deux Documentation

Release 1.2.0

Robinhood Markets

Sep 27, 2017

Contents

D E U X

Multifactor Authentication for Django Rest Framework

CHAPTER 1

Copyright

DEUX

Multifactor Authentication for Django Rest Framework

CHAPTER 2

Introduction

Version 1.2.0

Web https://deux.readthedocs.org/

Download https://pypi.python.org/pypi/deux

Source https://github.com/robinhood/deux

Keywords authentication, two-factor, multifactor

About

Multifactor Authentication provides multifactor authentication integration for the Django Rest Framework. It integrates with Token Authentication built into DRF and OAuth2 provided by django-oauth-toolkit.

What is Multifactor Authentication?

Multifactor Authentication (MFA) is a security system that requires more than one method of authentication from independent categories of credentials to verify the user's identity for a login or other transaction. (Source: SearchSecurity)

Quick Start

Go immediately to the *DRF Integration* guide to get started using deux in your Django Rest Framework projects.

Installation

You can install deux either via the Python Package Index (PyPI) or from source.

Requirements

`deux` version 1.2.0 runs on Python (2.7, 3.4, 3.5).

Installing with pip

To install using *pip*:

```
$ pip install -U deux
```

Downloading and installing from source

Download the latest version of deux from http://pypi.python.org/pypi/deux

You can install it by doing the following:

```
$ tar xvfz deux-0.0.0.tar.gz
$ cd deux-0.0.0
$ python setup.py build
# python setup.py install
```

The last command must be executed as a privileged user if you are not currently using a virtualenv.

Using the development version

With pip

You can install it by doing the following:

```
$ pip install https://github.com/robinhood/deux/zipball/master#egg=deux
```

Bug tracker

If you have any suggestions, bug reports or annoyances please report them to our issue tracker at https://github.com/robinhood/deux/issues/

Contributing

Development of *Deux* happens at GitHub: https://github.com/robinhood/deux

You are highly encouraged to participate in the development of *deux*. If you don't like GitHub (for some reason) you're welcome to send regular patches.

Be sure to also read the Contributing to Deux section in the documentation.

License

This software is licensed under the *New BSD License*. See the `LICENSE` file in the top distribution directory for the full license text.

CHAPTER 3

User Guide

Release 1.2

Guide for installing, configuring, and extending `deux` inside your application.

DRF Integration

Setup

To set up `deux` for your Django Rest Framework application, follow these steps. For help setting up a DRF project, see guide here.

1. Install deux.

```
$ pip install deux
```

2. Add `deux` to INSTALLED_APPS after `rest_framework.authtoken` and `oauth2_provider`, depending on which authentication protocol you use.

```
INSTALLED_APPS = (
    # ...,
    'rest_framework.authtoken',
    'oauth2_provider',
    # ...,
    'deux',
)
```

3. Migrate your database to add the `MultiFactorAuth` model.

```
$ python manage.py migrate
```

4. Configure your `settings.py` file, as described in *Settings*.

Views

The library comes with a standard set of views you can add to your Django Rest Framework API, that allows your users to enable/disable multifactor authentication.

To enable them, add the following configuration to your file `urls.py`:

```
url(r"^mfa/", include("deux.urls", namespace="mfa")),
```

The library also provides views for authenticating through multifactor authentication depending on your authentication protocol.

1. For `authtoken`, add the following to `urls.py`:

   ```
   url(r"^mfa/authtoken/", include(
       "deux.authtoken.urls", namespace="mfa-authtoken:login")),
   ```

2. For `oauth2`, add the following to `urls.py`:

   ```
   url(r"^mfa/oauth2/", include(
       "deux.oauth2.urls", namespace="mfa-oauth2:login")),
   ```

Settings

The library takes the following settings object. The default values are as followed:

```
DEUX = {
    "BACKUP_CODE_DIGITS": 12,
    "MFA_CODE_NUM_DIGITS": 6,
    "STEP_SIZE": 30,
    "MFA_MODEL": "deux.models.MultiFactorAuth",
    "SEND_MFA_TEXT_FUNC": "deux.notifications.send_mfa_code_text_message",
    "TWILIO_ACCOUNT_SID": "",
    "TWILIO_AUTH_TOKEN": "",
    "TWILIO_PHONE_NUMBER": "",
}
```

MFA Optional Settings

1. `BACKUP_CODE_DIGITS`: The length of multifactor backup code.
   - **Default**: `12`
2. `MFA_CODE_NUM_DIGITS`: The length of a multifactor authentication code.
   - **Default**: `6`
3. `STEP_SIZE`: The length of an authentication window in seconds.
   - **Usage**: An authentication code is valid for 3 windows: the window in which the code is generated, the window before, and the window after.
   - **Default**: `6`
4. `MFA_MODEL`: The model used for multifactor authentication
   - **Default**: `models.MultiFactorAuth`

- **Descrtiption**: The default model is a blank extension of `abstract_models.AbstractMultiFactorAuth`

Twilio Driver Settings

1. `SEND_MFA_TEXT_FUNC`: The function used for sending text messages to users.
   - **Default**: `deux.notifications.send_mfa_code_text_message`

If you use our default Twilio driver, you must also include your Twilio credentials in the settings object.

1. `TWILIO_ACCOUNT_SID`: Your Twilio account's SID.
2. `TWILIO_AUTH_TOKEN`: Your Twilio account's authentication token.
3. `TWILIO_PHONE_NUMBER`: Your Twilio account's phone number.

Usage

Introduction

This library provides support for enabling Multifactor Authentication and then authenticating through MFA.

Currently, the library only supports multifactor authentication over SMS, but it can be easily extended to support new challenge types. The high level API is as followed.

View detailed URL documentation here.

Getting MFA Status

Users can submit a request to `GET mfa/` to get information about whether MFA is enabled and which phone number it is enabled through.

Authentication

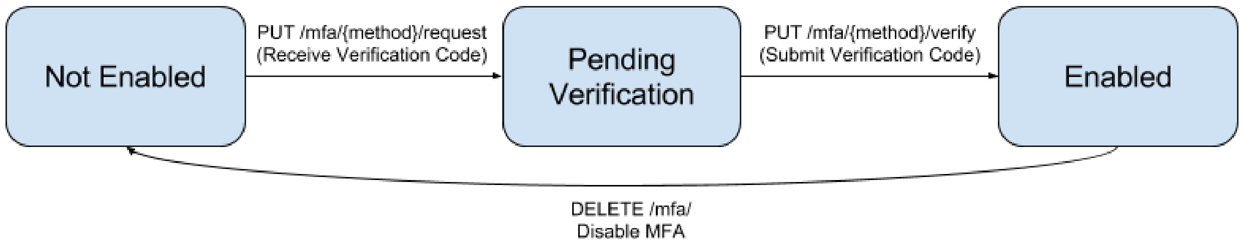
Deux supports authentication through both `authtoken` and `oauth2`. For both of these protocols, users must submit their username, password, and an MFA code or backup code. If the request is submitted without the token, they will be prompted for a token.

