Automatic Carbon Offsetting

Noel Strahm

Sep 29, 2021

CONTENTS


Automatic Carbon Offsetting

CHAPTER

ONE

ABOUT

Researchers in the environmental social sciences, broadly construed, are increasingly studying behavior in paradigms with actual environmental consequences. While studying people's willingness to invest in environmental public goods, many experimentalists rely on emission certificate markets to purchase and retire certificates to limit the caps in cap-and-trade regulated markets. Thereby, experimental decisions trigger actual environmental consequences.

Our software-module helps researchers using oTree to automate the process of purchasing and retiring emission certificates in experiments. When an experiment is finished, researchers receive an email and are directed to a customized donation form of Compensators.org, a platform facilitating the purchase and retirement of certificates. The tool helps researchers to purchase emission certificates with minimal effort, thereby reducing the entry cost for conducting experiments with environmental consequences.

This website provides simple step by step instructions that show researchers how to integrate the software module in their *oTree* projects.

CHAPTER

TWO

SUPPORT

For help, please contact noel.strahm@iop.unibe.ch

CHAPTER

THREE

CONTENTS:

3.1 Step 1: Creating SMTP account

The Simple Mail Transfer Protocol (SMTP) is an internet standard communication protocol that is used to send email over the internet. The software module requires a valid SMTP account, so that an email with all necessary information including the link to purchase the carbon-emission certificate can be sent after the experiment has finished. There is a multitude of SMTP service providers to choose from. Various service providers offer a free plan which is more than enough for the purposes of the software module.

Possible service providers with a free plan are Mailjet or SendinBlue. We used Mailjet because no credit card is needed, the free plan has no expiring date and it's possible to send up 200 emails a day.

3.2 Step 2: Importing Modules

In order to integrate the software module in your *oTree* project the following three modules have to be imported at the top of the models.py file of your oTree app.

- `smtblib`: Used to send the automated email.
- `requests`: Used to obtain the current CO2 price.
- `traceback`: Used for error handling purposes.

The top of your models.py file should look like this:

```
from otree.api import (
    models,
    widgets,
    BaseConstants,
    BaseSubsession,
    BaseGroup,
    BasePlayer,
    Currency as c,
    currency_range,
)
import smtplib
import requests
import traceback
#import everything else you need here
```

Note: Source Code: All source code described on this website can simply be copied into your *oTree* project from the code block at the bottom of this section.

3.3 Step 3: Embedding the function

Next, the `send_payment_mail()` function has to be embedded in the Subsession class of the models.py file. The following section explains how the function works, how it can be modified and how it has to be integrated in the Subsession class of your *oTree* app.

3.3.1 Initial set up

First, enter your SMTP account credentials in the *#CONSTANTS* section of the function.

- **MAIL_USER & MAIL_PASS:** There are two ways to enter your SMTP account info. Either enter a API key and secret key combination or enter your SMTP account email and your password.
- **MAIL_SERVER:** Enter the the name of the SMTP server of your provider. You can find the server's name in the SMTP configuration settings of your SMTP account. The *Mailjet* server is called "in-v3.mailjet.com", the *SendinBlue* server is called "smtp-relay.sendinblue.com".
- **MAIL_SENDER:** Enter your SMTP validated email address here. It is possible to enter the same email address for MAIL_USER and MAIL_SENDER.
- **MAIL_PORT:** Enter the port through which a connection to the server is established. Various ports are possible, click here for an overview. We found that port 465 works best for us. If you have troubles with port 465, try port 587.
- **DONATION_MINIMUM:** The minimal possible donation to make is 1 cent. This value must not be changed.

3.3.2 Parameters

The function requires the following six parameters:

- `self`: self is required by all functions in the oTree framework. It has no explicit use within this particular function.
- `weight_to_donate`: A float value used to pass the amount of carbon emission that is saved by the experimental participants.
- `unit`: A string value that defines the unit of the saved carbon emission. The following values are accepted: `["mg", "g", "kg", "t", "oz", "lbs", "st"]`
- `experiment_name`: A string value that specifies the name of the experiment (e.g. "Carbon Emission Task").
- `payment_e_mail_name`: A string that specifies the name of the person or team that receives the mail.
- `payment_e_mail_to`: A list containing the mail addresses of all recipients . If the mail is only to be sent to one address then a single string can be passed to the function.

