User Manual

Graphical User Interface for the Psychoacoustic Listening Laboratory (PALILA)

For version v1.2.1

Josephine Pockelé

Delft University of Technology

PALILA GUI

TU Delft

PALILA

– This page is intentionally left blank. –

Contents

1

Introduction

The *Graphical User Interface for the PsychoAcoustic Listening Laboratory*[1] *(PALILA GUI)* is a free, open-source, Python software developed by Josephine Pockelé to conduct listening experiments at the Delft University of Technology, Faculty of Aerospace Engineering.

BEFORE YOU START
This software is designed for experiments with human subjects. Researchers should always ensure the proper ethics approvals and data management strategies are in place!
The output from this software is not anonymous. While tools are provided to anonymise the data, researchers are always responsible to ensure the data management adheres to the rules of their organisation(s) / project(s).

NOTE for data management:
The output files of experiments are placed inside the folder where the GUI software is located. Keep this in mind during installation.

The interface is designed to allow for self-paced participation, without involvement of the responsible researchers during the listening part of the experiment. The GUI has a simple look with muted, neutral colours to avoid emotional interference with the results. Buttons are designed for use with touchscreen devices, to create better accessibility for participants.

The highlight features of the PALILA GUI are:

- A built-in questionnaire system to collect population and hearing health statistics,
- Different types of questions, to enable a variety of listening tests,
- Ability to compare two sound samples,
- Simple result format, in the common csv format,
- Experiment configuration in an easy-to-understand language.

This manual has the following chapters:

2) A guide to get started with this software.
3) A thorough example to follow along while learning the configuration language.
4) Full explanation of all GUI components related to the sound samples in an experiment.
5) Full explanation of all GUI components related to the questionnaire system.

[1] More information about the facility: `https://www.tudelft.nl/lr/palila`

2

A Quick Start Guide

This chapter covers the basics of using this software. The sections will cover:

2.1) Software installation.

2.2) The structure of an experiment in the PALILA GUI.

2.3) Creating your first experiment.

2.4) The syntax of the experiment configuration (.PALILA) file.

2.5) Parameters controlling the overall experiment.

NOTE: this software does not use Windows-specific Python packages, but compatibility with *macOS*, *Linux*, and other Unix-based systems is not guaranteed.

IMPORTANT: the interface is currently only designed for a **screen resolution of 1920x1200, with 150% scaling.** Different display settings will result in visual defects in the interface.

NOTE: support for other screen ratios and screen resolutions will be added in a future version.

2.1. Installing the GUI

First things first, let's get everything set up so you can run this GUI:

1. Download the latest version of the PALILA GUI software from *Zenodo*[1],
2. Unpack the *.zip* file into the folder where you want to use the GUI,
3. Install *Python 3.12*[2].

IMPORTANT: during Python installation, **select "*Add Python 3.12 to PATH*"**.

After these steps, run the *SETUP.bat* script. This will create the virtual environment required to run the GUI. In case the script closes without prompting to "*Press enter to exit...*", use the following steps to set up the virtual environment:

1. Open a terminal window in the GUI directory.
2. Create a Python virtual environment[3], with the name "*venv*".
3. Update the virtual environment using the following command:
   .\venv\Scripts\python.exe -m pip install --upgrade pip setuptools
4. Install the required Python packages into the virtual environment:
   .\venv\Scripts\python.exe -m pip install -r .\support_scripts\requirements.txt

Now, you should be good to go! Run *PALILA.bat*, and the GUI should appear in full-screen mode on your primary display. You can exit the GUI at any point with *Alt+F4*.

[1] Download available: `https://doi.org/10.5281/zenodo.15100497`
[2] Download available: `https://www.python.org/downloads/release/python-31210/`
[3] For further instructions, see: `https://docs.python.org/3.12/library/venv.html`

2.2. Listening Experiment Structure

This section is a short explanation of the experiment structure in this software. Figure 2.1 provides a visual representation. This overall structure cannot be altered without modifying the source code of the interface. Many aspects of this structure can be activated/deactivated, depending on the needs of the experiment.

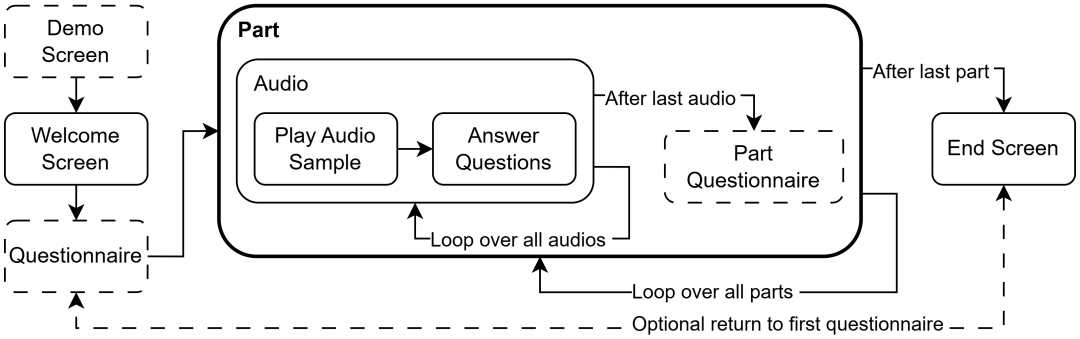


Figure 2.1: Overall structure of an experiment in the PALILA GUI.

The experiment starts with a *welcome screen*, or the optional *demonstration screen*. This demonstration can be used to show participants how to use the interface. The *welcome screen* shows a welcoming message and sets the participant ID.

The experiment can optionally start with a *questionnaire*. The software contains a default set of questions for establishing population statistics and hearing health.

The listening portion of the experiment is divided into *parts*. This grouping of *audios* can distinguish between different tests within one experiment. Participants have to listen to each *audio*, and answer one or two *question(s)*. At the end of each *part*, a questionnaire can be added to ask further questions related to that part.

The experiment ends with the *end screen*. On this screen, participants can return to the first questionnaire to check their answers. After finishing the experiment, a *goodbye* message will appear.

NOTE: during the experiment, the GUI can only be closed with *alt + F4*. After the *end screen*, *escape* will also close the interface.

NOTE: an optional questionnaire at the end of the experiment will be added in a future version.

2.3. Creating a New Experiment

Creating a listening experiment is a very simple process: run the *NEW_EXPERIMENT.bat* script and enter a name for your experiment when prompted. This will create one file and one folder with the name you entered. It also modifies line 20 in *PALILA.bat* to tell the GUI which experiment configuration to use.

IMPORTANT: line 20 in *PALILA.bat* tells the PALILA GUI which experiment configuration to use. **Always ensure line 20 in *PALILA.bat* contains your current experiment name!**

NOTE: to copy your experiment to another device, transfer:
1. the *.palila* file,
2. the experiment folder,
3. the modified *PALILA.bat* script.

Figure 2.2 shows the structure of the experiment created by the *NEW_EXPERIMENT.bat* script. This is one of the simplest possible configurations.

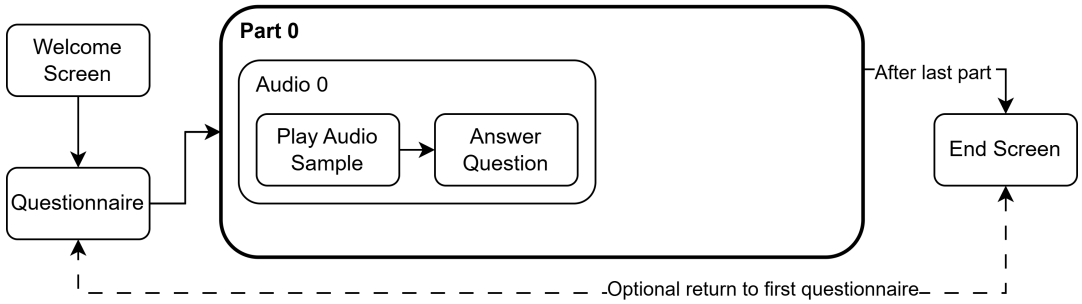


Figure 2.2: Structure of the new experiment.

The PALILA file can be opened with your favourite text editor or IDE (e.g.: Windows Notepad, VIM, Notepad++, etc). The configuration from *NEW_EXPERIMENT.bat* will contain the following code:

```
1  ## This is your experiment input file.
2  ## It is set up with a basic structure for a PALILA GUI experiment.
3
4  pid mode = auto
5  randomise = no
6  demo = no
7
8  [questionnaire]
9      default = yes
10
11 [part 0]
12     randomise = yes
13
14     [[questions]]
15         [[[question annoyance]]]
16             type = Annoyance
17
18     [[audio 0]]
19         filename = 'tone500Hz.wav'
```

The first lines of this file define parameters to control the overall experiment (see section 2.5):

- Line 4: the GUI will set the participant's ID to a timestamp.
- Line 5: the interface will keep the order of the *parts* in the PALILA file.
- Line 6: disables the optional *demo screen* of the GUI.