1. For `authtoken`, place the request to `PUT /mfa/authtoken/login`.
2. For `oauth2`, place the request to `PUT /mfa/oauth2/token` with a `password` grant type.

If MFA is not enabled, these endpoints will behave like the base authentication protocols.

Enabling MFA

The enabling process involves submitting a request to an MFA method and receiving back a code. The user must then submit the code to verify the request. If the code is correct, the MFA will then be enabled.



MFA can be enabled through the following methods:

SMS

To enable MFA through SMS, the user must first submit a request to `PUT mfa/sms/request/` with a phone number, which will send an SMS to the phone number with the MFA code.

The user should then submit a `PUT mfa/sms/verify/` request with the MFA code to enable MFA.

Disabling MFA

Users can submit a `DELETE mfa/` request to disable MFA.

Backup Code

Users only have one backup code which can be used to authenticate. If you use a backup code to authenticate, MFA will be disabled. To get the backup code, the user can submit a request to `GET mfa/recovery`.

The backup code will be reset every request.

Extending

Notifications

The send SMS function can be directly overridden by a custom function. You can configure the function in your `SEND_MFA_TEXT_FUNC` setting.

Your SMS function should throw `deux.exceptions.FailedChallengeError` for any errors to be caught by this library's functions.

Your function can look something like this:

```
def custom_send_function(mfa_instance, mfa_code):
    ...
```

To use the function, in your `settings.py`:

```
DEUX = {
    ...
    "SEND_MFA_TEXT_FUNC": "<module_path>.custom_send_function",
}
```

Models

You can write your own custom model that extends *deux.abstract_models.AbstractMultiFactorAuth* and configure the model in your MFA_MODEL setting.

Your model can look something like this:

```
class CustomMultiFactorAuth(AbstractMultiFactorAuth):
    ...
```

To use the function, in your settings.py:

```
DEUX = {
    ...
    "MFA_MODEL": "<module_path>.CustomMultiFactorAuth",
}
```

Authentication Protocols

Currently, the package supports authtoken and oauth2. You can easily extend the package to support your authentication protocol of choice.

Create a new sub-directory under the main application for your authentication and create a new login endpoint that follows the same two factor login protocol as the rest of the package.

Register your new endpoint in the test_proj/urls.py file like this:

```
url(r"^mfa/<protocol>/",
    include("deux.<protocol>.urls", namespace="<protocol>"),
),
```

Look at *deux.authtoken* or *deux.oauth2* for examples.

Challenge Methods

Currently, the package supports two factor over text message. However, it is easy to add your own challenge method for two factor (i.e. Google Authenticator or email).

Create a new challenge type in *deux.constants*.

```
YOUR_CHALLENGE_METHOD = "<Your challenge method.>"

CHALLENGE_TYPES = (SMS, YOUR_CHALLENGE_METHOD)
```

Then, add a new challenge method to the *deux.services.MultiFactorChallenge* class.

```
class MultiFactorChallenge(object):
    ...

    def generate_challenge(self):
        """
        Generates and executes the challenge object based on the challenge
        type of this object.
        """
        dispatch = {
            SMS: self._sms_challenge,
            YOUR_CHALLENGE_METHOD: self._your_challenge_method,
        }

    ...

    def _your_challenge_method(self):
        """Executes your challenge method."""
        ...
```

Then, add the necessary endpoints around requesting and verifying Two Factor with this challenge method.

```
url(r"^your_challenge_method/request/$",
    views.YourChallengeMethodRequestDetail.as_view(),
    name="your_challenge_method_request-detail"
),
url(r"^your_challenge_method/verify/$",
    views.YourChallengeMethodVerifyDetail.as_view(),
    name="your_challenge_method_verify-detail"
),
url(r"^sms/verify/$", views.SMSChallengeVerifyDetail.as_view(),
    name="sms_verify-detail"),
```

CHAPTER 4

API Reference

Release 1.2

Date Sep 27, 2017

deux

Multifactor Authentication for Django Rest Framework

GET /mfa/

Sample Response

```
200 OK
{
    "enabled": True or False,
    "challenge_type": "sms"
    "phone_number": "14085862744"
}
```

DELETE /mfa/

Expected Response

```
204 NO CONTENT
```

PUT /mfa/sms/request/

Expected Request

```
{
    "phone_number": "14085862744"
}
```

Expected Response

```
200 OK
{
    "enabled":  False,
    "challenge_type": ""
    "phone_number": "14085862744"
}
```

PUT /mfa/sms/verify/

Expected Request

```
{
    "mfa_code": "123456"
}
```

Expected Response

```
200 OK
{
    "enabled":  True,
    "challenge_type": "sms"
    "phone_number": "14085862744"
}
```

GET /mfa/recovery/

Expected Response

```
200 OK
{
    "backup_code: "123456789012"
}
```

deux.abstract_models

class `deux.abstract_models.`**`AbstractMultiFactorAuth`**

This abstract class holds user information, MFA status, and secret keys for the user.

`CHALLENGE_CHOICES` **= ((u'sms', u'SMS'), (u'', u'Off'))**

Different status options for this MFA object.

`backup_code`

Returns the users backup code.

`backup_key`

Secret key used for backup code.

`challenge_type`
Challenge type used for MFA.

`check_and_use_backup_code`(*code*)
Checks if the inputted backup code is correct and disables MFA if the code is correct.

This method should be used for authenticating with a backup code. Using a backup code to authenticate disables MFA as a side effect.

`disable`()
Disables MFA for this user.

The disabling process includes setting the objects challenge type to *DISABLED*, and removing the *backup_key* and *phone_number*.

`enable`(*challenge_type*)
Enables MFA for this user with the inputted challenge type.

The enabling process includes setting this objects challenge type and generating a new backup key.

Parameters **`challenge_type`** – Enable MFA for this type of challenge. The type must be in the supported *CHALLENGE_TYPES*.

Raises `AssertionError` – If `challenge_type` is not a supported challenge type.

`enabled`
Returns if MFA is enabled.

`get_bin_key`(*challenge_type*)
Returns the key associated with the inputted challenge type.