3.3.3 Sequence of events

1. The `weight_to_donate` value is converted to metric tons. The conversion is based on the `unit` value.
2. The current CO2 price per ton for emission certificates is fetched from a price endpoint that is provided by Compensators.
3. The price of the carbon-emission certificate is calculated.
4. The contents of the email are defined.
    - The mail subject includes the `experiment_name` parameter.
    - The mail body includes the `payment_e_mail_name` parameter as an initial greeting. Furthermore, the body includes the total weight of carbon-emission saved, the current price per ton for carbon-emission certificates, as well as the link to Compensators donation form with the correct price to make the carbon-emission certificate purchase. These contents can be changed at will.
5. A connection to the SMTP server is established and the email is sent to all recipients specified in the `payment_e_mail_to` list.

3.3.4 Add the function to your Subsession class

Simply insert the function into the Subsession Class of your models.py file. The Subsession class should look something like this:

```
class Subsession(BaseSubsession):

    #------------------------
    #ALL YOUR OTHER CODE HERE
    #------------------------

    def send_payment_mail(self,
                          weight_to_donate: float,
                          unit: str = "t",
                          experiment_name: str = "Experiment Name",
                          payment_e_mail_name: str = "John Doe",
                          payment_e_mail_to: list = ["john.doe@gmail.com"]):

        #CONSTANTS:
        MAIL_USER = "API key or SMTP account email"
        MAIL_PASS = "API secret key or SMTP account password"
        MAIL_SERVER = "SMTP Mail server here e.g.: `in-v3.mailjet.com`"
        MAIL_SENDER = "validated.email@gmail.com"
        MAIL_PORT = 465
        DONATION_MINIMUM = 1

        #UNIT CHECK:
        unit_list = ["mg", "g", "kg", "t", "oz", "lbs", "st"]
        if unit not in unit_list:
            raise Exception("unit parameter ", unit, "not recognised. Unit has to be in
↪", unit_list)

        #CONVERT UNIT TO METRIC TONS:
        if unit == "mg":
```

(continues on next page)

(continued from previous page)

```
            weight_in_tons = weight_to_donate / 1000000000
        if unit == "g":
            weight_in_tons = weight_to_donate / 1000000
        if unit == "kg":
            weight_in_tons = weight_to_donate / 1000
        if unit == "t":
            weight_in_tons = weight_to_donate
        if unit == "oz":
            weight_in_tons = weight_to_donate / 35273.96198069
        if unit == "lbs":
            weight_in_tons = weight_to_donate / 2204.62262185
        if unit == "st":
            weight_in_tons = weight_to_donate / 157.47304442

        #GETTING THE CURRENT CO2 PRICE:
        price = 0
        try:
            price = requests.get("http://compensate.compensators.org/price.php").json()
            if 'price_per_ton' not in price:
                raise Exception("Price not found in data")
            price_per_ton = float(price['price_per_ton'])
        except:
            pass
        donation_in_cents = weight_in_tons * price_per_ton

        # CHECK DONATION MINIMUM
        if donation_in_cents < DONATION_MINIMUM:
            print("The donation is less than 1 cent, therefore too small. No Mail was␣
↪sent.")

        #SENDING THE PAYMENT MAIL
        else:

            #Define the body of the mail
            body = f"""Hello {payment_e_mail_name},

The participants in your experiment: "{experiment_name}" donated {weight_to_donate:.3f}
↪{unit} of CO2 Emission.
This equals to {weight_in_tons:.3f} tons of CO2. At the current price of {(price_per_ton␣
↪/ 100):.2f} € per ton this sums up to a total donation of {(donation_in_cents / 100):.
↪2f} €.

To authorize the payment, please click here:
https://www.spendenformular-direkt.org/forms/6944d11a-60d9-48a2-803f-b4b0c7797cb9?
↪default_amount_1_in_cents={donation_in_cents}

Best Regards
The Automated Donation system :)
          """

            #DEFINE MAIL SUBJECT ADD MAIL BODY:
            email_text = f"Subject: [{experiment_name}] Please confirm the donation for␣
↪the experiment\n\n{body}"
```

(continues on next page)

(continued from previous page)

```
            try:
                #CONNECT TO THE SMTP SERVER:
                server = smtplib.SMTP_SSL(MAIL_SERVER, MAIL_PORT)

                #LOGIN TO THE SMTP SERVER
                server.login(MAIL_USER, MAIL_PASS)

                #SEND THE EMAIL
                server.sendmail(MAIL_SENDER, payment_e_mail_to, email_text.encode('utf8',
↪ 'ignore'))
                server.close()
                print("Your mail has been sent successfully")

            except:
                print("Unable to send mail")
                traceback.print_exc()
```

In order to call the function some additional set up in your code is needed.