Lines 8-9 add the questionnaire at the start, containing the default set of questions (see chapter 5).

The remaining lines define the listening part of the experiment (see chapter 4):

- Line 11: defines the name of the *part*. Each part can be given a descriptive name, which will be used for outputting the answers.
- Line 12: the interface will shuffle the different *audios* within *part* '0'. *Audios* in other *parts* are not affected.
- Line 14-16: this block defines the *questions* used for all *audios* in *part* '0'. In this case, the GUI will display an 11-point ICBEN annoyance rating scale to the participants.
- Line 18-19: defines a single *audio* file in *part* '0', which is the default 500 Hz tone in the folder with the name of the experiment.

2.4. PALILA File Syntax

With the setup out of the way, let's look at the actual syntax of the PALILA file. The PALILA GUI uses the *ConfigObj* package to read PALILA files, hence the basic syntax is the same as described in their documentation[4]. This section will shortly summarise the most relevant items for the PALILA GUI.

The example below shows the syntax to define parameters with values and add comments with hashtags (#). In the parameter definition, everything before the equal sign (=) sets the parameter name, while everything after the equal sign sets the value. After a parameter definition, a comment can be added with a hashtag.

```
# This line is a comment
parameter = value  # this is also a comment
```

NOTE: leading tabs and spaces are ignored in the parameter names. Trailing tabs and spaces are ignored in the values.

String values

A String value can contain any form of text. There are multiple ways to define a string value:

```
parameter = Without quotes
parameter = 'With single quotes'
parameter = "With double quotes"
parameter = '''With triple quotes'''
parameter = '''A string which spans multiple lines
    should always be in triple string quotes.'''
```

IMPORTANT: strings that appear in the GUI are always wrapped to fit their assigned space. When too many lines are needed to fit inside this area, the first line(s) will be cut.

NOTE: to allow for non-functional indentation in the PALILA file, leading tab characters are removed from each line of a string value. **To add indentations in the interface, use spaces.**

Numerical values

The PALILA GUI uses two types of numerical values: *integers* and *floats*. When defining numbers, it is very important to use the correct representation.

Integer parameters will only accept purely integer numbers, such as '2', '40', '5000'. Any other form will result in a conversion error in Python.

Floating point numbers can be defined in three ways:

1. integer (e.g. '2', '40', '5000', ...),
2. float in decimal notation (e.g. '1.2', '4.56', ...),
3. float in scientific notation (e.g. '1.2e3', '4.56e7', ...)

Boolean values

There are multiple ways to define a boolean value[5]. The four pairings below are guaranteed to work (both capitalised and lower case):

True	1	Yes	On
False	0	No	Off

[4] See: `https://configobj.readthedocs.io/`

[5] See the "as_bool" function: `https://configobj.readthedocs.io/en/latest/configobj.html#section-methods`

List of string values

For some parameters, multiple string values have to be entered. This can be done using a list of string values. To avoid confusion, the single quote string syntax is recommended. Other notations of strings (no quotes, or triple quotes) may work, but this is not guaranteed.

IMPORTANT: multi-line strings do not work correctly in lists of string values.

NOTE: in a future version, a fix for the above-mentioned limitation will be considered.

Code blocks

Next to the different parameter types, PALILA files have a hierarchy in the form of blocks and sub-blocks. In the current state of the GUI, there will be three levels of blocks. Blocks on the first level are defined with single square brackets [ ]. Each level down the hierarchy requires an extra set of brackets. The use of these blocks will be further explained in chapter 3 with an example configuration. In the examples throughout this manual, indentation is used as a visual aid to represent the block level.

```
[Level 1 block]
    [[Level 2 block]]
        [[[Level 3 block]]]
```

NOTE: Indentations in PALILA files only serve as visual aids to represent the block levels. The block level is always determined by the last set of square brackets!

NOTE: always use the TAB character for non-function indentations. Some IDEs will put 4 spaces instead, which will cause (visual) errors.

2.5. Experiment Parameters

This section covers the parameters at the top of a PALILA file, which control overall aspects of the interface and the experiment structure.

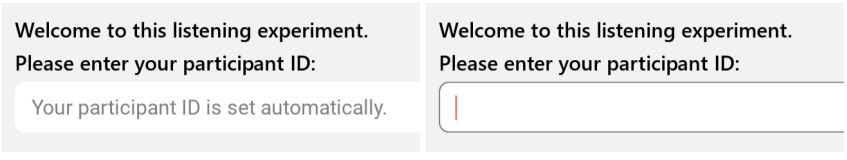
IMPORTANT: These parameters do not belong to any block. They should always be defined above the first block in the configuration file!

pid mode

Required parameter: sets mode of identifying participant results. Options: auto, input.

The Participant IDentification mode defines how the individual participant results are identified in the output filenames. Currently, two modes are accepted:

auto: Assigns the timestamp containing date and time of the participant starting the experiment.
input: An ID will be manually entered in the *welcome screen*.



(a) *auto* **(b)** *input*

Figure 2.3: Difference between the PID modes, in the *welcome screen*

When set to *auto*, a timestamp of the date and time of clicking *continue* in the *welcome screen* is used as participant ID. Example: the participant starts on 23 April 2025 at 09:41, generating the filename: *250423-0941.csv*.

In *input* mode, the result file will carry the ID entered in the *welcome screen*. Example: the participant enters 'palila123', generating the filename: *palila123.csv*.

welcome

Optional parameter: defines the welcome message. String value.

This parameter defines a custom message on the *welcome screen*. The "*Please enter your participant ID:* " line will always be added at the end. The default message is (see Figure 2.3):

"*Welcome to this listening experiment.*"

goodbye

Optional parameter: a custom message at the end of the experiment. String value.

At the end of the experiment, after clicking the '*Finish Experiment*' button, participants will see the *goodbye* message. The *goodbye* message shows up in the middle of the screen, with centred alignment. The default message is:

"*Thank you for your participation!*"

randomise

Optional parameter: activates shuffling of the experiment parts. Boolean value (default = 'no').

Normally, the experiment parts will appear in the order of the configuration file. The two code snippets below will ALWAYS result in the following order: "... → *part beta* → *part alpha* → ..."

```
[part beta]
    ...
[part alpha]
    ...
```

```
randomise = no

[part beta]
    ...
[part alpha]
    ...
```

When *randomise* is set to 'yes', some participants will receive the order "... → *part beta* → *part alpha* → ..."
Other participants will get the order "... → *part alpha* → *part beta* → ..."

NOTE: this *randomise* always shuffles all experiment *parts*. This parameter does not affect the order of samples within each *part* (see section 4.1).

```
randomise = yes

[part beta]
    ...

[part alpha]
    ...
```

demo

Optional parameter: activates the default demonstration screen. Boolean value.
Optional block: defines a custom demonstration screen. Level 1 block.

By default, the demonstration screen (see Figure 2.1) is deactivated (*demo = no*). The demonstration screen can be configured in two ways. To show the default demonstration comes with one audio and the ICBEN 11-point annoyance scale question (see subsection 4.3.5). Only a single line of code is required:

```
demo = yes
```

The default demonstration screen will show the elements in Figure 2.4:

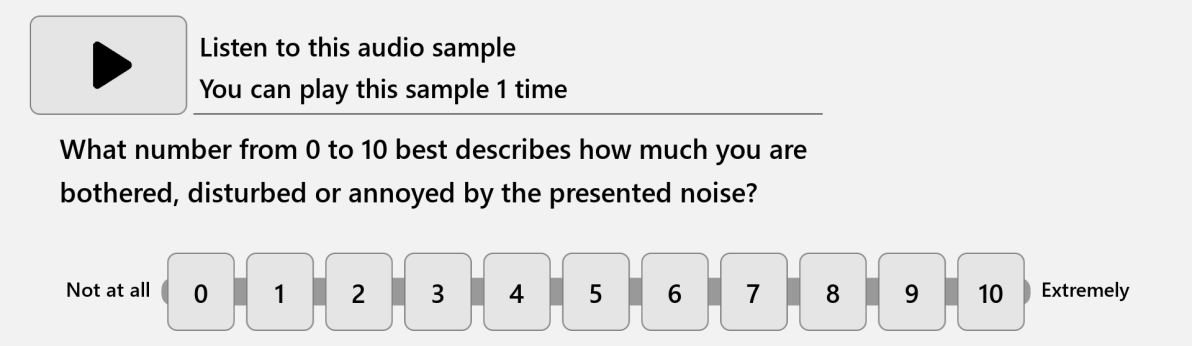


Figure 2.4: Audio and question elements of the default demonstration screen.

