2FpktzCNhRAkJd"
}
```

1.4.6 firetable

firetable package

Submodules

firetable.commands module

Airtable API command classes.

Copied fairly faithfully from the Airtable API documentation. I wouldn't have known where to begin if it weren't for nicocanali/airtable-python.

Why am I using the command design pattern? I don't know. I don't even know if I'm using it correctly. It seemed like a good idea. I was thinking something about validation methods, eventually, I think.

Some day.

The text width is set to 54, so I can see it comfortably in Pythonista for iOS.

class `firetable.commands.`**`Command`**(*endpoint*, ***kwargs*)
Bases: `object`

`execute`()

class `firetable.commands.`**`Delete`**(*endpoint*, ***kwargs*)
Bases: *`firetable.commands.Command`*

Delete command class for Airtable API.

Parameters **`endpoint`** (`Endpoint`) – An Endpoint object pointing to a record.

class `firetable.commands.`**`Get`**(*endpoint*, ***kwargs*)
Bases: *`firetable.commands.Command`*

Get command class for Airtable API.

Parameters

- **`endpoint`** (`Endpoint`) – An Endpoint object pointing to a record or a table.
- **`note`** (`.`) – record and table endpoints. However, per the Airtable API, the following parameters only apply to *table* endpoints. However again, issuing a command to a record endpoint is essentially the same as issuing a get command to the table endpoint with *offset=<recordID>* and *maxRecords=1*. So, I think I will implement it like that. The upshot, therefore, is that you can restrict the data using the *fields* parameter. You can still send the *filterByFormula*, *pageSize*, and *sort* parameters and they will (I think) have no effect. I'm not sure what will happen if you send a *view* parameter. Probably nothing if the record is included in the view and an error if it is not. We'll see.
- **`fields`** (*`Optional[List[str]]`*) – List of fields to be returned.
- **`filterByFormula`** (*`Optional[str]`*) – A formula for filtering records.
- **`maxRecords`** (*`Optional[int]`*) – Max number of records to return.
- **`pageSize`** (*`Optional[int]`*) – Max number of records to return.

  Airtable restricts this to a maximum of 100, which is also the default.
- **`sort`** (*`Optional[List[Sort]]`*) – List of sort objects specifying how the records should be sorted.

  Since we're using Python here, there's not much reason you can't do this after the fact.

- **view** (*Optional[str]*) – Name or ID of a view in the table.

  This is a useful shortcut for downloading records based on preset filters and sorts.

- **offset** (*Optional[str]*) – The record ID to start with when downloading.

  This is necessary because of the 100-record page size limit. I can't think of any other use for it. This package should be able to take care of this automatically soon.

Examples

```
>>> from endpoints import TableEndpoint
```

```
>>> e = TableEndpoint(
...     default_base,
...     'Restaurants',
...     )
```

```
>>> g = Get(e)
```

```
>>> g
...
Get(TableEndpoint(base='app...', table='Restaurants'))
```

```
>>> print g
...
Method: GET
URL: https://api.airtable.com/v0/app.../Restaurants
Headers: {'Authorization': 'Bearer key...'}
Params: {}
```

```
>>> g()
...
[{u'records': [{u'createdTime': ...}]}]
```

class firetable.commands.**Patch**(*endpoint*, ***kwargs*)
Bases: *firetable.commands.Command*

Patch command class for Airtable API.

Must be sent to a record endpoint. Updates selected fields on a record. Any field you don't include will be left alone. To update a field that links to other records, get the original list of recordIDs, add/remove as needed, and send the new list.

Parameters

- **endpoint** (Endpoint) – An Endpoint object pointing to a record.
- **fields** (*[Dict]*) – The updated values for the new record.

class firetable.commands.**Post**(*endpoint*, ***kwargs*)
Bases: *firetable.commands.Command*

POST Command for the Airtable API.

Parameters

- **endpoint** (Endpoint) – An Endpoint object pointing to a table.
- **note** (.) – table endpoint. They create a new record in that table.

- **fields** (*Optional[Dict]*) – The initial values for the new record. You can include all, some or none of the fields.
- **typecast** (*Optional[bool]*) – Tell Airtable to do some automatic data conversion. I don't understand the implications yet, but pass it if you wish. Disabled by default.

Examples

```
>>> from endpoints import TableEndpoint
```

```
>>> e = TableEndpoint(
...     default_base,
...     'Restaurants',
...     )
```

```
>>> post = Post(e, fields={
...     'Name': 'Five Guys',
...     'Cost': '$',
...     'Notes': 'The small fry is more than plenty.'
...             })
```

```
>>> post
...
Post(TableEndpoint(base='app...', table='Restaurants'))
```

```
>>> print post.request.url
...
Method: POST
URL: https://api.airtable.com/v0/app.../Restaurants
Headers: {'Content-Length': '0', 'Authorization': 'Bearer key...'}
Params: {'fields': {'Notes': ..., 'Cost': ..., 'Name': ...}}
```