Parameters **`challenge_type`** – The challenge type the key is requested for. The type must be in the supported *CHALLENGE_TYPES*.

Raises `AssertionError` – If `challenge_type` is not a supported challenge type.

`phone_number`
User's phone number.

`refresh_backup_code`()
Refreshes the users backup key and returns a new backup code.

This method should be used to request new backup codes for the user.

`sms_bin_key`
Returns binary data of the SMS secret key.

`sms_secret_key`
Secret key used for SMS codes.

`user`
User this MFA object represents.

deux.authtoken

POST /mfa/authtoken/login/

Expected Request

```
{
    "username": "testuser",
    "password": "mypassword",
    "mfa_code": "123456", (Optional)
    "backup_code": "123456789012" (Optional)
}
```

Expected Response if Authenticated

```
200 OK
{
    "token": "<token>",
}
```

Expected Response if MFA Required

```
200 OK
{
    "mfa_required": True,
    "mfa_type": "sms"
}
```

deux.authtoken.serializers

class `deux.authtoken.serializers.`**`MFAAuthTokenSerializer`**

This extends the `AuthTokenSerializer` to support multifactor authentication.

`backup_code` **= None**

Serializer field for Backup code.

`mfa_code` **= None**

Serializer field for MFA code field.

`validate`(*attrs*)

Extends the AuthTokenSerializer validate method to implement multi factor authentication.

If MFA is disabled, authentication requires just a username and password.

If MFA is enabled, authentication requires a username, password, and either a MFA code or a backup code. If the request only provides the username and password, the server will generate an appropriate challenge and respond with *mfa_required = True*.

Upon using a backup code to authenticate, MFA will be disabled.

Parameters **`attrs`** – Dictionary of data inputted by the user.

Raises **`serializers.ValidationError`** – If invalid MFA code or backup code are submitted. Also if both types of code are submitted simultaneously.

deux.authtoken.views

class `deux.authtoken.views.`**`ObtainMFAAuthToken`**

View for authenticating which extends the `ObtainAuthToken` from Django Rest Framework's Token Authentication.

post (*self*, *request*)
Override ObtainAuthToken's post method for multifactor authentication.

(1) When MFA is required, send the user a response indicating which challenge is required. (2) When authentication is successful return the auth token.

Parameters **request** – Request object from the client.

serializer_class
alias of `MFAAuthTokenSerializer`

deux.constants

`deux.constants.`**CHALLENGE_TYPES = (u'sms',)**
A tuple of all support challenge types.

`deux.constants.`**DISABLED = u''**
Represents the `DISABLED` state of MFA.

`deux.constants.`**SMS = u'sms'**
Represents the state of using `SMS` for MFA.

deux.exceptions

exception `deux.exceptions.`**FailedChallengeError**
Generic exception for a failed challenge execution.

exception `deux.exceptions.`**InvalidPhoneNumberError** (*message=u'Please enter a valid phone number.'*)
Exception for SMS that fails because phone number is not a valid number for receiving SMS's.

exception `deux.exceptions.`**TwilioMessageError** (*message=u'SMS failed to send.'*)
Exception that Twilio failed to send the text message for reasons other than `NotSMSNumberError`.

deux.models

class `deux.models.`**MultiFactorAuth**
Blank extension of `AbstractMultiFactorAuth` that is used as the default model in this package.

deux.notifications

deux.oauth2

POST /mfa/oauth2/token/

Expected Request

```
{
    "grant_type": "password"
    "username": "testuser",
    "password": "mypassword",
    "mfa_code": "123456", (Optional)
    "backup_code": "123456789012" (Optional)
}
```

Expected Response if Authenticated

```
200 OK
{
    "access_token": "<token>",
    "expires_in": "<seconds>",
    "token_type": "Bearer",
    "scope": "<scope>",
    "refresh_token": "<token>"
}
```

Expected Response if MFA Required

```
200 OK
{
    "mfa_required": True,
    "mfa_type": "sms"
}
```

deux.oauth2.backends

class `deux.oauth2.backends.`**`MFARequestBackend`**

OAuth2 backend class for MFA extending `JSONOAuthLibCore`. It extracts extra credentials (`mfa_code` and `backup_code`) from the request body.

`_get_extra_credentials`(*body*)

Gets dictionary of `mfa_code` and `backup_code` from the body.

Parameters **`body`** – The request body in url encoded form.

Returns Dictionary with `mfa_code` and `backup_code`.

`create_token_response`(*request*)

Overrides the base method to pass in the request body instead of the request because Django only allows the request data stream to be read once.

Parameters **`request`** – The request to create a token response from.

Returns The redirect uri, headers, body, and status of the response.

`extract_body`(*request*)

Extract request body by coercing the request to a Django Rest Framework Request.

Params request The request to extract the body from.

Returns Returns the items in the requests body.

deux.oauth2.exceptions

exception `deux.oauth2.exceptions.`**`ChallengeRequiredMessage`**`(`*challenge_type*`)`

This exception is used to prompt the user for an MFA code.

This exception is used when a user passes in their username and password, and they have MFA enabled.

`status_code` = **200**

This exception returns a 200 response.

`twotuples`

Returns a list of tuples that will be converted to the error response. This method override the `two_tuples` method from `OAuth2Error`.

exception `deux.oauth2.exceptions.`**`InvalidLoginError`**`(`*message*`)`

Generic exception for a failed login attempt through OAuth2. This exception will result in a 400 Bad Request error in the OAuth API.

`twotuples`

Returns a list of tuples that will be converted to the error response. This method override the `two_tuples` method from `OAuth2Error`.

deux.oauth2.validators

class `deux.oauth2.validators.`**`MFAOAuth2Validator`**

OAuth2 validator class for MFA that validates requests to authenticate with username and password by also verifying that they supply the correct MFA code or backup code if multifactor authentication is enabled.

`validate_user``(`*username*, *password*, *client*, *request*, **args*, ***kwargs*`)`

Overrides the OAuth2Validator validate method to implement multi factor authentication.

If MFA is disabled, authentication requires just a username and password.

If MFA is enabled, authentication requires a username, password, and either a MFA code or a backup code. If the request only provides the username and password, the server will generate an appropriate challenge and respond with *mfa_required = True*.

Upon using a backup code to authenticate, MFA will be disabled.

Parameters **`attrs`** – Dictionary of data inputted by the user.