IMPORTANT: the default demo screen requires *tone500Hz.wav* in your experiment folder. You can retrieve this audio file from *GUI/assets* inside the repository.

The second method is defining a *demo* block. This block has the same structure as an *audio* block, which is described in subsection 4.1.5. A *demo* block uses the following code snippet:

```
[demo]
    filename = <string value>       # (optional)
    filename_2 = <string value>     # (optional)
    max replays = <integer value>   # (optional)

    [[question 1]]                  # (optional)
        ...
    [[question 2]]                  # (optional)
        ...
```

3

Example Configuration File

To show how the PALILA file's block structure works, this chapter follows an example of building a complex experiment. The overall structure of the final experiment is shown in Figure 3.1. The final PALILA file is given in Appendix A, and included in the code repository as "support_scripts/example.palila".

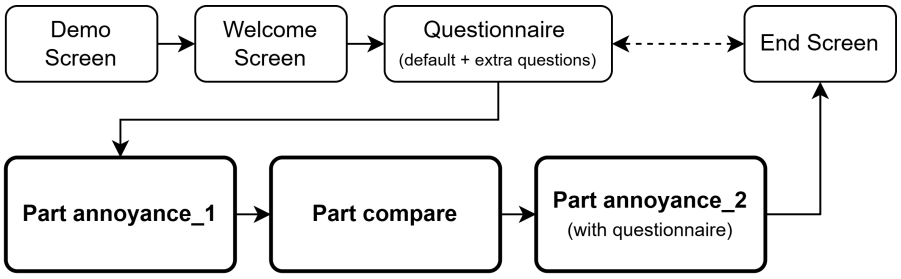


Figure 3.1: Final structure of the example experiment.

We start with the PALILA file from the *NEW_EXPERIMENT.bat* script, shown in section 2.3. First, we will set up the overall experiment parameters from section 2.5. We want an automatic participant ID, the default *welcome* and *goodbye* messages, and no randomisation of the parts, as well as the default demonstration screen. This requires the following code:

```
4 pid mode = auto
5 randomise = no
6 demo = yes
```

3.1. Questionnaire

While we want the default questionnaire, the experiment also calls for a few extra questions. A diagram of the desired layout of the questionnaire for this experiment is shown in Figure 3.2:

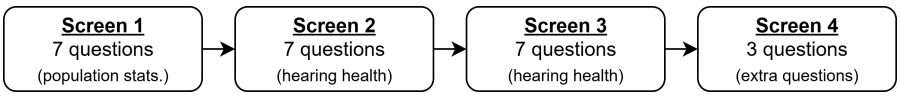


Figure 3.2: Diagram of the experiment questionnaire structure.

Let's add our extra questions to the questionnaire block. The code is shown below. Most importantly, these three question blocks start with double brackets, since they are on the second block level. The last question ('*live_distance*') is set up so it only needs to be answered when the question '*live_near*' is answered with '*Yes*'. The other parameters are further explained in chapter 5.

```
[questionnaire]
  default = yes

  [[question heard_before]]
    text = 'Have you ever heard wind turbine noise before?'
    type = MultipleChoice
    choices = 'Yes', 'No', 'Not sure'

  [[question live_near]]
    text = 'Can you see a wind turbine from your home?'
    type = MultipleChoice
    choices = 'Yes', 'No'

  [[question live_distance]]
    text = '''If yes, what is the approximate distance from
        your home to this/these wind turbines?'''
    type = MultipleChoice
    choices = '< 500 m', '500 - 1000 m', '1 - 2 km', '> 2 km'

    unlocked by = 'live_near'
    unlock condition = 'Yes'
```

The additional questions will appear on the fourth questionnaire screen and will look like Figure 3.3:

Figure 3.3: The extra screen of questions in the questionnaire of the experiment.

3.2. Part 1 - Annoyance (1)

The first part of the experiment's main body is called 'annoyance_1' (see Figure 3.1). Within this part, we want the questions to be in a random order for each participant. We therefore start the *part* block with:

```
[part annoyance_1]
  randomise = yes
```

To give more context to the participants, we will add a *intro* block with a custom text, and set a hold *time* of 10 seconds. Note that *intro* is again in double brackets, to indicate it is inside the *part* block.

```
  [[intro]]
    text = '''Thank you for filling out our questionnaire!

      In the first part of the listening experiment, you will listen
      to noise recordings of a wind turbine.

      While listening, imagine that this is the sound situation
      outdoors (e.g. your garden or a park).'''

    time = 10
```

The resulting introduction screen will look like Figure 3.4. The blue timer bar on top of the *continue* button will last 10 seconds, and the *continue* button will unlock once the timer bar is filled. This introduction screen is further explained in subsection 4.1.1.

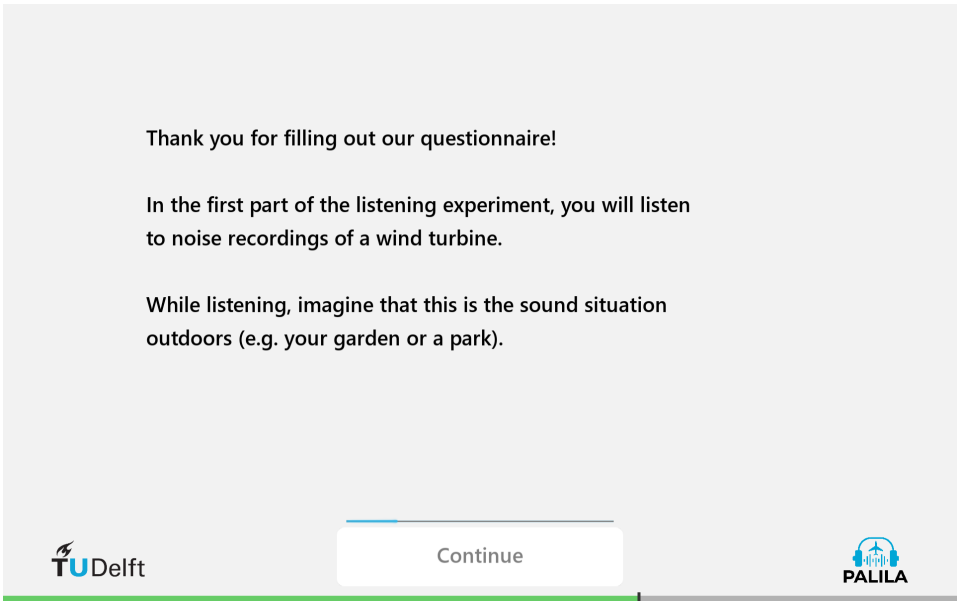


Figure 3.4: Introduction screen of the 'annoyance_1' part of the example experiment.

Because this part requires a single question for all audio samples, we define this in a separate *questions* block, so we don't have to duplicate the question every time (see also subsection 4.1.4). Note the triple brackets around the *question* definition, to signify it is a block inside the *questions* block.

```
45  [[questions]]
46    [[[question annoyance]]]
47      type = Annoyance
```

The *Annoyance* type question results simply in the ICBEN 11-point scale for measuring annoyance (see also subsection 4.3.5):

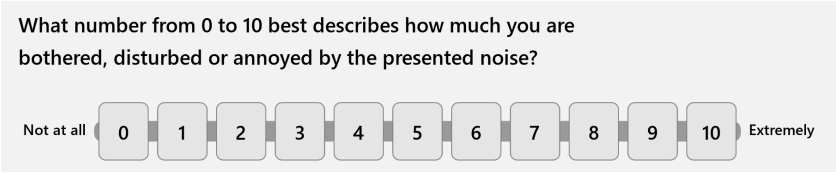


Figure 3.5: The ICBEN 11-point scale for measuring noise annoyance resulting from the *Annoyance* question type.

The 'annoyance' identification in line 49 will be used in the identifier of the answers in the output file. A more thorough explanation of defining the *questions* and their related types is provided in subsection 4.1.4 and section 4.3.

The following block, with identifier 'sample_1', defines one *audio* sub-block. This tells the GUI which sound samples to play, and some related settings. More information can be found in subsection 4.1.5. You have to define one of these blocks for each sample (or pair of samples) you want to include in your experiment.