```
>>> try:
...     post()
... except Exception as e:
...     print "Is the internet down?"
...
```

class firetable.commands.**Put**(*endpoint*, ***kwargs*)
Bases: *firetable.commands.Command*

Put command class for Airtable API.

Parameters

- **endpoint** (Endpoint) – An Endpoint object pointing to a record.
- **fields** (*[Dict]*) – The updated values for the record. (All other fields will be obliterated!)
- **typecast** (*Optional[bool]*) – Tell Airtable to do some automatic data conversion. I don't understand the implications yet, but pass it if you wish. Disabled by default.

firetable.commands.**main**()

firetable.endpoints module

Endpoints classes for Airtable API.

An endpoint is reference to a table or a record.

Creating a Table Endpoint

```
>>> BASE = 'app0123456789abcd'
```

```
>>> contacts = TableEndpoint(
...     base=BASE,
...     table='Contacts'
...     )
```

```
>>> contacts
...
TableEndpoint(base='app0123456789abcd',
    table='Contacts')
```

```
>>> contacts.url
'https://api.airtable.com/v0/app0123456789abcd/Contacts'
```

Creating a Record Endpoint

```
>>> john = RecordEndpoint(
...     base=BASE,
...     table='Contacts',
...     record='rec0123456789abcd'
...     )
```

```
>>> john
...
RecordEndpoint(base='app0123456789abcd',
    table='Contacts',
    record='rec0123456789abcd')
```

```
>>> john.url
'https://api.airtable.com/v0/app0123456789abcd/Contacts/rec0123456789abcd'
```

Sugar

Getting URLs through Endpoint.__str__() You can also get an endpoint's URL by getting its str representation:

```
>>> print john
https://api.airtable.com/v0/app0123456789abcd/Contacts/rec0123456789abcd
```

Dynamic Endpoint Creation Instead of instantiating endpoints directly, you can use make_endpoint to get an endpoint whose type is based on the arguments you provide. Combine it with the star notation and endpoint creation becomes very dynamic:

```
>>> a = (BASE, 'Cakes')
```

```
>>> b = (BASE, 'Cakes', 'rec0123456789abcd')
```

```
>>> make_endpoint(*a)
TableEndpoint(base='app0123456789abcd', table='Cakes')
```

```
>>> make_endpoint(*b)
RecordEndpoint(base='app0123456789abcd', table='Cakes', record='rec0123456789abcd')
```

class firetable.endpoints.**Endpoint**
Bases: object

get()

url

class firetable.endpoints.**RecordEndpoint**
Bases: *firetable.endpoints.Endpoint*, *firetable.endpoints.RecordEndpoint*

get()

class firetable.endpoints.**TableEndpoint**
Bases: *firetable.endpoints.Endpoint*, *firetable.endpoints.TableEndpoint*

get(***kwargs*)

firetable.endpoints.**make_endpoint**(*base*, *table*, *record=None*)

firetable.endpoints.**testmod**()

firetable.cli module

CLI

Command Line Interface based on click. Copied from http://chase-seibert.github.io/blog /2014/03/21/python-multilevel-argparse.html. No relation, I don't think. ;)

class firetable.cli.**CLI**(*raw_args*)
Bases: object

delete()

get(*raw_args*)

patch()

post()

put()

firetable.cli.**fire**()

Module contents

tests

firetable.tests.**test_get**()

1.4.7 Indices and tables

- genindex
- modindex
- search

CHAPTER 2

Firetable

2.1 A CLI tool and Python library for Airtable

2.1.1 Introduction

Airtable is the best, but sometimes (nay, *oftentimes*) a web interface is not the best for a workflow. Airtable provides a handy and comprehensive API for downloading and uploading JSON data. Firetable is meant to bridge Airtable and the command line with:

1. a git-esque command line tool that supports standard UNIX pipes' and
2. a Python library for using the API directly.

It's called Firetable to preserve the namespace rights in case the Airtable folks want to hand down a blessed airtable.py package someday, and because I hope to eventually expand this package beyond the simple API stuff into integration with pandas, reportlab, etc.

2.1.2 Things that are Working Right Now

1. Download a complete table
2. Download a table view
3. Download a table restricted to specified fields

2.1.3 Up Next

1. Add support for *sort* and *filterByFormula* parameters.
2. Add support for POST requests.
3. Editing
   (a) Download data and convert to YAML or Markdown.
   (b) Open temp file in default text editor.
   (c) Convert back to JSON and upload data.

2.1.4 Installation

Install firetable by running:

```
pip install firetable
```

2.1.5 Disclaimer

I am an amateur. Pull requests welcome and use at your own risk!

2.1.6 Contribute

- Issue Tracker: github.com/davidseibert/firetable/issues
- Source Code: github.com/davidseibert/firetable

2.1.7 License

The project is licensed under the MIT license.

f