Raises

- ***`deux.oauth2.exceptions.InvalidLoginError`*** – If invalid MFA code or backup code are submitted. Also if both types of code are submitted simultaneously.
- ***`deux.oauth2.exceptions.ChallengeRequiredMessage`*** – If the user has MFA enabled but only supplies the correct username and password. This exception will prompt the OAuth2 system to send a response asking the user to supply an MFA code.

deux.oauth2.views

class `deux.oauth2.views.`**`MFATokenView`**

Extends OAuth's base TokenView to support MFA.

oauthlib_backend_class
Use Deux's custom backend for the MFA OAuth api.

alias of `MFARequestBackend`

validator_class
Use Deux's custom validator for the MFA OAuth api.

alias of `MFAOAuth2Validator`

deux.serializers

class `deux.serializers.`**BackupCodeSerializer**
Serializer for the user requesting backup codes.

backup_code = None
Serializer field for the backup code.

get_backup_code(*instance*)
Method for retrieving the backup code. On every request, the backup code is refreshed.

Parameters **instance** – `MultiFactorAuth` instance to use.

Raises **serializers.ValidationError** – If MFA is disabled.

class `deux.serializers.`**MultiFactorAuthSerializer**
Basic MultiFactorAuthSerializer that encodes MFA objects into a standard response.

The standard response returns whether MFA is enabled, the challenge type, and the user's phone number.

to_representation(*mfa_instance*)
Encodes an MFA instance as the standard response.

Parameters **mfa_instance** – `MultiFactorAuth` instance to use.

Returns Dictionary with `enabled`, `challengetype`, and `phone_number` from the MFA instance.

class `deux.serializers.`**SMSChallengeRequestSerializer**
Serializer that facilitates a request to enable MFA over SMS.

challenge_type = u'sms'
This serializer represents the `SMS` challenge type.

update(*mfa_instance*, *validated_data*)
If the request data is valid, the serializer executes the challenge by calling the super method and also saves the phone number the user requested the SMS to.

Parameters

- **mfa_instance** – `MultiFactorAuth` instance to use.
- **validated_data** – Data returned by `validate`.

class `deux.serializers.`**SMSChallengeVerifySerializer**
Extension of `_BaseChallengeVerifySerializer` that implements challenge verification for the SMS challenge.

challenge_type = u'sms'
This serializer represents the `SMS` challenge type.

class `deux.serializers.`**_BaseChallengeRequestSerializer**
Base Serializer class for all challenge request.

challenge_type
Represents the challenge type this serializer represents.

Raises NotImplemented – If the extending class does not define `challenge_type`.

execute_challenge(*instance*)
Execute challenge for this instance based on the `challenge_type`.

Parameters instance – `MultiFactorAuth` instance to use.

Raises serializers.ValidationError – If the challenge fails to execute.

update(*mfa_instance*, *validated_data*)
If the request is valid, the serializer calls update which executes the `challenge_type`.

Parameters

- **mfa_instance** – `MultiFactorAuth` instance to use.
- **validated_data** – Data returned by `validate`.

validate(*internal_data*)
Validate the request to enable MFA through this challenge.

Extending classes should extend for additional functionality. The base functionality ensures that MFA is not already enabled.

Parameters internal_data – Dictionary of the request data.

Raises serializers.ValidationError – If MFA is already enabled.

class `deux.serializers.`**_BaseChallengeVerifySerializer**
This serializer first extracts MFA code from request body (*to_internal_value*). It then verifies the code against the corresponding *MultiFactorAuth* instance (*validate*). If the code is valid, it enables MFA based on the challenge type (*update*).

challenge_type
Represents the challenge type this serializer represents.

Raises NotImplemented – If the extending class does not define `challenge_type`.

mfa_code = None
Requests to verify an MFA code must include an `mfa_code`.

update(*mfa_instance*, *validated_data*)
If the request is valid, the serializer enables MFA on this instance for this serializer's `challenge_type`.

Parameters

- **mfa_instance** – `MultiFactorAuth` instance to use.
- **validated_data** – Data returned by `validate`.

validate(*internal_data*)
Validates the request to verify the MFA code. It first ensures that MFA is not already enabled and then verifies that the MFA code is the correct code.

Parameters internal_data – Dictionary of the request data.

Raises serializers.ValidationError – If MFA is already enabled or if the inputted MFA code is not valid.

deux.services

class `deux.services.`**`MultiFactorChallenge`**(*instance*, *challenge_type*)

A class that represents a supported challenge and has the ability to execute the challenge.

Parameters

- **`instance`** – `MultiFactorAuth` instance to use.
- **`challenge_type`** – Challenge type being used for this object.

Raises **`AssertionError`** – If `challenge_type` is not a supported challenge type.

`generate_challenge`()

Generates and executes the challenge object based on the challenge type of this object.

`deux.services.`**`generate_key`**()

Generates a key used for secret keys.

`deux.services.`**`generate_mfa_code`**(*bin_key*, *drift=0*)

Generates an MFA code based on the `bin_key` for the current timestamp offset by the `drift`.

Parameters

- **`bin_key`** – The secret key to be converted into an MFA code
- **`drift`** – Number of time steps to shift the conversion.

`deux.services.`**`verify_mfa_code`**(*bin_key*, *mfa_code*)

Verifies that the inputted `mfa_code` is a valid code for the given secret key. We check the `mfa_code` against the current time stamp as well as one time step before and after.

Parameters

- **`bin_key`** – The secret key to verify the MFA code again.
- **`mfa_code`** – The code whose validity this function tests.

deux.strings

`deux.strings.`**`BOTH_CODES_ERROR`** **= u'Login does not take both a verification and backup code.'**

Error if user submits both MFA and backup code for authentication.

`deux.strings.`**`DISABLED_ERROR`** **= u'Two factor authentication is not enabled.'**

Error if MFA is unexpectedly in a disabled state.

`deux.strings.`**`ENABLED_ERROR`** **= u'Two factor authentication is already enabled.'**

Error if MFA is unexpectedly in an enabled state.

`deux.strings.`**`INVALID_BACKUP_CODE_ERROR`** **= u'Please enter a valid backup code.'**

Error if an invalid backup code is entered.

`deux.strings.`**`INVALID_CREDENTIALS_ERROR`** **= u'Unable to log in with provided credentials.'**

Error if a user provides an invalid username/password combination.

`deux.strings.`**`INVALID_MFA_CODE_ERROR`** **= u'Please enter a valid code.'**

Error if an invalid MFA code is entered.

`deux.strings.`**`INVALID_PHONE_NUMBER_ERROR`** **= u'Please enter a valid phone number.'**

Error if an invalid phone number is entered.

