pyAirtable

Gui Talarico

Aug 08, 2021

DOCS

Python Client for the Airtable Api

Docs GitHub

Latest Release: 1.0.0.dev1

CHAPTER

ONE

GETTING STARTED

1.1 Installation

```
$ pip install pyairtable
```

Warning: Looking for `airtable-python-wrapper`?

1.2 Api Key

Your Api Key should be kept secure and should likely not be saved in your code. A common way to store and used it in your code is to save the key in your environment and load it

```
import os
api_key = os.environ["AIRTABLE_API_KEY"]
```

1.3 Quickstart

The easiest way to use this client is to the *Table* class to fetch or update your records:

```
>>> import os
>>> from pyairtable import Table
>>> api_key = os.environ['AIRTABLE_API_KEY']
>>> table = Table('base_id', 'base_id', api_key)
>>> table.all()
[ {"id": "rec5eR7IzKSAOBHCz", "fields": { ... }}]
>>> table.create({"Foo": "Bar"})
{"id": "recwAcQdqwe21as", "fields": { "Foo": "Bar" }}]
>>> table.update("recwAcQdqwe21as", {"Foo": "Foo"})
{"id": "recwAcQdqwe21as", "fields": { "Foo": "Foo" }}]
>>> table.delete("recwAcQdqwe21as")
True
```

For more details on all the available classes and methods checkout the *Airtable Api* section.

CHAPTER

TWO

AIRTABLE API

2.1 Overview

This client offers three classes you can use to access the Airtable Api:

- *Table* - represents an Airtable **Table**
- *Base* - represents an Airtable **Base**
- *Api* - represents an Airtable **Api**

The interfaces of these are nearly identical, the main difference is if `base_id` and `table_id` are provided on initialization or on calls.

For example, the three `all()` calls below would return the same result:

```
from pyairtable import Api, Base, Table

api = Api('apikey')
api.all('base_id', 'table_name')

base = Base('apikey', 'base_id')
base.all('table_name')

table = Table('apikey', 'table_name', 'base_id')
table.all()
```

2.2 Interface

The table below shows a comparison of the methods used in the library compared with the official API equivalent.

Table 1: pyAirtable Api

Description	pyAirtable	Airtable Api
Retrieve a single Record	`get()`	`GET baseId/recordId`
Iterate over record pages	`iterate()`	`GET baseId/`
Get all records	`all()`	`GET baseId/`
Get all matches	`all(formula=match(...)`	`GET baseId/?filterByFormula=...`
Get first match	`first(formula=match(...)`	`GET baseId/?filterByFormula=... &maxRecords=1`
Create record	`create()`	`POST baseId/`
Update a record	`update()`	`PUT baseId/`
Replace a record	`update(replace=True)`	`PATCH base/`
Delete a record	`delete()`	`DELETE baseId/`

2.3 Examples

Examples below use the *`Table`* Api for conciseness - all methods are available for all three interfaces (`Api`, `Base`, and `Table`).

2.3.1 Fetching Records

`iterate()`

Iterate over a set of records of size `page_size`, up until `max_records` or end of table, whichever is shorter.

```
>>> for records in table.iterate(page_size=100, max_records=1000):
...     print(records)
[{id:'rec123asa23', fields': {'Last Name': 'Alfred', "Age": 84}, ...}, ... ]
[{id:'rec123asa23', fields': {'Last Name': 'Jameson', "Age": 42}, ...}, ... ]
```

`all()`

This method returns a single list with all records in a table. Note that under the hood it uses *`iterate()`* to fetch records so multiple requests might be made.

```
>>> table.all(sort=["First Name", "-Age"]):
[{id:'rec123asa23', fields': {'Last Name': 'Alfred', "Age": 84}, ...}, ... ]
```

2.3.2 Creating Records

create()

Creates a single record from a dictionary representing the table's fields.

```
>>> table.create({'First Name': 'John'})
{id:'rec123asa23', fields': {'First Name': 'John', ...}}
```

batch_create()

Batch create records from a list of dictionaries representing the table's fields.

```
>>> table.batch_create([{'First Name': 'John'}, ...])
[{id:'rec123asa23', fields': {'First Name': 'John', ...}}, ...]
```