For this part, we want each sample-question pairing to appear twice, which is achieved with the *repeat* parameter. For this example, we will use the 500 Hz tone audio file every time. You can replace the

filename with any audio file inside the experiment folder. Two more identical *audio* blocks are defined with identifiers 'sample_2' and 'sample_3'.

```
49   [[audio sample_1]]
50     filename = 'tone500Hz.wav'
51     repeat = 2
```

3.3. Testing your configuration

At this point, we have reached the minimum requirements for the GUI to work: (1) defined the required experiment parameters, and (2) added a *part* block with at least one valid *audio* block. To test the configuration more easily, use the *override* parameter. This will unlock most continue buttons so you can scroll through your experiment quickly:

```
override = yes
```

Part intros and *breaks* cannot be skipped with this method, but setting the hold *time* to 0 seconds will instantly unlock the *continue* buttons:

```
time = 0
```

IMPORTANT: Before running your experiment with participants, **reset these lines of code**, so your participants cannot skip parts of your experiment.

3.4. Part 2 - Comparison

As the name of the second experiment block implies, it is an example of including two sound samples in a single *audio* screen. The *part* block is started with the following:

```
62 [part compare]
63   randomise = yes
64
65   [[intro]]
66     ...
```

A 10-second break is added to the middle of this listening block, using a *breaks* sub-block inside the *part* block. The *interval* parameter is set to '-2' to split this part in the middle. A negative number is used so no break screen appears at the end of this *part*. This is further explained in subsection 4.1.2.

```
75   [[breaks]]
76     text = '''This is a short break from listening.
77       Please stay as long as you need.
78       When you continue, imagine that this is the sound situation
79       outdoors (e.g. your garden or a park).'''
80
81     interval = -2
82     time = 10
```

During these comparisons, two questions will be asked to participants. A *questions* sub-block is used once more, since these questions are used for all comparisons. The first question uses the *Multiple-Choice* type, to ask participants which sample they find more annoying. If one sample is more annoying, the *IntegerScale* question will ask by how much, from 1 to 10. The *unlocked by* and *unlock condition* are used, so participants do not have to answer the second question if they find both samples equally annoying.

```
84   [[questions]]
85     [[[question which]]]
86       text = '''Which sample did you experience as more annoying?'''
87       type = MultipleChoice
88
89       choices = 'Left', 'Equally Annoying', 'Right'
90
91     [[[question rating]]]
92       text = '''On this scale from 1 to 10, how much MORE bothered,
     disturbed or annoyed were you by the most annoying sample, compared to
     the less annoying sample?'''
93       type = IntegerScale
94
95       min = 1
96       max = 10
97       left note = 'Almost equal'
98       right note = 'A lot more'
99
100      unlocked by = which
101      unlock condition = Left;Right
```

Adding a second sound sample to an *audio* block is done with the *filename_2* parameter. This is shown in the following example:

```
104  [[audio comparison_1]]
105    filename = 'tone500Hz.wav'
106    filename_2 = 'tone500Hz.wav'
107
108    repeat = 2
109    max replays = 2
```

In this part of the experiment, the *max replays* parameter is set to '2', so participants can listen to each of the samples two times. The *repeat* parameter is again set to '2', so each comparison will be presented twice.

An identical *audio* block named 'comparison_2' is also added to this experiment (see Appendix A).

Ultimately, the *audio* and *questions* sub-blocks will result in four times the following screen:

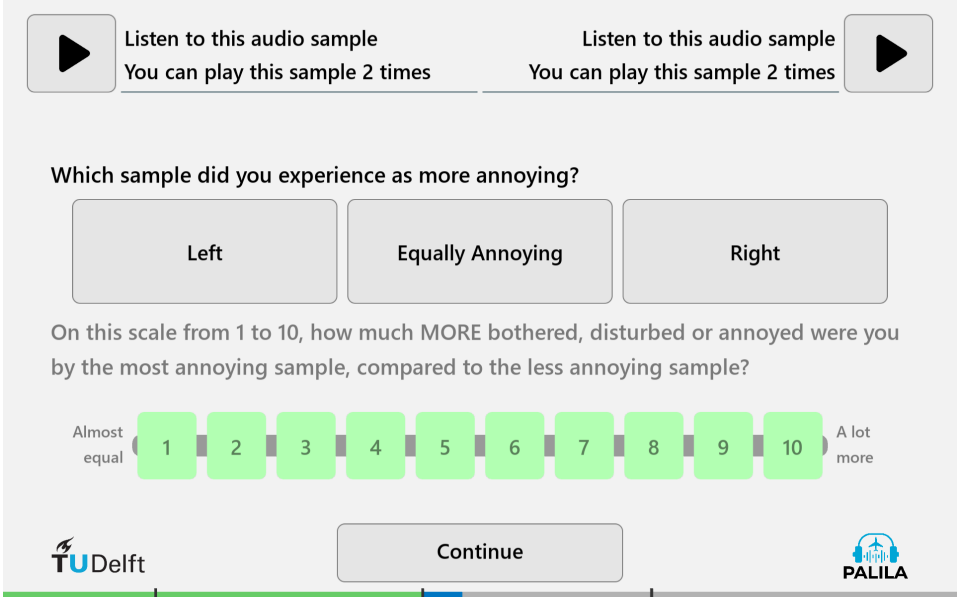


Figure 3.6: The audio screen in the comparison part of the example GUI.

3.5. Part 3 - Annoyance (2)

Lastly, a third *part* with simple annoyance questions is added to the experiment. Since this is the last *part* of the experiment, a small questionnaire is added at the end. For the purpose of this example, we make a copy of the 'annoyance_1' part and rename it to 'annoyance_2':

```
119 [part annoyance_2]
120   ...
```

At the end of this *part* block, a *questionnaire* sub-block is added with two multiple choice questions:

```
148   [[questionnaire]]
149     [[[question wt_opinion_change]]]
150       type = MultipleChoice
151       text = '''Has this experiment changed your attitude
152         towards wind turbines?'''
153       choices = 'Yes, positively', 'No', 'Yes, negatively'
154
155     [[[question wtn_acceptable]]]
156       type = MultipleChoice
157       text = '''Would you find any of the presented sound situations
158         acceptable in your daily life?'''
159       choices = 'None', 'Some', 'Most', 'All'
```

This final questionnaire's questions will appear as Figure 3.7

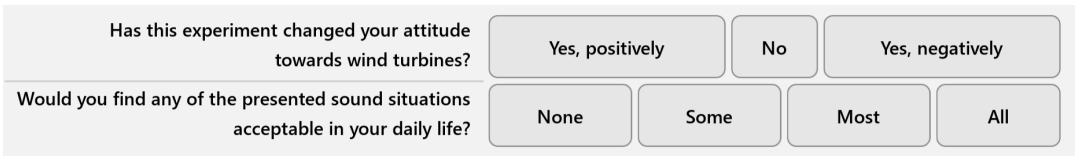


Figure 3.7: Questionnaire questions at the end of the third part of the example GUI configuration.

3.6. Output files

Finally, it's important to discuss what the output will look like. Each participant's individual response will be stored in your experiment directory in the "responses" folder. These output files are assigned a name according to the principle outlined in section 2.5. Inside these files, you will find a table with two rows. The first row contains the full question identifiers, and the second row contains the corresponding responses. The question identifiers are constructed as follows:

<part_name>-<audio_name>(_<repetition_index>)-<question_name>

The repetition index is only included when one audio is repeated during the experiment with the *repeat* parameter (see section 4.1.5). Table 3.1 shows all the IDs resulting from the example configuration file.

NOTE: the IDs Table 3.1 are split into the "<part_name>" and the other parts of the full ID. In the real output files, you will find, for example: *"annoyance_1-sample_1_01-annoyance"*.

Table 3.1: List of all question IDs in the output file of the example experiment.

main-questionnaire	**annoyance_1**	**compare**
age	sample_1_01-annoyance	comparison_1_01-replays-left
gender	sample_1_02-annoyance	comparison_1_01-replays-right
employment	sample_2_01-annoyance	comparison_1_01-which
education	sample_2_02-annoyance	comparison_1_01-rating
how_found	sample_3_01-annoyance	comparison_1_02-replays-left
how_found_other	sample_3_02-annoyance	comparison_1_02-replays-right
affiliation		comparison_1_02-which
hearing_rating	**annoyance_2**	comparison_1_02-rating
accident	sample_1_01-annoyance	comparison_2_01-replays-left
hearing_aid	sample_1_02-annoyance	comparison_2_01-replays-right
wearing_aid	sample_2_01-annoyance	comparison_2_01-which
blast_trauma	sample_2_02-annoyance	comparison_2_01-rating
work_protection	sample_3_01-annoyance	comparison_2_02-replays-left
which_protection	sample_3_02-annoyance	comparison_2_02-replays-right
ear_disease	questionnaire-wt_opinion_change	comparison_2_02-which
which_disease	questionnaire-wtn_acceptable	comparison_2_02-rating
tinnitus		
cold		
cold_magnitude		
feeling		

Running the experiment with multiple participants will result in multiple output files, one per participant. With no randomisation, merging these output files is straightforward. With randomisation, the question identifiers will appear out of order in the output files, with a different order per participant. While this can be used to check for the shuffling, it requires extra attention while merging the output files into one table.

