

# **RUNNING EXPERIMENTS ONLINE**

AN INTRODUCTION TO PCIBEX FARM

RUNYI YAO

UNIVERSITY OF OXFORD

11 NOV 2022

# PREREQUISITES

Building experiments on PCIBex doesn't require any background in coding.

However, you should have some basic knowledge of **experiment design**.

A beginner in experimental linguistics? Here are some resources that might be helpful:

- A brief video introduction of linguistic experiment:  
[https://www.youtube.com/watch?v=4sR\\_toy4YjA](https://www.youtube.com/watch?v=4sR_toy4YjA)
- A book that is available on SOLO:  
Gillioz, C., & Zufferey, S. (2020). *Introduction to experimental linguistics*. John Wiley & Sons.

1. What is PClbex Farm?
2. Why PClbex?
3. Creating a basic production task
4. Other software for web-based linguistic experiments

# WHAT IS PCIBEX FARM?

# WHAT IS PCIBEX FARM?


A free, open-source, versatile, and user-friendly online experiment builder that was developed and is maintained by Jeremy Zehr and Florian Schwarz.

<https://farm.pcibex.net/>



## What you can do with PCIBex:

- Building behavioral experiments online
- Running behavioral experiments online



A screenshot of the PCIBex web application interface. It features a vertical list of experiment templates, each preceded by a small icon of a document with a checkmark. The templates are: 'Empty project', 'Masked Priming', 'Stroop Task', 'Self-Paced Reading', 'Covered Box Experiment', 'Mouse Tracking', 'Oral Production (MediaRecorder)', and 'EyeTracker'. The interface has a clean, modern design with a light gray background and a white sidebar on the right.

- Empty project
- Masked Priming
- Stroop Task
- Self-Paced Reading
- Covered Box Experiment
- Mouse Tracking
- Oral Production (MediaRecorder)
- EyeTracker

**WHY PCIBEX?**

# WHY PCIBEX?

## Pros

1. Convenience
2. Larger participant pools
3. Flexible & Multi-function
4. User-friendly
5. Free

## Cons

1. Delayed & unstable
2. Internet access required



# **CREATING A BASIC PRODUCTION TASK**

- **Question:** In English, the voiceless unaspirated stops (i.e., [p, t, k]) can be derived either from the underlying voiceless stops (i.e., /p, t, k/) or the underlying voiced stops (i.e., /b, d, g/). Will the underlying phonological form influence the onset  $F_0$  of vowels following voiceless unaspirated stops in British English?
- **Aim:** Displaying stimuli words randomly and recording (optional)
- **Sequence:** Consent - Instruction - Practice trial  
-Experimental trial - Save results - End

# SETTING UP

The screenshot displays the PennController web interface. At the top, a status bar shows 'Projects — 0.28/64MB (0%)', 'each PCibex Farm account has a 64MB storage quota', and 'Empty project\_copy (0MB)'. The main interface is divided into several panels:

- Left Panel:** Contains 'Folders and Files' and 'Scripts (0MB)'. The 'Scripts' panel shows a file named 'main.js' (0KB) labeled 'experiment script'.
- Top Center:** A code editor panel with a red border containing the following JavaScript code:

```
1 PennController.ResetPrefix()
2
3- newTrial(
4   newButton("Hello world")
5     .print()
6     .wait()
7 )
```
- Right Panel:** An 'Actions' panel with a 'documentation' link and a list of actions: 'Unpublished', 'Results', 'Share', 'Download', 'Git Sync', and 'Settings'.
- Bottom Center:** A 'Preview experiment' area showing a 'Hello world' button and a 'progress' indicator.
- Bottom Right:** A 'Debugger' window titled 'Debug (PennController 2.0)' showing 'No errors found'.
- Bottom Left:** A 'Message' box stating: 'This is an unsaved copy of the experiment named Empty project (slug CVCGXgf)'.

or: <https://github.com/yao-ry/PCibex-demo-project.git>



# CREATING A NEW TRAIL

Trials are building blocks that you combine to create an experiment. They are composed of **elements** (things that have content, such as **Text** and **Button**) and **commands** (things that make elements do things, such as **.print** and **.center**).

```
//Consent
newTrial "consent"
    ,
    newText("consent-demo", "This is a demo.")
        .center()
        .print()
    ,
    newButton("Continue.")
        .center()
        .print()
        .wait()
    ;
```



# CREATING NEW ELEMENTS

Elements may contain multimedia content or/and interactive content.