Note: **Source Code:** All source code described on this website can simply be copied into your *oTree* project from the code block at the bottom of this section.

3.4 Step 4: Adding additional code

Some additional code is needed to ensure that the email is sent at the right time, containing the correct data. The easiest way to do this, is to send the email (e.g. calling the function) after all participants have finished the experiment and the correct amount of saved carbon emission has been calculated.

In order to monitor the status of each participant and make sure that all players have finished the experiment, it is recommended to implement the following fields and functions in your models.py file.

3.4.1 Player is finished

You should add a Boolean field `is_finished` in the Player class, that states whether or not a player has finished the experiment. The initial value of this field should be set to `False`, and turn `True` once the player has completed the experiment. Add the following code to your Player class:

```
class Player(BasePlayer):
    #------------------------
    #ALL YOUR OTHER CODE HERE
    #------------------------

    is_finished = models.BooleanField(initial=False)
```

3.4.2 Subsession is finished

Secondly, a Boolean field `all_players_finished` should be added to your Subsession class that states whether or not all players in the Subsession have finished the experiment. This field has to be initialised as `False` and be set to `True` once every player has finished the experiment. In addition to this field a corresponding function `set_all_players_finished` must be added in the Subsession class. This function counts the total number of players that have finished the experiment and sets the `all_players_finished` field to `True` once all players have finished. Add the following code to your Subsession class:

```
class Subsession(BaseSubsession):
    #------------------------------------------------
    #ALL YOUR OTHER CODE HERE
    #def send_payment_mail(...) should be here too
    #------------------------------------------------

    all_players_finished = models.BooleanField(initial=False)

    def set_all_players_finished(self):
        sum_finished = 0
        for p in self.get_players():
            if p.is_finished:
                sum_finished += 1

        if sum_finished == self.session.num_participants:
            self.all_players_finished = True
```

After this code is implemented in your models.py file, the function can be called at the correct time, including the correct data.

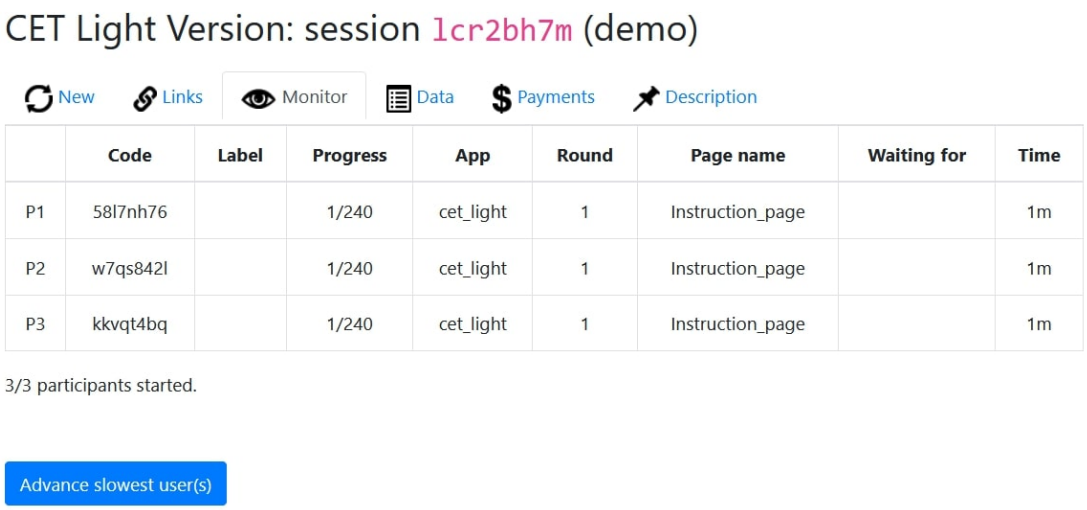
Note: Source Code: All source code described on this website can simply be copied into your *oTree* project from the code block at the bottom of this section.