`deux.strings.`**`MFA_CODE_TEXT_MESSAGE`** **= u'Two Factor Authentication Code: {code}'**
Message body for a MFA code.

`deux.strings.`**`PHONE_NUMBER_NOT_SET_ERROR`** **= u'MFA phone number must be set for this challenge.'**
Error if phone number is not set for a challenge that requires it.

`deux.strings.`**`SMS_SEND_ERROR`** **= u'SMS failed to send.'**
Error if SMS fails to send.

deux.validators

`deux.validators.`**`phone_number_validator`** **= <django.core.validators.RegexValidator object>**
Regex validator for phone numbers.

deux.views

class `deux.views.`**`BackupCodeDetail`**
View for retrieving the user's backup code.

`serializer_class`
alias of `BackupCodeSerializer`

class `deux.views.`**`MultiFactorAuthDetail`**
View for requesting data about MultiFactorAuth and disabling MFA.

`perform_destroy`(*instance*)
The delete method should disable MFA for this user.

Raises **`rest_framework.exceptions.ValidationError`** – If MFA is not enabled.

`serializer_class`
alias of `MultiFactorAuthSerializer`

class `deux.views.`**`MultiFactorAuthMixin`**
Mixin that defines queries for MFA objects.

`get_object`()
Gets the current user's MFA instance

class `deux.views.`**`SMSChallengeRequestDetail`**
View for requesting SMS challenges to enable MFA through SMS.

`serializer_class`
alias of `SMSChallengeRequestSerializer`

class `deux.views.`**`SMSChallengeVerifyDetail`**
View for verify SMS challenges to enable MFA through SMS.

`serializer_class`
alias of `SMSChallengeVerifySerializer`

class `deux.views.`**`_BaseChallengeView`**
Base view for different challenges.

`challenge_type`
Represents the challenge type this serializer represents.

Raises **`NotImplemented`** – If the extending class does not define `challenge_type`.

CHAPTER 5

Contributing

Welcome!

This document is fairly extensive and you are not really expected to study this in detail for small contributions;

> The most important rule is that contributing must be easy and that the community is friendly and not nitpicking on details such as coding style.

If you're reporting a bug you should read the Reporting bugs section below to ensure that your bug report contains enough information to successfully diagnose the issue, and if you're contributing code you should try to mimic the conventions you see surrounding the code you are working on, but in the end all patches will be cleaned up by the person merging the changes so don't worry too much.

Community Code of Conduct

The goal is to maintain a diverse community that is pleasant for everyone. That is why we would greatly appreciate it if everyone contributing to and interacting with the community also followed this Code of Conduct.

The Code of Conduct covers our behavior as members of the community, in any forum, mailing list, wiki, website, Internet relay chat (IRC), public meeting or private correspondence.

The Code of Conduct is heavily based on the Ubuntu Code of Conduct, and the Pylons Code of Conduct.

Be considerate.

Your work will be used by other people, and you in turn will depend on the work of others. Any decision you take will affect users and colleagues, and we expect you to take those consequences into account when making decisions. Even if it's not obvious at the time, our contributions to Deux will impact the work of others. For example, changes to code, infrastructure, policy, documentation and translations during a release may negatively impact others work.

Be respectful.

The Deux community and its members treat one another with respect. Everyone can make a valuable contribution to Deux. We may not always agree, but disagreement is no excuse for poor behavior and poor manners. We might all experience some frustration now and then, but we cannot allow that frustration to turn into a personal attack. It's important to remember that a community where people feel uncomfortable or threatened is not a productive one. We expect members of the Deux community to be respectful when dealing with other contributors as well as with people outside the Deux project and with users of Deux.

Be collaborative.

Collaboration is central to Deux and to the larger free software community. We should always be open to collaboration. Your work should be done transparently and patches from Deux should be given back to the community when they are made, not just when the distribution releases. If you wish to work on new code for existing upstream projects, at least keep those projects informed of your ideas and progress. It many not be possible to get consensus from upstream, or even from your colleagues about the correct implementation for an idea, so don't feel obliged to have that agreement before you begin, but at least keep the outside world informed of your work, and publish your work in a way that allows outsiders to test, discuss and contribute to your efforts.

When you disagree, consult others.

Disagreements, both political and technical, happen all the time and the Deux community is no exception. It is important that we resolve disagreements and differing views constructively and with the help of the community and community process. If you really want to go a different way, then we encourage you to make a derivative distribution or alternate set of packages that still build on the work we've done to utilize as common of a core as possible.

When you are unsure, ask for help.

Nobody knows everything, and nobody is expected to be perfect. Asking questions avoids many problems down the road, and so questions are encouraged. Those who are asked questions should be responsive and helpful. However, when asking a question, care must be taken to do so in an appropriate forum.

Step down considerately.

Developers on every project come and go and Deux is no different. When you leave or disengage from the project, in whole or in part, we ask that you do so in a way that minimizes disruption to the project. This means you should tell people you are leaving and take the proper steps to ensure that others can pick up where you leave off.

Reporting Bugs

Security

You must never report security related issues, vulnerabilities or bugs including sensitive information to the bug tracker, or elsewhere in public. Instead sensitive bugs must be sent by email to `security@robinhood.com`.