2.3.3 Updating Records

update()

Updates a single record for the provided `record_id` using a dictionary representing the table's fields.

```
>>> table.update('recwPQIfs4wKPyc9D', {"Age": 21})
[{id:'recwPQIfs4wKPyc9D', fields': {"First Name": "John", "Age": 21, ...}}, ...]
```

batch_update()

Batch update records from a list of records.

```
>>> table.batch_update([{"id": "recwPQIfs4wKPyc9D", "fields": {"First Name": "Matt"}}, ..
↪.])
[{id:'recwPQIfs4wKPyc9D', fields': {"First Name": "Matt", "Age": 21, ...}}, ...]
```

2.3.4 Deleting Records

delete()

Deletes a single record using the provided `record_id`.

```
>>> table.delete('recwPQIfs4wKPyc9D')
{ "deleted": True, ... }
```

batch_delete()

Batch delete records using a list of record ids.

```
>>> table.batch_delete(['recwPQIfs4wKPyc9D', 'recwAcQdqwe21as'])
[  { "deleted": True, ... }, ... ]
```

2.3.5 Return Values

Return Values: when records are returned, will most often be alist of Airtable records (dictionary) in a format as shown below.

```
>>> table.all()
... [{
...     "records": [
...         {
...             "id": "recwPQIfs4wKPyc9D",
...             "fields": {
...                 "COLUMN_ID": "1",
...             },
...             "createdTime": "2017-03-14T22:04:31.000Z"
...         },
...         {
...             "id": "rechOLltN9SpPHq5o",
...             "fields": {
...                 "COLUMN_ID": "2",
...             },
...             "createdTime": "2017-03-20T15:21:50.000Z"
...         },
...         {
...             "id": "rec5eR7IzKSAOBHCz",
...             "fields": {
...                 "COLUMN_ID": "3",
...             },
...             "createdTime": "2017-08-05T21:47:52.000Z"
...         }
...     ],
...     "offset": "rec5eR7IzKSAOBHCz"
... }, ... ]
```

The *Base* class is similar to *Table*, the main difference is that . *table_name* is not provided during initialization. Instead, it can be specified on each request.

```
>>> base = Base('appEioitPbxI72w06', 'apikey')
>>> base.all('Contacts)
[{id:'rec123asa23', fields': {'Last Name': 'Alfred', "Age": 84}, ... ]
```

2.4 Classes

2.4.1 Api

New in version 1.0.0.

class pyairtable.api.**Api**(*api_key*, *timeout=None*)

Represents an Airtable Api.

The Api Key is provided on init and `base_id` and `table_id` can be provided on each method call.

If you are only operating on one Table, or one Base, consider using *Base* or *Table*.

Usage:

```
>>> api = Api('apikey')
>>> api.all('base_id', 'table_name')
```

__init__(*api_key*, *timeout=None*)

Parameters **api_key** (str) – An Airtable API Key.

Keyword Arguments **timeout** (Tuple) – A tuple indicating a connect and read timeout. eg. `timeout=(2,5)` would configure a 2 second timeout for the connection to be established and 5 seconds for a server read timeout. Default is `None` (no timeout).

all(*base_id*, *table_name*, ***options*)

Retrieves all records repetitively and returns a single list.

```
>>> api.all('base_id', 'table_name', view='MyView', fields=['ColA', '-ColB'])
[{'fields': ... }, ...]
>>> api.all('base_id', 'table_name', maxRecords=50)
[{'fields': ... }, ...]
```

Parameters

- **base_id** (str) – An Airtable base id.
- **table_name** (str) – An Airtable table name. Table name should be unencoded, as shown on browser.

Keyword Arguments

- **view** – The name or ID of a view. If set, only the records in that view will be returned. The records will be sorted according to the order of the view.
- **page_size** – The number of records returned in each request. Must be less than or equal to 100. Default is 100.
- **max_records** – The maximum total number of records that will be returned. If this value is larger than *page_size* multiple requests will be needed to fetch all records.
- **fields** – Name of field or fields to be retrieved. Default is all fields. 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.
- **sort** – List of fields to sort by. Default order is ascending. This parameter specifies how the records will be ordered. If you set the view parameter, the returned records in that view will be sorted by these fields. If sorting by multiple columns, column names can be passed as a list. Sorting Direction is ascending by default, but can be reversed by prefixing the column name with a minus sign `-`.
- **formula** – An Airtable formula. The formula will be evaluated for each record, and if the result is not 0, false, "", NaN, [], or #Error! the record will be included in the response. If combined with view, only records in that view which satisfy the formula will be returned. For example, to only include records where `COLUMN_A` isn't empty, pass in: `"NOT({COLUMN_A}='')"`.