3.5 Step 5: Calling the function

This is the last step of integrating the software-module in your *oTree* project. You need to add the following things in your pages.py file.

3.5.1 Adding Timeouts

The email should be sent once all players have finished the experiment. Since it's impossible to guarantee that every single player finishes the experiment you have to account for players that have dropped out and might not finish the experiment on their own. One way to do this is to manually force a timeout by clicking the "Advance slowest participants" button in *oTree's* admin interface. Like this:



However, this can also be done more elegantly by adding a Timeout to every single page of your experiment. By adding a timeout to every Page class in your pages.py file you don't have to manually advance the players and you can still make sure that every player finishes the experiment. Add the following code to all your page classes:

```
class Page1(Page):
    #------------------------
    #ALL YOUR OTHER CODE HERE
    #------------------------

    timeout_seconds = XX  # add amount of seconds until timeout happens
```

3.5.2 Before Next Page

Lastly, the following lines have to be added to the last page of your experiment. All code within the `before_next_page()` function is executed once the player finishes the last page of your experiment. Click here for additional information. The code below does the following:

1. Once a player finishes the experiment the `is_finished` field of the player is set to `True`
2. The `set_all_players_finished()` function checks if every player has finished the experiment.
3. Once the last player finishes and therefore all players have finished the experiment, the `send_payment_mail()` function is called and the email is sent.

This is an example of possible parameters for the function:

Mail parameters:

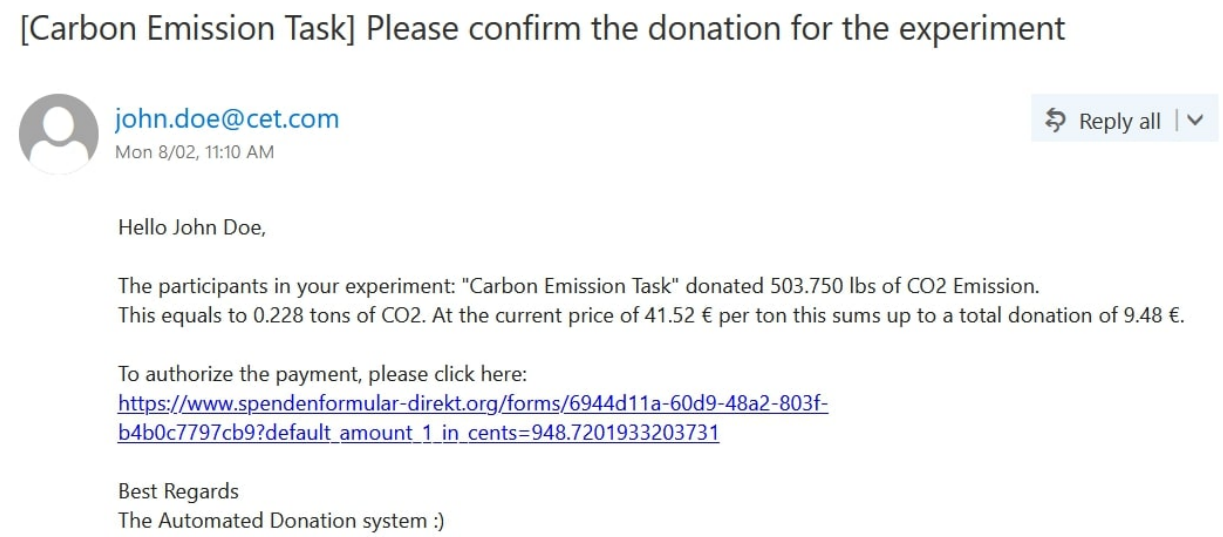
- The `sum_saved_emission` field is the total weight of CO2 emission that was saved by participants. Add your own variable here.
- The unit of the weight is lbs.
- The name of the experiment is Carbon Emission Task.
- The name of the recipient is John Doe.
- the recipient's email address is john.doe@cet.com. (Multiple addresses have to be specified in a list e.g. ["john.doe@cet.com", "jane.doe@cet.com"].