This software includes a script to automatically merge the output files: *MERGE_RESPONSES.bat*. The script will prompt the user to enter the experiment name, i.e. the name of the experiment folder. It will look for all *csv* files in *<experiment name>/responses*, and merge those into one *csv* file table. Each row represents one participant, and each column contains the result per question. The resulting table is stored as *<experiment name>/responses_table.csv*.

This merged file removes the identification of individual participants, and it shuffles the order of the participants in the final output table. The index column is reduced to integers starting from 1. This is the maximum extent of anonymisation provided by this software. Further data processing is at the discretion of researchers using this software.

4

Audio Configuration

This chapter covers all topics related to the listening part of an experiment in this GUI. The sections will cover:

4.1) the *part* block structure, and its parameters and sub-blocks.

4.2) the *question* sub-block, which defines the individual questions.

4.3) the available *question* types available for sound sample testing.

4.1. Part (block)

Required block: defines a part of the experiment with similar audios. Level 1 block

The most important component(s) of the experiment configuration is/are the *part* block(s). Each experiment requires at least one *part*. The *part* blocks provide the experiment structure in both the interface and in the output file. This system allows researchers to include a clear distinction between different types of testing during a single experiment, as illustrated in chapter 3.

The structure of a *part* block should look like the following example, where 'part_name' will be used in the question identifiers in the output file. All possible sub-blocks are further explained in the following sections.

```
[part part_name]
  randomise = <boolean>    # (optional)

  [[intro]]                # (optional)
    ...

  [[breaks]]               # (optional)
    ...

  [[questions]]            # (optional)
    ...

  [[questionnaire]]        # (optional)
    ...


  [[audio audio_name_1]]
    ...

  [[audio audio_name_2]]   # (optional)
    ...

  ...
```

Randomise

Optional parameter: activates shuffling the audios within a part. Boolean value (default = 'no')

Similar to how *part* blocks can be shuffled in the overall experiment, the *audio* sub-blocks can be shuffled within each *part*. With this parameter set to *yes*, each participant will be presented with all *audios* in a different pseudo-random order.

NOTE: the randomisation does not affect the audio and question identifiers in the output files. The same ID will always be assigned to the exact same audio-question pairing.

NOTE: the *randomise* parameter inside a *part* block only shuffle the *audios* within said *part*. The *audios* in other *parts* will not be affected.

4.1.1. Intro (sub-block)

Optional sub-block: defines a custom introduction to the part. Level 2 block

This sub-block can be used to customise the introduction screen of a *part*. The introduction screen always shows up in the experiment. An *intro* sub-block is structured as follows:

```
[[intro]]
  text = <string value>
  time = <float value>
```

text

Required parameter: defines the custom text of the introduction. String value

NOTE: the available space for the intro text is 12 lines high, and approximately 60 characters wide. The recommended limit for this parameter is 720 characters.

time

Required parameter: defines the hold time of this introduction. Float value

The *time* parameter defines in seconds how long the participants will be held in the introduction before they can continue.

4.1.2. Breaks (sub-block)

Optional sub-block: defines mandatory breaks for participants. Level 2 block

To alleviate participant fatigue, you can put mandatory breaks into the experiment. This sub-block defines all breaks within a single *part*. This sub-block will have the following structure:

```
[[breaks]]
  text = <string value>
  time = <float value>
  interval = <integer value>
```

text

Optional parameter: defines a custom text in the breaks. String value

NOTE: the available space for the break text is 12 lines high, and approximately 60 characters wide. The recommended limit for this parameter is 720 characters.

time

Required parameter: defines the hold time of this introduction. Float value

The *time* parameter defines in seconds how long the participants will be held in a break before they can continue.

interval

Required parameter: defines the interval between breaks. Integer value

There are three ways to define the interval:

1. Zero: a break is placed at the end of the *part*, after the questionnaire.
2. Positive n: a break is placed every n *audios*, and at the end of the *part*.
3. Negative n: a break is placed every n *audios*, but not at the end of the *part*.

NOTE: the behaviour or "*interval = 0*" will be changed in a future version. The break will appear before the questionnaire instead of after.

NOTE: in a future version, a separate *text* can be set for the *break* at the end of a *part*.

4.1.3. Questionnaire (sub-block)

Optional sub-block: defines a questionnaire at the end of the part. Level 2 block

Use of this sub-block enables an extra questionnaire at the end of an experiment part. This can be used to ask supporting questions about the *audios*, or as a closing questionnaire. This sub-block is defined with the exact syntax as the main questionnaire at the start of the experiment. See chapter 5 for more. This sub-block should be structure as follows:

```
[[questionnaire]]
  [[[question question_name]]]
    ...
  ...
```

NOTE: the *default* parameter is not available for part questionnaires.

IMPORTANT: *questions* in the *part* questionnaires are defined as **LEVEL 3 BLOCKS**, and thus require triple square brackets.

4.1.4. Questions (sub-block)

Optional sub-block: defines common questions for all audios in a part. Level 2 block

This sub-block allows you to define a common pair of questions for all *audios* within a *part*. Simply define one or two *question* sub-blocks within this sub-block, according to the syntax described in section 4.2. This sub-block will be structured as follows:

```
[[questionnaire]]
  [[[question question_name_1]]]
    ...
  [[[question question_name_2]]]  # (optional)
    ...
```

IMPORTANT: this sub-block overwrites all *questions* defined within the *audio* sub-blocks!

4.1.5. Audio (sub-block)

Required sub-block: defines one audio sample or pairing of two samples. Level 2 block

The *audio* sub-block defines one full screen with one or two sound samples and one or two related questions.

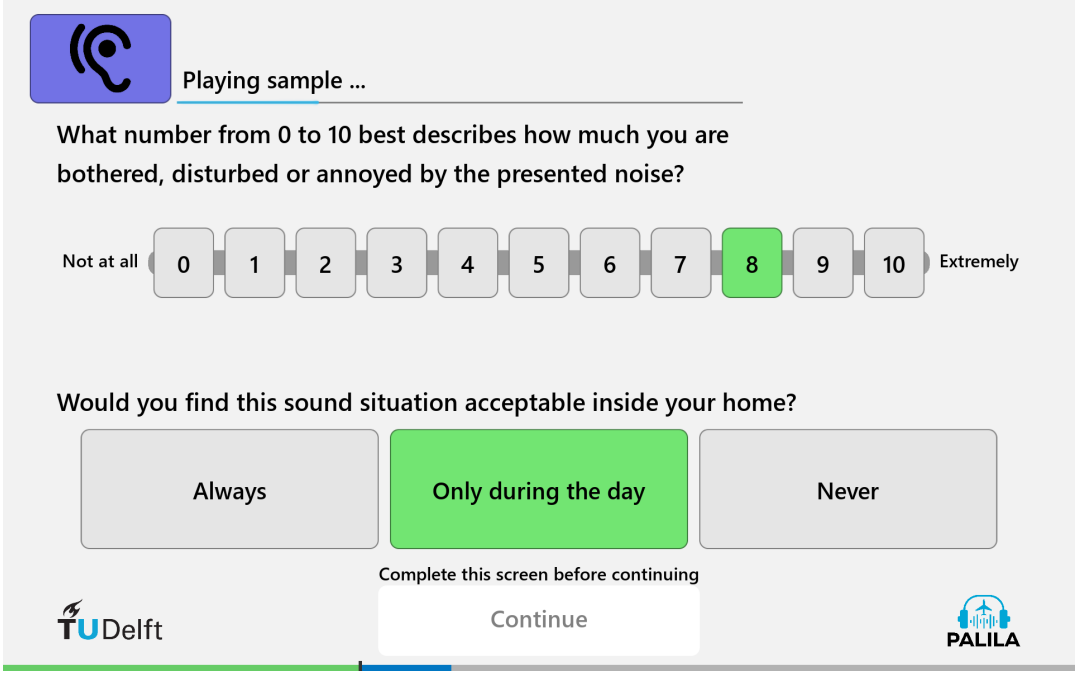


Figure 4.1: Example of the *Audio* screen in the interface.