Returns List of Records

Return type records (list)

```
>>> records = all(maxRecords=3, view='All')
```

batch_create(*base_id*, *table_name*, *records*, *typecast=False*)

Breaks records into chunks of 10 and inserts them in batches. Follows the set API rate. To change the rate limit you can change `API_LIMIT = 0.2` (5 per second)

```
>>> records = [{'Name': 'John'}, {'Name': 'Marc'}]
>>> api.batch_insert('base_id', 'table_name', records)
```

Parameters

- **base_id** (`str`) – An Airtable base id.
- **table_name** (`str`) – An Airtable table name. Table name should be unencoded, as shown on browser.
- **records** (`List[dict]`) – List of dictionaries representing records to be created.

Keyword Arguments **typecast** – The Airtable API will perform best-effort automatic data conversion from string values. Default is False.

Returns list of added records

Return type records (`list`)

batch_delete(*base_id*, *table_name*, *record_ids*)

Breaks records into batches of 10 and deletes in batches, following set API Rate Limit (5/sec). To change the rate limit set value of `API_LIMIT` to the time in seconds it should sleep before calling the function again.

```
>>> record_ids = ['recwPQIfs4wKPyc9D', 'recwDxIfs3wDPyc3F']
>>> api.batch_delete('base_id', 'table_name', records_ids)
```

Parameters

- **base_id** (`str`) – An Airtable base id.
- **table_name** (`str`) – An Airtable table name. Table name should be unencoded, as shown on browser.
- **record_ids** (`list`) – Record Ids to delete

Returns list of records deleted

Return type records(`list`)

batch_update(*base_id*, *table_name*, *records*, *replace=False*, *typecast=False*)

Updates a records by their record id's in batch.

Parameters

- **base_id** (`str`) – An Airtable base id.
- **table_name** (`str`) – An Airtable table name. Table name should be unencoded, as shown on browser.
- **records** (`list`) – List of dict: [{"id": record_id, "field": fields_to_update_dict}]

Keyword Arguments

- **replace** (bool, optional) – If True, record is replaced in its entirety by provided fields - eg. if a field is not included its value will bet set to null. If False, only provided fields are updated. Default is False.
- **typecast** – The Airtable API will perform best-effort automatic data conversion from string values. Default is False.

Returns list of updated records

Return type records(list)

create(*base_id*, *table_name*, *fields*, *typecast=False*)

Creates a new record

```
>>> record = {'Name': 'John'}
>>> api.create('base_id', 'table_name', record)
```

Parameters

- **base_id** (str) – An Airtable base id.
- **table_name** (str) – An Airtable table name. Table name should be unencoded, as shown on browser.
- **fields** (dict) – Fields to insert. Must be dictionary with Column names as Key.

Keyword Arguments **typecast** – The Airtable API will perform best-effort automatic data conversion from string values. Default is False.

Returns Inserted record

Return type record (dict)

delete(*base_id*, *table_name*, *record_id*)

Deletes a record by its id

```
>>> record = api.match('base_id', 'table_name', 'Employee Id', 'DD13332454')
>>> api.delete('base_id', 'table_name', record['id'])
```

Parameters

- **base_id** (str) – An Airtable base id.
- **table_name** (str) – An Airtable table name. Table name should be unencoded, as shown on browser.
- **record_id** (str) – An Airtable record id.

Returns Deleted Record

Return type record (dict)

first(*base_id*, *table_name*, ***options*)

Retrieves the first found record or None if no records are returned.

This is similar to all(), except it it sets page_size and max_records to 1 to optimize query.