By adding the following code to the last page of your experiment, you successfully integrated the tool for automatic carbon offsetting in your *oTree* project.

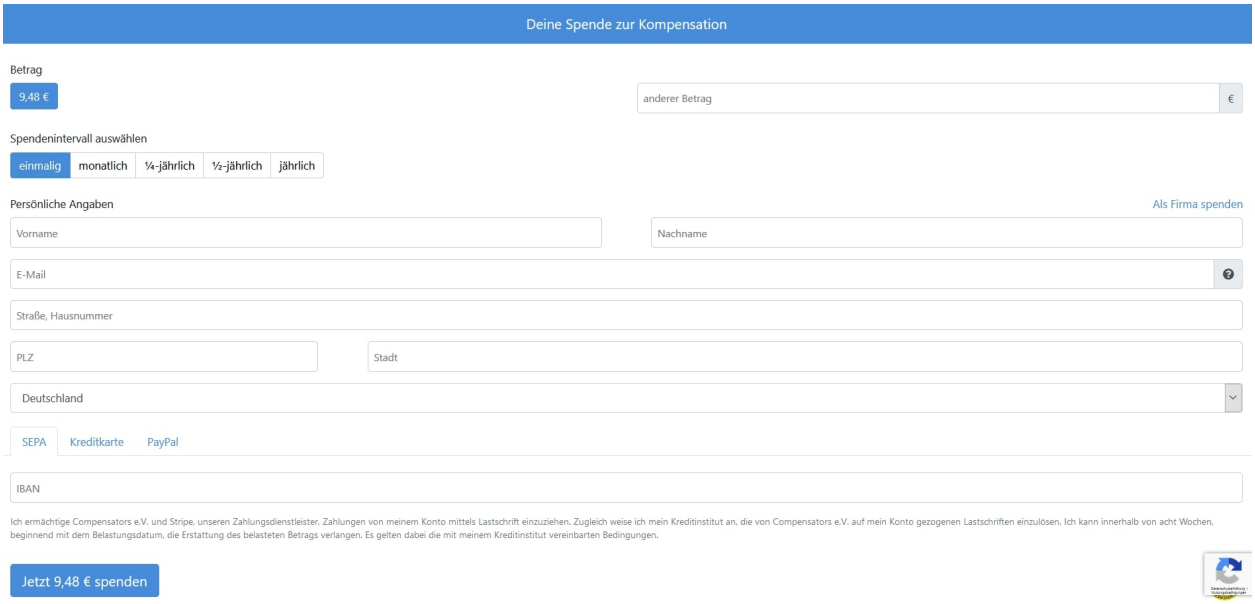
```
class LAST_PAGE(Page):
    #------------------------
    #ALL YOUR OTHER CODE HERE
    #------------------------

    def before_next_page(self):
        #Is Finished fields and functions
        self.player.is_finished = True
        self.subsession.set_all_players_finished()

        # All finished check and send mail
        if self.subsession.all_players_finished:
            self.subsession.send_payment_mail(self.subsession.sum_saved_emission,
                                              "lbs",
                                              "Carbon Emission Task",
                                              "John Doe",
                                              "john.doe@cet.com")
```

3.6 Examples

This is an example of a generated email:



The link directs you to the donation form, where the carbon-emission certificate purchase can be made. The donation form looks like this:


Deine Spende zur Kompensation
Betrag
9,48 €
anderer Betrag
€
Spendenintervall auswählen
einmalig
monatlich
¼-jährlich
½-jährlich
jährlich
Persönliche Angaben
Als Firma spenden
Vorname
Nachname
E-Mail
Straße, Hausnummer
PLZ
Stadt
Deutschland
SEPA
Kreditkarte
PayPal
IBAN
Ich ermächtige Compensators e.V. und Stripe, unseren Zahlungsdienstleister, Zahlungen von meinem Konto mittels Lastschrift einzuziehen. Zugleich weise ich mein Kreditinstitut an, die von Compensators e.V. auf mein Konto gezogenen Lastschriften einzulösen. Ich kann innerhalb von acht Wochen, beginnend mit dem Belastungsdatum, die Erstattung des belasteten Betrags verlangen. Es gelten dabei die mit meinem Kreditinstitut vereinbarten Bedingungen.
Jetzt 9,48 € spenden