If you'd like to submit the information encrypted our PGP key is:

```
-----BEGIN PGP PUBLIC KEY BLOCK-----
Version: SKS 1.1.5

mQINBFfPKmcBEADYx/ZGUwc6/x3CtViIRXz1ZyOHxERAcE2Lenmkr6oop3bt36smIgFSsU7K
VMl32j+OlKaoLlVGRevxj6kKsFdNyqYGTUM2CTWx1gmd39QBPOqQeWDmTUa6ze332bJ1yJG1
dtd/m2PuUZLAYLvUOLJSMmZgSeB22DKvNjnCZnNIw7nuGW/OIZHNYYZztNAxjIVCpXYvzPUh
2yRBN+8ZxHaQUzrwXvU8h924mS06F0q2FRz++ClMKUh42UIXUFlIkXv5iIvTM6G4TVM5wt5p
G+gCnRzbPUmStoU/RYbLj8GkFMs52rb3gAFHy+Yx/K3awVTV985eo7PuJM+TzMqdD4zPeE7Z
V626fO+cVVCSmF+3ikO65RZJ8eWYeTWlQ3dQr+kLxQcK9ZUBFCjNqab4m9OchjamvtyvKt//
V4H6datfIN/4Ss5qcpegQ3SwOokz/vWPU4qZSKAp2cQY2WU4fSkKQK2Q9m5zKhyH6E/GH9nR
x4MsBIFgRAAts8FeP3d/Xf49qd8oLje8UkNChHrLUbzaSdRNZQu3KM7K/OVI13OzSRov+mP8
Twhk2xXFRy6iibR1n4YsSWmtHv7iiin3rWk9uJXO9P7V9P8xghudfja3SstxnK1ueASTbC2f
iJgN4H0mPXNn0BC1I2na2xczP/83sOv0nHCk9PjeuSYsjhk+8wARAQABtDVSb2Jpbmhvb2Qg
TWFya2V0cyAoU2VjdXJpdHkpIDxzZWN1cml0eUByb2Jpbmhvb2QuY29tPokCOAQTAQIAIgUC
V88qZwIbAwYLCQgHAwIGFQgCCQoLBBYCAwECHgECF4AACgkQFKwy0jX7jrxJFA//TKzjxO84
yodjwAO4IIO/nUeqvwWKiSr2dcPtAFQGUno5NjxM0iM170ff8qg5WoLQsic786PM71Q0I0aF
OGFiiNRxRdS/sm0e1XYyIqu/24hwyHybpmxM+LYAoZNpUi6hAy5a+iTrorCJnGpFOUlYPDpM
rMjOhRNeo5YOLW1WXQ0mAH93lwIHCm8XkkZWiFtrg/3zLyHLz0KV7nwpY4/fm0qjp2C/B/kw
lF/Ol3opHrX8WNDYnr9IillRurqjh0Hvm8U7aNlDx9nFwb4uMYcXano37EMyVOnnCBVYT9kM
BiGBxnucTPsgs/KZLCRqihSt2qkSK3EB344oFZ5bqum8jKn/cGCLYv2GzG217FNTdNTIlAMN
zkgPlUCK885YpJDNaqScuOXphgpJr+4a71ml6GhM2G+Grkfo+YVR/d8X3Z7MJSRXxWHf5P5U
PK1QS7pbQdTG5TrEd4NNI6a4ixBWk0OJIsBcer2dFDTBQXIMfcUZ+Nb1C1vxdrvBPVkUtCIf
XbXeW4cYjxO7/AoarvPANqFol6mhZeBSHw/AiADaXs8oCIYVHPoaa5sJALhZD65vUvYZxYom
QJE+8EuV5X5EhDSWoqvnC+ugVum9wSBjI2OF4PlEfifhfo+z5Xhpus6GdniEQ9jNBr4+Lvoc
rssIUSxQQ4fsNqAgrmTauIOOWaa5Ag0EV88qZwEQALUX5gUbAmK6CRxM/15+eRuKq0IAP6+5
sJsH0IrRr7mHUi8QxYzHouWK9klVdjRvd1crr9Q48wsty13togbiDTFPRCa/Z6K0vKdAneeS
RQL89FGpQBq7nMM9GytUoBQ6BWAxItxdRiRKQ4NeyzCTcJjq1zN3KRd1d+RwnFLr3HTWbevv
yOktdbklV6ld7IT8mMsuiZw3AA74tIWD0res6FtIqUVS2I2CEIODOlIXEjRDdcTES0bXxH/2
/T3wPIfMEb1aSyhBYsGHRB7pAAqGrpb7LguVTt2hpfRShtew5O9hwLquA+kaGU/MIjKIKrxH
PVkng8TwqhS3Et/hhAdLXtWj1ZXbRV5RPa1T90+JVX2PU4IapvHjZG4iZ4Oe7wtwtRSU1mQK
Q30BpArsv7+1ezZMALsenYxbAh1ckp8bDEiNTboDzn7rBGXY2sUvLrl05oUbA7ntX0w6PIP4
SHtWshCtu5+4g2/QX4zv4OEfFY6CeLHuuaw2zSUCXEAkVJCdjAXjmLpH5LftHDGn91kqmfgl
VSQWeIfTCEue7Ehvfke1k5ASKi/L3+HPinRtT8JhCFGFM2gViNXtFMk5Dqb7TFo3g6s/Kd0/
gCpnE0844ts5Wh6S1DtbZ2YawS8lxEh0yQ1VJ4FraVEiMQ3SHFtKsAsGuR1Sz2/QL+tcRyXj
xv17ABEBAAGJAh8EGAECAAkFAlfPKmcCGwwACgkQFKwy0jX7jrwOPw//XeJ64XoWVY9NAxLP
PwXhKdZGfB8WxIs0pyF2KOAqbXisbp9Cu9OYgm42/idzobyHq1ebkQrW/hKs0248oX+dz83J
TbkllHf+5SBPJCYm5jBnWRz+knaLZwFGkjtdy7NIkArfK9u5ytzKAhWqsi06B90e3MWSWo+X
aLIGIiKZBDGbj2OCDDQyY1Sxh2r6i7Wx4ViI64GoZ0Te8gGM7r2swXYn95vSKISRDaffrczD
83qwdenp84pPFarSMtCTaNzmwwc7MzUXAEnehlfcxs6aPp3I+H4G9JrWB6jUs8pGqqe2qyvo
85K2ffTLUsmoA1+Z7tPqK8nmFe9TPUuAQiZRJuV8X0Ur4l01QwBdmFKqp9yvARoIqIEVbIxM
xofwaRkDLeewWcVa5/tVTdeovI0zAyIfiFgNes1Zi3JK1Z13cGhjHZun2EWY3dufEdzkmGxY
1D09/QyHyLi2NcDavMEblJjg95NWVwQMkTzkAngd/1bJXXzwC82wtrmTYPnDHOaLqO9WbV6L
OuCHg+ZKaLuG3fRYO+n6dYXqdoAnnYrgxhxLPFWW8knso+mz5HEc+N1ND27xzBCimQQEEjlA
YgQslkRvzsczaG7feItsnz1vWAUQvwtr22iJtaYxG1+QhKDINkJkJ9LluK7nMC3SYvZBkh4n
HBu2dHUJXU7b845lvTo=
=JVgV
-----END PGP PUBLIC KEY BLOCK-----
```

Other bugs

The best way to report an issue and to ensure a timely response is to use the issue tracker.

1. **Create a GitHub account.**

You need to create a GitHub account to be able to create new issues and participate in the discussion.

2. **Determine if your bug is really a bug.**

You should not file a bug if you are requesting support.

3. **Make sure your bug hasn't already been reported.**

Search through the appropriate Issue tracker. If a bug like yours was found, check if you have new information that could be reported to help the developers fix the bug.

4. **Check if you're using the latest version.**

A bug could be fixed by some other improvements and fixes - it might not have an existing report in the bug tracker. Make sure you're using the latest release of Deux, and try the development version to see if the issue is already fixed and pending release.

