CONTENTS

1 Example 3
2 API 5
WebStore Client is a simple Python wrapper to easily access WebStore, a web-based table store used for on-line data storage, processing and visualization. WebStore supports various ways to access the data stored in it, but this Python client library makes using it as simple as a generic `csv.DictWriter`. 
To use WebStore, you need to have an instance of the WebStore server running either locally or on the web. If you also want to have write access, you’ll need valid access credentials. As WebStore doesn’t handle authentication internally, this usually either means signing up with an associated instance of CKAN or adding a user to an Apache `.htaccess` file.

Once you have both a server and credentials, you can start using WebClient by creating a `Database` object:

```python
>>> from webstore.client import Database
>>> database = Database('webstore.myserver.org', 'owner', 'mydatabase')
```

Note that each database has a user that owns it and that needs to be specified when connecting to a database. If you were to sign into your own database, the could would look like this:

```python
>>> database = Database('webstore.myserver.org', 'me', 'testdb',
                       http_user='me', http_password='secret')
```

Or, if you are using API key rather than user and password:

```python
>>> database = Database('webstore.myserver.org', 'me', 'testdb',
                       http_apikey='my-api-key')
```

There is no special command to create a database, so just connecting to an arbitrary name within your own namespace will create one. Once you have connected to a database, you can list tables or check for a specific name:

```python
>>> database.tables()
["testdb", "postal_codes", "movies"]
```

```python
>>> 'triples' in database
False
```

To actually begin using a table, you can select a table:

```python
>>> table = database['weather']
```

... but what good is an empty table? So let’s fill this thing with some rows:

```python
>>> table.writerow({'place': 'Berlin', 'temp': 23})
>>> table.writerows([{'place': 'London', 'temp': 5},
                   {'place': 'Moscow', 'temp': -2}])
```

As you run this, both the table and the required columns are created automatically. This means you don’t need to worry about schema creation at all. You cannot, however, store complex objects like `dict`, `list`, `tuple` or custom classes to WebStore.

While it’s simple to add new data, for updating existing rows, we use a little trick: `unique_columns`. This set of column names will be used to try and perform an update:
>>> table.writerow({'place': 'Berlin', 'temp': 18},
                  unique_columns=['place'])

This will update the temp values of all rows mentioning Berlin, but leave any other columns intact.

Now that we have added some data to the table, we can try and traverse it:

```python
>>> for row in table:
    print row['place']
Berlin
London
Moscow
```

Using traverse() instead will give you the option to apply limits, offsets and very simple column filters:

```python
>>> for row in table.traverse(place='Berlin', _limit=4, _offset=0):
    print row['temp']
18
```

For more informations on how you can use the WebStore client, have a look at the API documentation for Table.
Access to the WebStore client happens via two simple classes: Database and Table.