Parameters

- **base_id** (str) – An Airtable base id.
- **table_name** (str) – An Airtable table name. Table name should be unencoded, as shown on browser.

get(*base_id*, *table_name*, *record_id*)
Retrieves a record by its id

```
>>> record = api.get('base_id', 'table_name', 'recwPQIfs4wKPyc9D')
```

Parameters

- **base_id** (str) – An Airtable base id.
- **table_name** (str) – An Airtable table name. Table name should be unencoded, as shown on browser.
- **record_id** (str) – An Airtable record id.

Returns Record

Return type record

get_record_url(*base_id*, *table_name*, *record_id*)
Returns a url for the provided record

Parameters

- **base_id** (str) – An Airtable base id.
- **table_name** (str) – An Airtable table name. Table name should be unencoded, as shown on browser.

iterate(*base_id*, *table_name*, ***options*)
Record Retriever Iterator

Returns iterator with lists in batches according to pageSize. To get all records at once use *all()*

```
>>> for page in api.iterate('base_id', 'table_name'):
...     for record in page:
...         print(record)
{"id": ... }
...
```

Parameters

- **base_id** (str) – An Airtable base id.
- **table_name** (str) – An Airtable table name. Table name should be unencoded, as shown on browser.

Keyword Arguments

- **view** – The name or ID of a view. If set, only the records in that view will be returned. The records will be sorted according to the order of the view.
- **page_size** – The number of records returned in each request. Must be less than or equal to 100. Default is 100.
- **max_records** – The maximum total number of records that will be returned. If this value is larger than *page_size* multiple requests will be needed to fetch all records.
- **fields** – Name of field or fields to be retrieved. Default is all fields. 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.

- **sort** – List of fields to sort by. Default order is ascending. This parameter specifies how the records will be ordered. If you set the view parameter, the returned records in that view will be sorted by these fields. If sorting by multiple columns, column names can be passed as a list. Sorting Direction is ascending by default, but can be reversed by prefixing the column name with a minus sign `-`.
- **formula** – An Airtable formula. The formula will be evaluated for each record, and if the result is not 0, false, "", NaN, [], or #Error! the record will be included in the response. If combined with view, only records in that view which satisfy the formula will be returned. For example, to only include records where `COLUMN_A` isn't empty, pass in: `"NOT({COLUMN_A}='')"`.

Returns Record Iterator, grouped by page size

Return type iterator

update(*base_id*, *table_name*, *record_id*, *fields*, *replace=False*, *typecast=False*)

Updates a record by its record id. Only Fields passed are updated, the rest are left as is.

```
>>> table.update('recwPQIfs4wKPyc9D', {"Age": 21})
{id:'recwPQIfs4wKPyc9D', fields': {"First Name": "John", "Age": 21}}
>>> table.update('recwPQIfs4wKPyc9D', {"Age": 21}, replace=True)
{id:'recwPQIfs4wKPyc9D', fields': {"Age": 21}}
```

Parameters

- **base_id** (`str`) – An Airtable base id.
- **table_name** (`str`) – An Airtable table name. Table name should be unencoded, as shown on browser.
- **record_id** (`str`) – An Airtable record id.
- **fields** (`dict`) – Fields to update. Must be dictionary with Column names as Key

Keyword Arguments

- **replace** (`bool`, optional) – If `True`, record is replaced in its entirety by provided fields - eg. if a field is not included its value will bet set to null. If False, only provided fields are updated. Default is `False`.
- **typecast** – The Airtable API will perform best-effort automatic data conversion from string values. Default is False.

Returns Updated record

Return type record (`dict`)

2.4.2 Base

New in version 1.0.0.

class `pyairtable.api.`**Base**(*api_key*, *base_id*, *timeout=None*)

Represents an Airtable Base. This calss is similar to *Api*, except `base_id` is provided on init instead of provided on each method call.

Usage:

```
>>> base = Base('apikey', 'base_id')
>>> base.all()
```

__init__(*api_key*, *base_id*, *timeout=None*)

Parameters

- **api_key** (str) – An Airtable API Key.
- **base_id** (str) – An Airtable base id.