5. **Collect information about the bug.**

To have the best chance of having a bug fixed, we need to be able to easily reproduce the conditions that caused it. Most of the time this information will be from a Python traceback message, though some bugs might be in design, spelling or other errors on the website/docs/code.

1. If the error is from a Python traceback, include it in the bug report.
2. We also need to know what platform you're running (Windows, macOS, Linux, etc.), the version of your Python interpreter, and the version of Deux, and related packages that you were running when the bug occurred.

6. **Submit the bug.**

By default GitHub will email you to let you know when new comments have been made on your bug. In the event you've turned this feature off, you should check back on occasion to ensure you don't miss any questions a developer trying to fix the bug might ask.

Issue Tracker

The Deux issue tracker can be found at GitHub: https://github.com/robinhood/deux

Versions

Version numbers consists of a major version, minor version and a release number, and conforms to the SemVer versioning spec: http://semver.org.

Stable releases are published at PyPI while development releases are only available in the GitHub git repository as tags. All version tags starts with "v", so version 0.8.0 is the tag v0.8.0.

Branches

Current active version branches:

- master (https://github.com/robinhood/deux/tree/master)

You can see the state of any branch by looking at the Changelog:

> https://github.com/robinhood/deux/blob/master/Changelog

If the branch is in active development the topmost version info should contain meta-data like:

```
2.4.0
======
:release-date: TBA
:status: DEVELOPMENT
:branch: master
```

The `status` field can be one of:

- `PLANNING`

  The branch is currently experimental and in the planning stage.

- `DEVELOPMENT`

  The branch is in active development, but the test suite should be passing and the product should be working and possible for users to test.

- `FROZEN`

  The branch is frozen, and no more features will be accepted. When a branch is frozen the focus is on testing the version as much as possible before it is released.

`master` branch

The master branch is where development of the next version happens.

Maintenance branches

Maintenance branches are named after the version, e.g. the maintenance branch for the 2.2.x series is named `2.2`. Previously these were named `releaseXX-maint`.

The versions we currently maintain is:

- 1.0

  This is the current series.

Archived branches

Archived branches are kept for preserving history only, and theoretically someone could provide patches for these if they depend on a series that is no longer officially supported.

An archived version is named `X.Y-archived`.

Deux does not currently have any archived branches.

Feature branches

Major new features are worked on in dedicated branches. There is no strict naming requirement for these branches.

Feature branches are removed once they have been merged into a release branch.

Tags

Tags are used exclusively for tagging releases. A release tag is named with the format `vX.Y.Z`, e.g. `v2.3.1`. Experimental releases contain an additional identifier `vX.Y.Z-id`, e.g. `v3.0.0-rc1`. Experimental tags may be removed after the official release.

Working on Features & Patches

Note: Contributing to Deux should be as simple as possible, so none of these steps should be considered mandatory.

You can even send in patches by email if that is your preferred work method. We won't like you any less, any contribution you make is always appreciated!

However following these steps may make maintainers life easier, and may mean that your changes will be accepted sooner.

Forking and setting up the repository

First you need to fork the Deux repository, a good introduction to this is in the GitHub Guide: Fork a Repo.

After you have cloned the repository you should checkout your copy to a directory on your machine:

```
$ git clone git@github.com:username/deux.git
```

When the repository is cloned enter the directory to set up easy access to upstream changes:

```
$ cd deux
$ git remote add upstream git://github.com/robinhood/deux.git
$ git fetch upstream
```

If you need to pull in new changes from upstream you should always use the `--rebase` option to `git pull`:

```
git pull --rebase upstream master
```

With this option you don't clutter the history with merging commit notes. See Rebasing merge commits in git. If you want to learn more about rebasing see the Rebase section in the GitHub guides.

If you need to work on a different branch than `master` you can fetch and checkout a remote branch like this:

```
git checkout --track -b 3.0-devel origin/3.0-devel
```

Running the unit test suite

To run the Deux test suite you need to install a few dependencies. A complete list of the dependencies needed are located in `requirements/test.txt`.

If you're working on the development version, then you need to install the development requirements first:

```
$ pip install -U -r requirements/dev.txt
```

Both the stable and the development version have testing related dependencies, so install these next:

```
$ pip install -U -r requirements/test.txt
$ pip install -U -r requirements/default.txt
```

After installing the dependencies required, you can now execute the test suite by calling:

```
$ python setup.py test
```

This will run all of the test.

Creating pull requests

When your feature/bugfix is complete you may want to submit a pull requests so that it can be reviewed by the maintainers.

Creating pull requests is easy, and also let you track the progress of your contribution. Read the Pull Requests section in the GitHub Guide to learn how this is done.

You can also attach pull requests to existing issues by following the steps outlined here: http://bit.ly/koJoso

Calculating test coverage

To calculate test coverage you must first install the coverage module.

Installing the coverage module:

```
$ pip install -U coverage
```

Code coverage in HTML:

```
$ make cov
```

The coverage output will then be located at `cover/index.html`.

Running the tests on all supported Python versions

There is a tox configuration file in the top directory of the distribution.

To run the tests for all supported Python versions simply execute:

```
$ tox
```

Use the `tox -e` option if you only want to test specific Python versions:

```
$ tox -e 2.7
```

Building the documentation

To build the documentation you need to install the dependencies listed in `requirements/docs.txt`:

```
$ pip install -U -r requirements/docs.txt
```

After these dependencies are installed you should be able to build the docs by running:

```
$ cd docs
$ rm -rf _build
$ make html
```

Make sure there are no errors or warnings in the build output. After building succeeds the documentation is available at `_build/html`.

Verifying your contribution

To use these tools you need to install a few dependencies. These dependencies can be found in `requirements/pkgutils.txt`.

Installing the dependencies:

```
$ pip install -U -r requirements/pkgutils.txt
```

pyflakes & PEP8

To ensure that your changes conform to PEP8 and to run pyflakes execute:

```
$ make flakecheck
```

To not return a negative exit code when this command fails use the `flakes` target instead:

```
$ make flakes
```

API reference

To make sure that all modules have a corresponding section in the API reference please execute:

```
$ make apicheck
$ make configcheck
```

If files are missing you can add them by copying an existing reference file.

If the module is internal it should be part of the internal reference located in `docs/internals/reference/`. If the module is public it should be located in `docs/reference/`.