An *audio* sub-block should have the following structure, where 'audio_name' will be used in the question identifiers in the output file. One *audio* block can contain two *question* sub-blocks, defined according to the syntax of section 4.2. These *question* blocks are overwritten by a *questions* block in the overarching *part* block (see subsection 4.1.4).

```
[[audio audio_name]]
  filename = <string value>
  filename_2 = <string value>     # (optional)

  repeat = <integer value>        # (optional)
  max replays = <integer value>   # (optional)

  [[[question question_name_1]]]  # (optional)
    ...
  [[[question question_name_2]]]  # (optional)
    ...
```

filename

Required parameter: defines the file name of the sound sample. String value

This parameter defines the file name of a sound sample in your experiment folder. For example, with a PALILA file named 'experiment.palila', the GUI attempts to find a sound sample with filename '*sound_sample.wav*' at *path_to_gui/experiment/sound_sample.wav*.

NOTE: this parameter is used as a path relative to the experiment folder. Audio files can therefore be stored in subfolders of the experiment folder.

IMPORTANT: the *filename* should always include the file extension (i.e. '.wav', '.mp3', etc.)!

filename_2

Optional parameter: defines the file name of a second sound sample. String value

Similar to *filename*, this parameter will add a second sound sample on the same screen. This can be used, for example, for comparisons. Participants are mandated to listen to both sound samples. The two sound samples are blocked from playing simultaneously. The sound sample bar on top of the *audio* screen will look like:

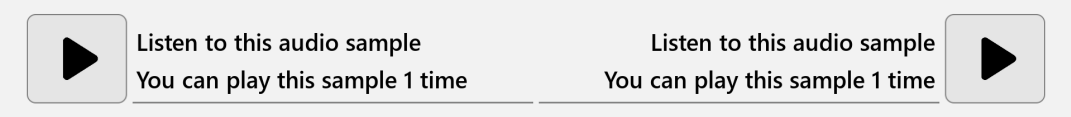


Figure 4.2: Example of two side-by-side audio samples using *filename_2*.

NOTE: the audio from *filename_2* will always appear on the right. This can be exploited similarly to the example in section 3.4.

NOTE: in a future version, an option will be added to randomly shuffle the left and right audio, per participant.

repeat

Optional parameter: number of repetitions of an audio block. Integer value (default = 1)

With this parameter, the *audio* screen from one *audio* sub-block is repeated multiple times, without the need for duplicated definitions. In the output file, these repetitions will result in separate outputs, distinguished by an additional *repetition index* (see section 3.6).

max replays

Optional parameter: number of times a sound sample can be played. Integer value (default = 1)

This parameter defines the number of times a sound sample in one *audio* screen can be played by participants. In the example configuration in Figure 3.6 and Figure 4.2, *max replays* is set to '2', meaning participants can listen to each sample twice. In case of two sound samples on one screen, this parameter applies to both samples.

NOTE: in a future version, an option will be added to allow pausing a sound sample after it has been fully played once.

4.2. Question (sub-block)

Optional sub-block: defines a single question. Level 3 block

A *Question* sub-block defines one question on the *audio* screen in the interface. It should be defined inside an *audio* or *questions* sub-block. The code for the *question* sub-block has the following structure. Depending on the *type* of question, more parameters may be required. These extra parameters are further explained in section 4.3.

```
[[[question question_name]]]
  type = <string value>
  text = <string value>
  ...

  unlocked by = <string value>       # (optional)
  unlock condition = <string value>  # (optional)
```

text

Required parameter: defines the question to be asked to participants. String value

NOTE: the available space for the question text is two lines high, and approximately 80 characters wide. The recommended limit for this parameter is 160 characters.

type

Required parameter: defines the type of this question. String value

For each question, the type has to be defined. The choices for question types on an *audio* screen are listed in section 4.3. The value entered in this parameter should match exactly with the names listed below, including capitalisation. Depending on the set type, more parameters will be required, which will be listed below.

unlocked by

Optional parameter: identifier of a question which unlocks this question. String value

Together with the *unlock condition*, this parameter allows for a question to only require an answer if another question is answered in a certain way. This parameter defines the identifier of the other question which unlocks this one. This identifier is 'question_name' in the example at the top of this section. An example of this unlocking system is shown in section 3.4.

NOTE: The unlock system will only work within the scope of one *audio* screen. Therefore, this system will not work across different *audio* blocks.

unlock condition

Optional parameter: answers which will unlock this question. String value

This parameter is required when *unlocked by* is defined. This parameter will list all answers of the other question that unlock this question. For one *unlock condition* value, any form of the *string* value syntax can be used (see section 2.4. Multiple values requires a very specific formatting. No quotation marks should be used, and the values should be split by semicolons (;). This semicolon cannot be followed by a space!

In case a question remains locked (i.e. no answer required), the output value will be '*na*'.

NOTE: for multiple *unlock conditions*, the semicolon will be changed to a normal comma to match the usual *list of strings* behaviour. This would also remove any restrictions regarding the spaces and quotes.

4.3. Question Types

This section describes all available question *types* and the parameters required to define those. This section is related to the *question* sub-block (see section 4.2).

NOTE: the *FreeText* question type from the questionnaire will be made available for audios in a future version.

4.3.1. Text

This question type only shows text. Unlike the normal *question text*, it can show up to five lines. The configuration of this question should look as follows:

```
[[[question text_example]]]
    type = Text
    text = <string value>
    ...
```

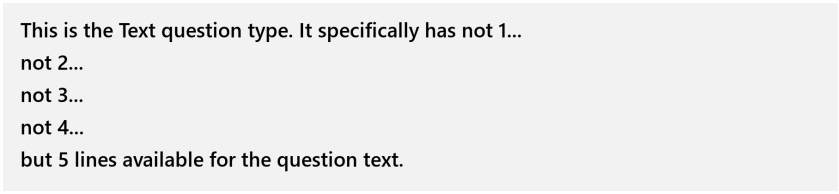


Figure 4.3: Example of the *Text* question type.

NOTE: the available space for the text is 5 lines high, and approximately 80 characters wide. The recommended limit for this parameter is 160 characters.

4.3.2. MultipleChoice

This question type provides a button-based multiple-choice answering system. This question type can force users to select a single answer, or it can allow for multiple answers. The structure of this question should look like:

```
[[[question multiplechoice_example]]]
    type = MultipleChoice
    text = <string value>

    choices = <list of strings>
    multi = <boolean>            # (optional)
    ...
```

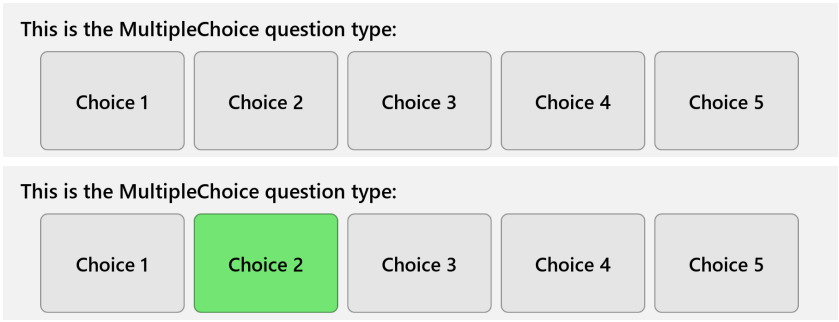


Figure 4.4: Example of the *MultipleChoice* question type. Before (top) and after (bottom) answering the question.

choices

Required parameter: list of the possible answers. List of string values

NOTE: The recommended limit of *choices* is 5, but the actual maximum depends on the length of the individual answer options. For a multiple choice question with more answer options, use the *Spinner* type (see subsection 4.3.3)

multi

Optional parameter: allows for multiple answers. Boolean value (default = 'no')

When set to 'yes', this parameter allows participants to select multiple answers on the *MultipleChoice* question. In the results file, the answers will be separated by a semicolon (;).

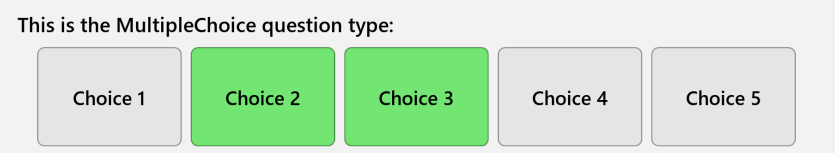


Figure 4.5: Example of the *MultipleChoice* question type with multiple answers selected.