PennController has 21 types of elements, including:

- Text: Text content
- Button: A clickable button
- Html: An HTML document.
- Timer: A timer.
- Audio: Audio content that can interact with the experiment script

...



```
//Consent
newTrial  "consent"
,
  newText("consent-demo", "This is a demo.")
  .center()
  .print()
,
  newButton("Continue.")
  .center()
  .print()
  .wait()
;
```

# CREATING NEW ELEMENTS

Syntax to create new elements:

`newX(ELEMENT_NAME, Y)`



- X is a type of element
- ELEMENT\_NAME is the name of the element
- Y are optional parameters that depend on the element type. For example, if you create a Text element you'll also need to specify the content.

```
//Consent
newTrial  "consent"
    ,
    newText("consent-demo", "This is a demo.")
        .center()
        .print()
    ,
    newButton("Continue.")
        .center()
        .print()
        .wait()
;
```

# CALLING COMMANDS

Commands can manipulate elements and the content they contain. There are three types of commands:

- **Element commands:** Used within a trial and called on an element.
  - ▶ Action command: Directly manipulates an element.
  - ▶ Test command: Runs a test on an element.
- **Global commands:** Used outside of a trial.
- **Special commands:** Used within a trial, but not called on an element.

```
//Consent
newTrial  "consent"
    ,
    newText("consent-demo", "This is a demo.")
    .center()
    .print()
    ,
    newButton("Continue.")
    .center()
    .print()
    .wait()
;
```

# TRY IT!

Now we have created the first trial called 'consent'. Can you try to create an instruction trail in this way by yourself?



*progress*

Welcome!

In this task, you will need to read some words aloud.

Your reading will be recorded.

[Click to see some examples.](#)

```
//Consent
newTrial ["consent"
  ,
  newText("consent-demo", "This is a demo.")
  .center()
  .print()
  ,
  newButton("Continue.")
  .center()
  .print()
  .wait()
];
```



*progress*

This is a demo.

[Continue.](#)



# TRY IT!

```
//Instruction
newTrial("instructions",
  defaultText
    .center()
    .print()
  ,
  newText("instructions-1", "Welcome!")
  ,
  newText("instructions-2", "In this task, you will need to read some words aloud.")
  ,
  newText("instructions-3", "Your reading will be recorded.")
  ,
  newButton("wait", "Click to see some examples.")
    .center()
    .print()
    .wait()
```

`defaultText.X()`: Implicitly inserts the command `.X` below each `newText` element in the trial. (You can set defaults for all types of elements following this pattern.)

# DISPLAYING STIMULI

It's not difficult to create a trial to display stimuli following the previous steps.

However, here is a new question: What if we have hundreds of stimuli?

```
//practice trials

newTrial("practice-1",

    newText("explain", "Please read the following word aloud:")
        .center()
        .italic()
        .print()
    ,

    newText("word1", "Oxford")
        .center()
        .print()
    ,
    newButton("wait", "Next")
        .center()
        .print()
        .wait()
);

newTrial("practice-2",

    newText("explain", "Please read the following word aloud:")
        .center()
        .italic()
        .print()
    ,

    newText("word2", "linguistics")
        .center()
        .print()
    ,
    newButton("wait", "Next")
        .center()
        .print()
        .wait()
);
```

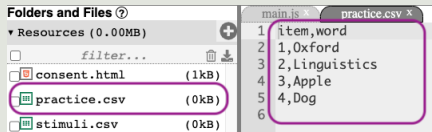
## CREATING A TRIAL TEMPLATE

A **trial template** is a template for creating trials: it's a trial that's partially complete, with some values (represented by variables) to be added in "later". PennController fills in the variables with values from a specified CSV table, which is stored in the 'Resources' of the project.

# CREATING A TRIAL TEMPLATE

A **trial template** is a template for creating trials: it's a trial that's partially complete, with some values (represented by variables) to be added in "later". PennController fills in the variables with values from a specified CSV table, which is stored in the 'Resources' of the project.

Item	word
1	Oxford
2	linguistics
3	apple
4	dog



# CREATING A TRIAL TEMPLATE

To define a trial template, we'll use the global command `Template`.

Syntax:

```
Template(TABLE_NAME, row =>  
  newTrial("TRIAL_LABEL",  
    newX("ELEMENT_1", row.COLUMN_NAME)  
    .COMMAND_1()  
  )  
)
```

- `TABLE_NAME` is the name of a table.
- `row` is an array variable.
- `newX` is an element corresponding to the type of an array variable in the CSV table.