Keyword Arguments **timeout** (Tuple) – A tuple indicating a connect and read timeout. eg. `timeout=(2,5)` would configure a 2 second timeout for the connection to be established and 5 seconds for a server read timeout. Default is `None` (no timeout).

all(*table_name*, ***options*)
Same as *Api.all* but without `base_id` arg.

batch_create(*table_name*, *records*, *typecast=False*)
Same as *Api.batch_create* but without `base_id` arg.

batch_delete(*table_name*, *record_ids*)
Same as *Api.batch_delete* but without `base_id` arg.

batch_update(*table_name*, *records*, *replace=False*, *typecast=False*)
Same as *Api.batch_update* but without `base_id` arg.

create(*table_name*, *fields*, *typecast=False*)
Same as *Api.create* but without `base_id` arg.

delete(*table_name*, *record_id*)
Same as *Api.delete* but without `base_id` arg.

first(*table_name*, ***options*)
Same as *Api.first* but without `base_id` arg.

get(*table_name*, *record_id*)
Same as *Api.get* but without `base_id` arg.

get_record_url(*table_name*, *record_id*)
Same as *Api.get_record_url* but without `base_id` arg.

iterate(*table_name*, ***options*)
Same as *Api.iterate* but without `base_id` arg.

update(*table_name*, *record_id*, *fields*, *replace=False*, *typecast=False*)
Same as *Api.update* but without `base_id` arg.

2.4.3 Table

New in version 1.0.0.

class pyairtable.api.**Table**(*api_key*, *base_id*, *table_name*, *, *timeout=None*)
Represents an Airtable Table. This calss is similar to *Api*, except `base_id` and `table_id` are provided on init instead of provided on each method call.

Usage:

```
>>> table = Table('apikey', 'base_id', 'table_name')
>>> table.all()
```

__init__(*api_key*, *base_id*, *table_name*, *, *timeout=None*)

Parameters

- **api_key** (str) – An Airtable API Key.
- **base_id** (str) – An Airtable base id.
- **table_name** (str) – An Airtable table name. Table name should be unencoded, as shown on browser.

Keyword Arguments **timeout** (Tuple) – A tuple indicating a connect and read timeout. eg. `timeout=(2,5)` would configure a 2 second timeout for the connection to be established and 5 seconds for a server read timeout. Default is `None` (no timeout).

all(***options*)
Same as *Api.all* but without base_id and table_name arg.

batch_create(*records*, *typecast=False*)
Same as *Api.batch_create* but without base_id and table_name arg.

batch_delete(*record_ids*)
Same as *Api.batch_delete* but without base_id and table_name arg.

batch_update(*records*, *replace=False*, *typecast=False*)
Same as *Api.batch_update* but without base_id and table_name arg.

create(*fields*, *typecast=False*)
Same as *Api.create* but without base_id and table_name arg.

delete(*record_id*)
Same as *Api.delete* but without base_id and table_name arg.

first(***options*)
Same as *Api.first* but without base_id and table_name arg.

get(*record_id*)
Same as *Api.get* but without base_id and table_name arg.

get_record_url(*record_id*)
Same as *Api.get_record_url* but without base_id and table_name arg.

iterate(***options*)
Same as *Api.iterate* but without base_id and table_name arg.

property table_url
Returns the table URL

update(*record_id*, *fields*, *replace=False*, *typecast=False*)
Same as *Api.update* but without base_id and table_name arg.

2.5 Parameters

Airtable offers a variety of options to control how you fetch data.

Most options in the Airtable Api (eg. `sort`, `fields`, etc) have a corresponding `kwargs` that can be used with fetching methods like *iterate()*.

Table 2: Title

Parameter	Airtable Option	Notes
`max_records`	`maxRecords`	The maximum total number of records that will be returned. If this value is larger than *page_size* multiple requests will be needed to fetch all records.
`sort`	`sort`	List of fields to sort by. Default order is ascending. This parameter specifies how the records will be ordered. If you set the view parameter, the returned records in that view will be sorted by these fields. If sorting by multiple columns, column names can be passed as a list. Sorting Direction is ascending by default, but can be reversed by prefixing the column name with a minus sign `-`.
`view`	`view`	The name or ID of a view. If set, only the records in that view will be returned. The records will be sorted according to the order of the view.
`page_size`	`pageSize`	The number of records returned in each request. Must be less than or equal to 100. Default is 100.
`formula`	`filterByFormula`	An Airtable formula. The formula will be evaluated for each record, and if the result is not 0, false, "", NaN, [], or #Error! the record will be included in the response. If combined with view, only records in that view which satisfy the formula will be returned. For example, to only include records where `COLUMN_A` isn't empty, pass in: `"NOT({COLUMN_A}='')"`.
`fields`	`fields`	Name of field or fields to be retrieved. Default is all fields. 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.