4.3.3. Spinner

The *Spinner* is a variant of the *MultipleChoice* question, with a dropdown menu instead of buttons. Unlike the *MultipleChoice* question, the *multi* option is not available. The code for this question type requires the following structure:

```
[[[question spinner_example]]]
    type = Spinner
    text = <string value>

    choices = <list of string values>
    ...
```

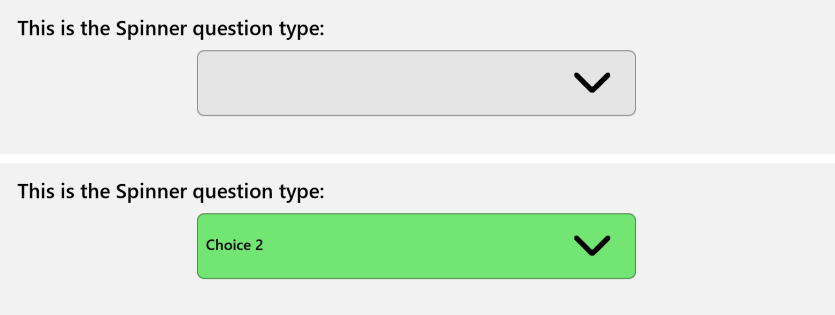


Figure 4.6: Example of the *Spinner* question type. Before (top) and after (bottom) answering the question.

choices

Required parameter: list of the possible answers. List of string values

See the *MultipleChoice* question type (subsection 4.3.2).

4.3.4. IntegerScale

This question type gives participants a scale with integer values to answer. It is highly customisable. The GUI will automatically scale the answer buttons depending on the number of them. The step between numbers is always 1, thus the number of buttons is always $max - min + 1$. Text can be defined on the left and right side of the scale, to give participants an indication of the meaning of the extreme values.

```
[[[question integerscale_example]]]
    type = IntegerScale
    text = <string value>

    min = <integer value>
    max = <integer value>

    left note = <string value>   # (optional)
    right note = <string value>  # (optional)
    ...
```

NOTE: since this question type is derived from *MultipleChoice*, the variable *multi* is available for the *IntegerScale* and *Annoyance* question types (see section 4.3.2).

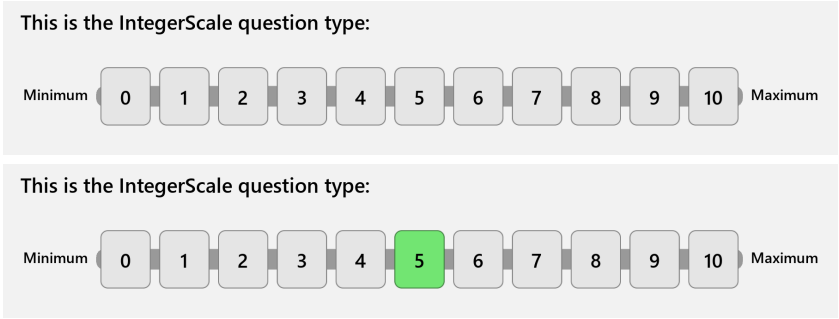


Figure 4.7: Example of the *IntegerScale* question type. Before (top) and after (bottom) answering the question.

min, max

Required parameters: define the range of the scale. Integer values

These parameters define the minimum and maximum values on the scale, respectively. The recommended maximum range ($max - min$) is 10 with a *left note* or a *right note*, and 12 for a scale without.

left note, right note

Optional parameters: define text on either side of the scale. String values

On both ends of the scale, a *note* can be placed. Either of these notes can be left empty, but the scale will always be adjusted to fit both notes.

NOTE: These notes can fit approximately 10 characters on one line, with a recommended maximum of two lines (i.e. 20 characters total).

4.3.5. Annoyance

A derivative of the *IntegerScale* with the standardised 11-point annoyance scale. The example below shows this configuration. This question type only requires the *type* parameter:

```
[[[question annoyance_example]]]
    type = Annoyance
    text = <string value>  # (optional)
    multi = <boolean>      # (optional)
    ...
```

IMPORTANT: The *text* parameter is strictly optional for this question type and will always overwrite the default text.

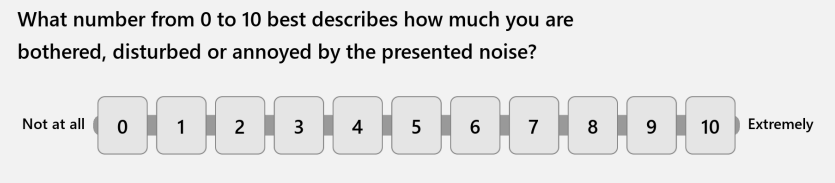


Figure 4.8: Example of the *Annoyance* question type.

4.3.6. Slider

NOTE: the look and functionality of this question type will be revised in a future version, to allow more configurations.

This is a variant of the *IntegerScale* question type, but with a slider instead of buttons. This type allows for a more granular numerical answer. The code for this question type should contain the following:

```
[[[question slider_example]]]
    type = Slider
    text = <string value>

    min = <float value>
    max = <float value>
    step = <float value>

    left note = <string value>   # (optional)
    right note = <string value>  # (optional)

    multi = <boolean>            # (optional)
    ...
```

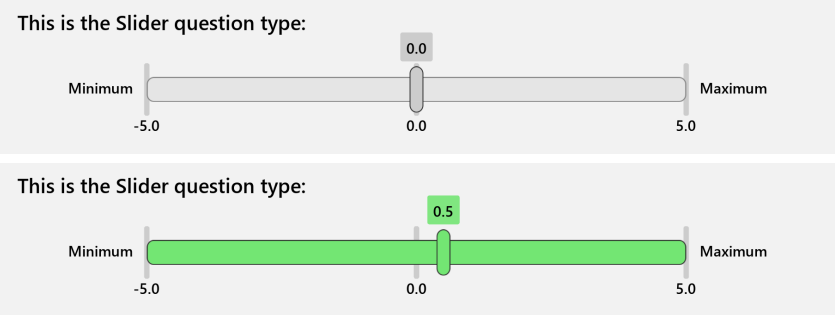


Figure 4.9: Example of the *Slider* question type. Before (top) and after (bottom) answering the question.

min, max

Required parameters: define the range of the scale. Integer values

See the *IntegerScale* question type (subsection 4.3.4)

left note, right note

Optional parameters: define text on either side of the scale. String values

See the *IntegerScale* question type (subsection 4.3.4)

step

Required parameter: defines the delta between possible answers. Float value

With the slider, it is possible to allow for smaller steps than 1 between possible numbers. E.g. if the step is set to 0.1, it is possible to answer 6.9, 4.2, etc. With a step of 0.5, only 6.5 and 4.0 would be possible.

5

Questionnaire Configuration

This chapter covers the configuration of *questionnaires* in the PALILA GUI. These sections apply to both the *questionnaire* at the start of an experiment, and the ones in the experiment *parts*. Each section covers:

5.1) The overall *questionnaire* block and the *default* questionnaire.

5.2) The different types of *questions* available in the *questionnaires*.

NOTE: the option for a separate questionnaire at the end of the experiment (independent from the *parts*) will be added in a future version.

5.1. Questionnaire Block

Optional block: defines the main questionnaire at the start of an experiment. Level 1 block

To define the questionnaire at the start of the experiment (i.e. the '*main questionnaire*'), you have to use the *questionnaire* block on the base level of the PALILA file. If this block is not defined, or left empty, the experiment will start with the first *part* instead. The block is defined with the following line of code, with single brackets (level 1 block) and the keyword 'questionnaire':

```
[questionnaire]
  default = <boolean>          # (optional)
  manual split = <boolean>     # (optional)

  [[question question_name]]   # (optional)
    ...
  ...
```

All questions in this *main questionnaire* are assigned question identifiers that start with 'main-questionnaire'. *Questionnaire* questions in a *part* are assigned identifiers starting with the name of that *part*.

NOTE: the *questionnaires* in *part* blocks have to be defined on one extra block level. I.e. the *questionnaire* block should be level 2, and the *question* sub-blocks should be level 3.

manual split

Optional parameter: turn on manual question distribution. Boolean value (default = 'no')

This setting turns on the system for manually distributing questions over multiple questionnaire screens. This manual split should adhere to the following:

1. Each question in the questionnaire requires the parameter *manual screen* (see section 5.1.1).
2. A maximum of 7 questions can be assigned to each screen.

default

Optional parameter: turns on the default questions. Boolean value (default = 'no')

The default questionnaire, obtained by setting '*default* = *yes*' in the main *questionnaire* block, contains three screens with seven questions. The first screen contains questions to collect basic population statistics, which are listed in Table 5.1. Table 5.2 lists all questions and answer options from the second and third questionnaire screens. These questions are aimed at gauging the hearing health of participants.

NOTE: the default questionnaire is defined in the file: *GUI/default_questionnaire.palila*. This file can be modified using the syntax from this chapter. The questions in this PALILA file are defined as level 1 blocks, since the file acts as the *questionnaire* block.

NOTE: the options of *how_found* will be changed in a future version to be more generally applicable to other organisations.