# CREATING A TRIAL TEMPLATE

```
//practice trail
Template("practice.csv", row =>

    newTrial("practice-trial",

        newText("explain", "Please read the following word aloud:")
        .center()
        .italic()
        .print()
        ,

        newText("practice-word", row.word)
        .center()
        .print()
    )
);
```

## CREATING A TRIAL TEMPLATE

In fact, the experiment design can be much more complicated. In this example experiment, we will have a 2x3 design:

item	PoA	underlying	word
1	bilabial	voiceless	spot
1	bilabial	voiced	bot
1	alveolar	voiceless	stock
1	alveolar	voiced	dock
1	velar	voiceless	scot
1	velar	voiced	got

# CREATING A TRIAL TEMPLATE

Sometimes, we'll also need a within-subject design which would allow every participant to see items from every conditions:

item	group	POA	underlying	word
1	A	bilabial	voiceless	spot
1	B	bilabial	voiced	bot
1	C	alveolar	voiceless	stock
1	D	alveolar	voiced	dock
1	E	velar	voiceless	scot
1	F	velar	voiced	got
2	B	bilabial	voiceless	spark
2	C	bilabial	voiced	bark
2	D	alveolar	voiceless	stark
2	E	alveolar	voiced	dark
2	F	velar	voiceless	scarf
2	A	velar	voiced	gart



When you run an experiment, by default PennController logs only when each trial starts and ends. You will need the `log` command if you want to collect additional information about the trial.

There are two ways to use the `log` command:

- `.log( )`: called on an element, and logs information as a row in the results file.
- `log(NAME, VALUE)`: called on a trial, and logs information as a column in the results file.

# LOGGING DATA

In the command `log(NAME, VALUE)`:

- NAME: the name of the column added to the results file
- VALUE: the value added to each row of the results file, in the column indicated by NAME.

```
// Experimental trial

Template("stimuli.csv", row =>
  newTrial("experimental-trial",

    newText("explain", "Please read the following word aloud:")
      .center()
      .italic()
      .cssContainer({"margin-bottom": "1em"})
      .print()
    ,

    newText("word", row.word)
      .center()
      .settings.css("font-size", "15em")
      .cssContainer({"margin-bottom": "1em"})
      .print()
    ,

    newButton("next", "Next")
      .center()
      .print()
      .wait()

    .log("POA", row.PoA)
    .log("underlying", row.underlying)
    .log("item", row.item)
  );
```

## SOME NOTES

Here are some additional questions:

Can I make the stimuli play automatically?

Yes. You can add a new element Timer.

## SOME NOTES

Here are some additional questions:

Can I make the stimuli play automatically?

Yes. You can add a new element `Timer`.

Can I randomize the order of items?

Yes. We can use global commands `sequence` and `randomize`.

## SOME NOTES

Here are some additional questions:

Can I make the stimuli play automatically?

Yes. You can add a new element `Timer`.

Can I randomize the order of items?

Yes. We can use global commands `sequence` and `randomize`.

Can I record with the project?

Yes. You can create a new element `MediaRecorder` to collect audio or video data with the project. However, you will also need to have access to a server where you can upload and execute a PHP file. [Here](#) is more information if you are interested.

If you want to customize your experiment:

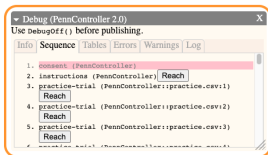
- Using CSS styles for font size, color and spacing.
- Sending results manually by adding a global command `SendResults`
- Hiding the progress bar by adding `showProgressBar=false`

.....

## Let's see the project!

- Demo Link: <https://farm.pcibex.net/r/bfykzp/>
- Data collection Link: <https://farm.pcibex.net/p/XdgCMW/>

This is a demo version of this experiment. [Click here to edit a copy in the PCIBex Farm.](#)



progress

### Production Task

#### Participant Information Sheet

It is highly recommended that you complete this task in Firefox or Google Chrome on a computer. Safari and Internet Explorer may not be fully supported.

Please note: you may only complete this task once.

You are invited to participate in an approximately 30 minute research study investigating native speakers' pronunciation of several English words. To participate, you need to be native speakers of Standard Southern British English aged 18 or over. If you choose to participate, your answers in this task will be collected and will be stored in researchers' personal computers for further analysis in this study. Your participation in this survey is anonymous: researchers will not receive any identifying information about you.

Do you consent to participate in this study?

Please delete this sentence and leave your name (or participant ID) here.

I have read the consent statement and I agree to continue.

When you are 100% done designing your experiment:

- Using `DebugOff( )` to hide the debugger
- Publishing the project and sending out the data collection link.
- Downloading the result file (CSV) and reading it in R.



# **OTHER SOFTWARE FOR WEB-BASED LINGUISTIC EXPERIMENTS**

Tools for building web-based experiments:

- jsPsych
- psychopy
- Gorilla

Platforms for deploying experiments:

- Prolific
- Amazon's Mechanical Turk

THANKS!