2.6 Formulas

The formula module provides funcionality to help you compose airtable formulas. For more information see Airtable Formula Reference

```
>>> table = Table("apikey", "base_id", "Contact")
>>> formula = match({"First Name": "John", "Age": 21})
>>> table.first(formula=formula)
{"id": "recUwKa6lbNSMsetH", "fields": {"First Name": "John", "Age": 21}}
>>> formula
"AND({First Name}='John',{Age}=21)"
```

pyairtable.formulas.**match**(*dict_values*)

Creates one or more `EQUAL()` expressions for each provided dict value. If more than one assetions is included, the expressions are groupped together into using `AND()`.

This function also handles escaping field names and casting python values to the appropriate airtable types.

Parameters **dict_values** – dictionary containing column names and values

Usage:

```
>>> match({"First Name": "John", "Age": 21})
"AND({First Name}='John',{Age}=21)"
```

2.6.1 Raw Formulas

New in version 1.0.0.

This module also includes many lower level functions you can use if you want to compose formulas:

pyairtable.formulas.**EQUAL**(*left*, *right*)

Creates an equality assertion

```
>>> EQUAL(2, 2)
'2=2'
```

Return type str

pyairtable.formulas.**FIELD**(*name*)

Creates a reference to a field

Parameters **name** (str) – field name

Usage:

```
>>> FIELD("First Name"")
'{First Name}'
```

Return type str

pyairtable.formulas.**AND**(**args*)

Creates an AND Statement

```
>>> AND(1, 2, 3)
'AND(1, 2, 3)'
```

Return type str

CHAPTER

THREE

ORM

New in version 1.0.0.

Warning: This feature is experimental.

3.1 Model

The *Model* class allows you create an orm-style class for your Airtable tables.

```
>>> from pyairtable.orm import Model, fields
>>> class Contact(Model):
...     first_name = fields.TextField("First Name")
...     last_name = fields.TextField("Last Name")
...     email = fields.EmailField("Email")
...     is_registered = fields.CheckboxField("Registered")
...     partner = fields.LinkField("Partner", "Contact", lazy=False)
...
...     class Meta:
...         base_id = "appaPqizdsNHDvlEm"
...         table_name = "Contact"
...         api_key = "keyapikey"
```

Once you have a class, you can create new objects to represent your Airtable records. Call *save()* to create a new record.

```
>>> contact = Contact(
...     first_name="Mike",
...     last_name="McDonalds",
...     email="mike@mcd.com",
...     is_registered=False
... )
...
>>> assert contact.id is None
>>> contact.exists()
False
>>> assert contact.save()
>>> contact.exists()
True
```

(continues on next page)

(continued from previous page)

```
>>> contact.id
rec123asa23
```

You can read and modify attributes. If record already exists, *save()* will update the record:

```
>>> assert contact.is_registered is False
>>> contact.is_registered = True
>>> contact.save()
>>> assert contact.is_registered = True
>>> contact.to_record()
{
    "id": recS6qSLw0OCA6Xul",
    "createdTime": "2021-07-14T06:42:37.000Z",
    "fields": {
        "First Name": "Mike",
        "Last Name": "McDonalds",
        "Email": "mike@mcd.com",
        "Resgistered": True
    }
}
```

And you can use *delete()* to delete the record:

```
>>> contact.delete()
True
```

class pyairtable.orm.model.**Model**(***fields*)

This class allows you create an orm-style class for your Airtable tables.

This is a meta class and can only be used to define sub-classes.