For example if reference is missing for the module `deux.awesome` and this module is considered part of the public API, use the following steps:

Use an existing file as a template:

```
$ cd docs/reference/
$ cp deux.request.rst deux.awesome.rst
```

Edit the file using your favorite editor:

```
$ vim deux.awesome.rst

    # change every occurrence of ``deux.request`` to
    # ``deux.awesome``
```

Edit the index using your favorite editor:

```
$ vim index.rst

    # Add ``deux.awesome`` to the index.
```

Commit your changes:

```
# Add the file to git
$ git add deux.awesome.rst
$ git add index.rst
$ git commit deux.awesome.rst index.rst \
    -m "Adds reference for deux.awesome"
```

Coding Style

You should probably be able to pick up the coding style from surrounding code, but it is a good idea to be aware of the following conventions.

- All Python code must follow the PEP-8 guidelines.

pep8.py is an utility you can use to verify that your code is following the conventions.

- Docstrings must follow the PEP-257 conventions, and use the following style.

  Do this:

  ```
  def method(self, arg):
      """Short description.

      More details.

      """
  ```

  or:

  ```
  def method(self, arg):
      """Short description."""
  ```

  but not this:

  ```
  def method(self, arg):
      """
      Short description.
      """
  ```

- Lines should not exceed 78 columns.

  You can enforce this in **vim** by setting the `textwidth` option:

  ```
  set textwidth=78
  ```

  If adhering to this limit makes the code less readable, you have one more character to go on, which means 78 is a soft limit, and 79 is the hard limit :)

- Import order

  – Python standard library (*import xxx*)

  – Python standard library (‘from xxx import‘)

- Third-party packages.
- Other modules from the current package.

or in case of code using Django:

- Python standard library (*import xxx*)
- Python standard library (‘from xxx import‘)
- Third-party packages.
- Django packages.
- Other modules from the current package.

Within these sections the imports should be sorted by module name.

Example:

```
import threading
import time

from collections import deque
from Queue import Queue, Empty

from .datastructures import TokenBucket
from .five import zip_longest, items, range
from .utils import timeutils
```

- Wild-card imports must not be used (*from xxx import **).
- For distributions where Python 2.5 is the oldest support version additional rules apply:
  - Absolute imports must be enabled at the top of every module:

    ```
    from __future__ import absolute_import
    ```

  - If the module uses the `with` statement and must be compatible with Python 2.5 (deux is not) then it must also enable that:

    ```
    from __future__ import with_statement
    ```

  - Every future import must be on its own line, as older Python 2.5 releases did not support importing multiple features on the same future import line:

    ```
    # Good
    from __future__ import absolute_import
    from __future__ import with_statement

    # Bad
    from __future__ import absolute_import, with_statement
    ```

    (Note that this rule does not apply if the package does not include support for Python 2.5)
- Note that we use “new-style‘ relative imports when the distribution does not support Python versions below 2.5

  This requires Python 2.5 or later:

  ```
  from . import submodule
  ```

Contributing features requiring additional libraries

Some features like a new result backend may require additional libraries that the user must install.

We use setuptools *extra_requires* for this, and all new optional features that require third-party libraries must be added.

1. Add a new requirements file in *requirements/extras*

   E.g. for a Cassandra backend this would be `requirements/extras/cassandra.txt`, and the file looks like this:

   ```
   pycassa
   ```

   These are pip requirement files so you can have version specifiers and multiple packages are separated by newline. A more complex example could be:

   ```
   # pycassa 2.0 breaks Foo
   pycassa>=1.0,<2.0
   thrift
   ```

2. Modify `setup.py`

   After the requirements file is added you need to add it as an option to `setup.py` in the `extras_require` section:

   ```
   extra['extras_require'] = {
       # ...
       'cassandra': extras('cassandra.txt'),
   }
   ```

3. Document the new feature in `docs/includes/installation.txt`

   You must add your feature to the list in the Bundles section of `docs/includes/installation.txt`.

   After you've made changes to this file you need to render the distro `README` file:

   ```
   $ pip install -U requirements/pkgutils.txt
   $ make readme
   ```

That's all that needs to be done, but remember that if your feature adds additional configuration options then these needs to be documented in `docs/configuration.rst`.

Contacts

This is a list of people that can be contacted for questions regarding the official git repositories, PyPI packages Read the Docs pages.

If the issue is not an emergency then it is better to *report an issue*.

Committers

Abhishek Fatehpuria

github https://github.com/abhishek776

Jamshed Vesuna

github https://github.com/JamshedVesuna

Packages

Deux

git https://github.com/robinhood/deux

CI http://travis-ci.org/#!/robinhood/deux

Windows-CI https://ci.appveyor.com/project/robinhood/deux

PyPI http://pypi.python.org/pypi/deux

docs http://deux.readthedocs.io

Release Procedure

Updating the version number

The version number must be updated two places:

- `deux/__init__.py`
- `docs/include/introduction.txt`

After you have changed these files you must render the `README` files. There is a script to convert Sphinx syntax to generic reStructured Text syntax, and the make target *readme* does this for you:

```
$ make readme
```

Now commit the changes:

```
$ git commit -a -m "Bumps version to X.Y.Z"
```

and make a new version tag:

```
$ git tag vX.Y.Z
$ git push --tags
```

Releasing

Commands to make a new public stable release:

```
$ make distcheck  # checks pep8, autodoc index, runs tests and more
$ make dist  # NOTE: Runs git clean -xdf and removes files not in the repo.
$ python setup.py sdist upload --sign --identity='Ask Solem'
$ python setup.py bdist_wheel upload --sign --identity='Ask Solem'
```

If this is a new release series then you also need to do the following:

- **Go to the Read The Docs management interface at:** http://readthedocs.org/projects/deux

- Enter "Edit project"

  Change default branch to the branch of this series, e.g. `2.4` for series 2.4.

- Also add the previous version under the "versions" tab.

CHAPTER 6

Changelog

1.2.0

release-date 2016-10-28 06:00 P.M PDT

release-by Abhishek Fatehpuria

- [oauth2] Support url-encoded requests for oauth

1.1.1

release-date 2016-09-13 06:00 P.M PDT

release-by Abhishek Fatehpuria

- [bugfix] Remove duplicate strings in Deux strings.py

1.1.0

release-date 2016-09-13 06:00 P.M PDT

release-by Abhishek Fatehpuria

- Added pytest
- Added codecov
- Changed settings name from "Deux" to "DEUX"

1.0.0

release-date 2016-09-07 03:00 P.M PDT

release-by Abhishek Fatehpuria

- Initial release

CHAPTER 7

Indices and tables

- modindex
- search

Python Module Index

Index