The `Meta` is reuired and must specify all three attributes: `base_id`, `table_id`, and `api_key`.

```
>>> from pyairtable.orm import Model, fields
>>> class Contact(Model):
...     first_name = fields.TextField("First Name")
...     age = fields.IntegerField("Age")
...
...     class Meta
...         base_id = "appaPqizdsNHDvlEm"
...         table_name = "Contact"
...         api_key = "keyapikey"
```

class Meta

api_key: **str**

base_id: **str**

table_name: **str**

timeout: **Optional[Tuple[int, int]]**

typecast: **bool**

created_time: **str = ''**

delete()
Deleted record. Must have 'id' field

Return type bool

exists()
Returns boolean indicating if instance exists (has 'id' attribute)

Return type bool

fetch()
Fetches field and resets instance field values from pyairtable record

classmethod from_id(*record_id*, *fetch=True*)
Create an instance from a *record_id*

Parameters **record_id** (str) – An Airtable record id.

Keyward Args:

fetch: If *True*, record will be fetched from pyairtable and fields will be updated. If *False*, a new instance is created with the provided *id*, but field values are unset. Default is *True*.

Returns Instance of model

Return type (Model)

classmethod from_record(*record*)
Create instance from record dictionary

Return type ~T

classmethod get_table()
Return Airtable *Table* class instance

Return type *Table*

id: str = ''

save()
Saves or updates a model. If instance has no 'id', it will be created, otherwise updatedself.

Returns *True* if was created and *False* if it was updated

Return type bool

to_record()

Return type dict

3.2 Fields

Field classes are used to define the the data type of your Airtable columns.

Internally these are implemented as descritors, so they can access and set values seamleslly.

Descriptors are also annotated so you can use them with mypy.

```
>>> contact.to_record()
{
    "id": recS6qSLw0OCA6Xul",
    "createdTime": "2021-07-14T06:42:37.000Z",
    "fields": {
        "First Name": "George",
        "Age": 20,
    }
}
```

3.2.1 Link Fields

In addition to standard data type fields, the *LinkField* class offers a special behaviour that can fetch related records.

In other words, you can transverse related records through their `Link Fields`:

```
>>> contact.partner.first_name
```

---

class pyairtable.orm.fields.**CheckboxField**(*field_name*)
Airtable Checkbox field. Uses `bool` to store value

class pyairtable.orm.fields.**DateField**(*field_name*)
Airtable Date field. Uses `Date` to store value

static to_internal_value(*value*)
Airtable returns ISO 8601 date string eg. "2014-09-05

Return type `date`

static to_record_value(*value*)
Airtable expects ISO 8601 date string eg. "2014-09-05

Return type `str`

class pyairtable.orm.fields.**DatetimeField**(*field_name*)
Airtable Datetime field. Uses `datetime` to store value

static to_internal_value(*value*)
Airtable returns ISO 8601 string datetime eg. "2014-09-05T07:00:00.000Z"

Return type `datetime`

static to_record_value(*value*)
Airtable expects ISO 8601 string datetime eg. "2014-09-05T07:00:00.000Z"

Return type `str`

class pyairtable.orm.fields.**EmailField**(*field_name*)
Airtable Email field. Uses `str` to store value

class pyairtable.orm.fields.**FloatField**(*field_name*)
Airtable Number field with Decimal precision. Uses `float` to store value

class pyairtable.orm.fields.**IntegerField**(*field_name*)
Airtable Number field with Integer Precision. Uses `int` to store value

class pyairtable.orm.fields.**LinkField**(*field_name*, *model*, *lazy=True*)
Airtable Link field. Uses `List[Model]` to store value

__init__(*field_name*, *model*, *lazy=True*)

Parameters

- **field_name** (str) – Name of Airtable Column
- **model** (Union[str, Type[~T_Linked]]) – Model of Linked Type. Must be subtype of Model
- **lazy** – Use *True* to load linked model when looking up attribute. *False* will create empty object with only *id* but will not fetch fields.

Usage:

```
>>> TODO
```

class pyairtable.orm.fields.**TextField**(*field_name*)

Airtable Single Text or Multiline Text Fields. Uses str to store value

CHAPTER

FOUR

METADATA API

The metadata api gives you the ability to list all of your bases, tables, fields, and views.

Warning: If you want to develop an integration using the Metadata API, you must register here for access and to receive a client secret. Enterprise Airtable accounts do not require a separate Metadata API client secret.

pyairtable.metadata.**get_api_bases**(*api*)

Return list of Bases from an Api or Base instance. For More Details Metadata Api Documentation

Parameters **api** (Union[*Api*, *Base*]) – Api or Base instance

Usage:

```
>>> table.get_bases()
    {
        "bases": [
            {
                "id": "appY3WxIBCdKPDdIa",
                "name": "Apartment Hunting",
                "permissionLevel": "create"
            },
            {
                "id": "appSW9R5uCNmRmfl6",
                "name": "Project Tracker",
                "permissionLevel": "edit"
            }
        ]
    }
```

Return type dict

pyairtable.metadata.**get_base_schema**(*base*)

Returns Schema of a Base For More Details Metadata Api Documentation

Parameters **base** (Union[*Base*, *Table*]) – Base or Table instance

Usage:

```
>>> get_base_schema(base)
    {
        "tables": [
```

(continues on next page)

(continued from previous page)

```
            {
                "id": "tbltp8DGLhqbUmjK1",
                "name": "Apartments",
                "primaryFieldId": "fld1VnoyuotSTyxW1",
                "fields": [
                    {
                        "id": "fld1VnoyuotSTyxW1",
                        "name": "Name",
                        "type": "singleLineText"
                    },
                    {
                        "id": "fldoaIqdn5szURHpw",
                        "name": "Pictures",
                        "type": "multipleAttachment"
                    },
                    {
                        "id": "fldumZe00w09RYTW6",
                        "name": "District",
                        "type": "multipleRecordLinks"
                    }
                ],
                "views": [
                    {
                        "id": "viwQpsuEDqHFqegkp",
                        "name": "Grid view",
                        "type": "grid"
                    }
                ]
            }
        ]
    }
```

Return type dict

pyairtable.metadata.**get_table_schema**(*table*)

Returns the specific table schema record provided by base schema list

Parameters **table** (*Table*) – Table instance

Usage:

```
>>> get_table_schema(table)
{
    "id": "tbltp8DGLhqbUmjK1",
    "name": "Apartments",
    "primaryFieldId": "fld1VnoyuotSTyxW1",
    "fields": [
        {
            "id": "fld1VnoyuotSTyxW1",
            "name": "Name",
            "type": "singleLineText"
        }
```

(continues on next page)

(continued from previous page)

```
    ],
    "views": [
        {
            "id": "viwQpsuEDqHFqegkp",
            "name": "Grid view",
            "type": "grid"
        }
    ]
}
```

Return type Optional[dict]

CHAPTER

FIVE

0.X MIGRATION

Airtable Python Wrapper was renamed to **pyAirtable** starting on its first major release, `1.0.0`. The docs for the older release will remain on Read the Docs, the source code on this branch. The last `0.x` release will remain available on PYPI.

You can read about the reasons behind the renaming here.

5.1 API Changes

When writing pyAirtable, we a few changes to the api:

- Introduced a simpler api that's more closely aligned with Airtable Api's patterns.
- Created more a flexible API (*`Api`*, *`Base`*, *`Table`*)

Table 1: Changes

0.x (airtable-python-wrapper)	1.0 (pyAirtable)
`Airtable()`	*`Api`*, *`Base`*, *`Table`*
`get()`	`get()`
`get_iter()`	`iterate()`
`get_all()`	`all()`
`search()`	`all(formula=match({"Name" : "X"})`
`match(**kwargs)`	`first(formula=match({"Name" : "X"})`
`insert()`	`create()`
`update()`	`update()`
`replace()`	use `update(replace=True)`
`delete()`	`delete()`

CHAPTER

SIX

ABOUT

6.1 Questions

Post them over in the project's Github Page

6.2 Contribute

```
git clone git@github.com:gtalarico/pyairtable.git
cd pyairtable
pip install -e .
```

6.3 License

MIT License

CHAPTER

SEVEN

CHANGELOG

Warning: Looking for airtable-python-wrapper changelog? See *0.x Migration*.

7.1 1.0.0

Release Date: 2021-08-07

- pyAirtable rewrite for 1.0. See *0.x Migration*.

7.2 0.15.3

Release Date: 2021-07-26

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

D

E

F

G

I

L

M

P

S

T

U