NOTE: in a future version, an option will be added to change the name of the organisation in the *affiliation question*.

Table 5.1: Population statistics questions on the first screen of the default questionnaire.

Identifier	Type	Text	Answer Option(s)
age	FreeNumber	Age:	-
gender	MultipleChoice	Gender identity:	Man Woman Other Prefer not to say
employment	Spinner	Current employment status:	Prefer not to say Employed for wages Self-employed Homemaker Student Retired Out of work Unable to work
education	Spinner	Highest completed level of school/ education:	Prefer not to say Elementary (Kindergarten through primary school) Some High School (No Diploma) High School Graduate Some College, but No Degree Bachelor's Degree Master's Degree Doctorate
how_found	Spinner	How did you find out about this experiment?	Posters Information screens Social media TU Delft website Brightspace Email Other
how_found_other	FreeText	Which other way made you find this experiment?	-
affiliation	Spinner	Your current, most direct affiliation with TU Delft:	Prefer not to say Student / employee at TU Delft Student / employee at a partner university Employee at a partner company None of the above

Table 5.2: Hearing health questions on the second and third screens of the default questionnaire.

Identifier	Type	Text	Answer Option(s)
hearing_rating	MultipleChoice	How good do you think your hearing is?	Excellent Very Good Good Fair Poor
accident	MutlipleChoice	Have you had an accident involving your head that affected your hearing?	Yes No
hearing_aid	MutlipleChoice	Do you normally use a hearing aid?	Yes No
wearing_aid	MutlipleChoice	Are you currently wearing a hearing aid?	Yes No
blast_trauma	MutlipleChoice	Have you suffered a shooting or explosion injury (blast trauma)?	Yes No
work_protection	MutlipleChoice	Do you have to wear hearing protection at work, or did you have to in the past?	Yes No
which_protection	FreeText	What noise do/did you have to protect yourself from?	-
ear_disease	MultipleChoice	Have you suffered, or are you currently suffering from any ear disease?	Yes No
which_disease	FreeText	If yes, what kind of ear disease?	-
tinnitus	MultipleChoice	Do you suffer from ringing in the ears (tinnitus)?	Yes No
cold	MultipleChoice	Do you currently have a cold?	Yes No
cold_magnitude	MultipleChoice	How bad is your cold? From 1 (very weak) to 5 (very strong)	1 2 3 4 5
feeling	MultipleChoice	Are you currently feeling well?	Yes No
tired	MultipleChoice	Are you currently feeling very tired?	Yes No

5.1.1. Question (Sub-block)

Optional sub-block: defines a single question. Level 2 block

This sub-block will define all parameters of one question. The syntax and behaviour are similar to the *audio* questions in section 4.2. Below is the structure of the code for a *questionnaire* question:

```
[[[question question_name]]]
  type = <string value>
  text = <string value>
  manual screen = <integer>          # (optional)
  ...

  unlocked by = <string value>       # (optional)
  unlock condition = <string value>  # (optional)
```

text

Required parameter: defines the question to be asked to participants. String value

NOTE: the available space for the question text is two lines high, and approximately 45 characters wide. The recommended limit for this parameter is 90 characters.

type

Required parameter: defines the type of this question. String value

For each question, the type has to be defined. The choices for question types in the *questionnaire* are listed in section 5.2.

manual screen

Optional parameter: sets the screen number for this question to appear on. Integer value

When *manual split* is turned on (see section 5.1), this parameter is required to define which screen the question will appear on. The screens are sorted from lowest screen number to highest. These screen numbers do not need to be consecutive.

unlocked by

Optional parameter: identifier of a question which unlocks this question. String value

Together with the *unlock condition*, this parameter allows for a question to only require an answer if another question is answered in a certain way. This parameter defines the identifier of the other question which unlocks this one. This identifier is 'question_name' in the example at the top of this section.

The unlock system only works within the scope of one *questionnaire* screen. Therefore, a question on one screen cannot unlock a question on a different screen. An example of this unlocking system is shown in section 3.1.

NOTE: The unlock system will be expanded in a future version of the GUI, to allow for unlocking questions across questionnaire screens.

unlock condition (parameter)

Optional parameter: answers which will unlock this question. String value

This parameter is required when *unlocked by* is defined. This parameter will list all answers of the other question that unlock this question. For more information, see section 4.2.

5.2. Question Types

This section lists the *question* types available in the *questionnaire* blocks. Some of these require extra parameters over the ones listed in subsection 5.1.1.

5.2.1. FreeText

This question type allows users to freely enter any answer in a text box with the keyboard. Participants have to enter at least one character in the box for the question to register an answer. This question type requires the following *question* sub-block:

```
[[[question question_name]]]
  type = FreeText
  text = <string value>
  ...
```

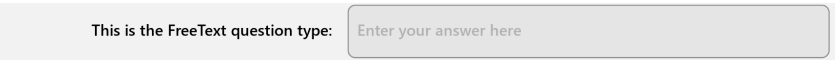


Figure 5.1: Example of the *FreeText* question type.

5.2.2. FreeNumber

This question is a derivative of the *FreeText* question type. Participants can enter any number in this text box, as long as it is an integer. A numpad shows on the screen when the box is active, to allow for touchscreen use. This question type requires the following *question* sub-block:

```
[[[question question_name]]]
  type = FreeNumber
  text = <string value>
  ...
```

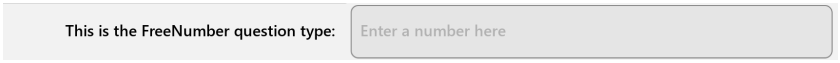


Figure 5.2: Example of the *FreeNumber* question type.

Figure 5.3: The numpad of the *FreeNumber* question type.

5.2.3. MultipleChoice

This question type provides a button-based multiple-choice answering system. This question type can force users to select a single answer, or it can allow for multiple answers. This question type requires the following *question* sub-block:

```
[[[question question_name]]]
  type = MultipleChoice
  text = <string value>

  choices = <list of string values>
  multi = <boolean>                  # (optional)
  ...
```

Figure 5.4: Example of the *MultipleChoice* question type (*questionnaire*).

choices

Required parameter: list of the possible answers. List of string values

NOTE: The recommended limit of *choices* is 5, but the actual maximum depends on the length of the individual answer options. For a multiple choice question with more answer options, use the *Spinner* type (see subsection 4.3.3)

multi

Optional parameter: allows for multiple answers. Boolean value (default = 'no')

When set to 'yes', this parameter allows participants to select multiple answers on the *MultipleChoice* question. In the results file, the answers will be separated by a semicolon (;). The example above shows multiple selected answers with this parameter turned on.

5.2.4. Spinner

The *Spinner* is a variant of the *MultipleChoice* question, with a dropdown menu instead of buttons. Unlike the *MultipleChoice* question, participants can only select one answer. This question type requires the following *question* sub-block:

```
[[[question question_name]]]
  type = Spinner
  text = <string value>

  choices = <list of string values>
  ...
```

Figure 5.5: Example of the *Spinner* question type (*questionnaire*).

choices

Required parameter: list of the possible answers. List of string values

See the *MultipleChoice* question type (subsection 5.2.3).

A

Example PALILA file from chapter 3

```
## This is your experiment input file. It is set up with a basic structure
    for a PALILA GUI experiment.
override = yes

pid mode = auto
randomise = no
demo = no

[questionnaire]
  default = yes

  [[question heard_before]]
    text = 'Have you ever heard wind turbine noise before?'
    type = MultipleChoice
    choices = 'Yes', 'No', 'Not sure'

  [[question live_near]]
    text = 'Can you see a wind turbine from your home?'
    type = MultipleChoice
    choices = 'Yes', 'No'

  [[question live_distance]]
    text = '''If yes, what is the approximate distance from
      your home to this/these wind turbines?'''
    type = MultipleChoice
    choices = '< 500 m', '500 - 1000 m', '1 - 2 km', '> 2 km'

    unlocked by = 'live_near'
    unlock condition = 'Yes'


[part annoyance_1]
  randomise = yes

  [[intro]]
    text = '''Thank you for filling out our questionnaire!

      In the first part of the listening experiment, you will listen
      to noise recordings of a wind turbine.

      While listening, imagine that this is the sound situation
      outdoors (e.g. your garden or a park).'''

    time = 0
```

```
44
45    [[questions]]
46      [[[question annoyance]]]
47        type = Annoyance
48
49    [[audio sample_1]]
50      filename = 'tone500Hz.wav'
51      repeat = 2
52
53    [[audio sample_2]]
54      filename = 'tone500Hz.wav'
55      repeat = 2
56
57    [[audio sample_3]]
58      filename = 'tone500Hz.wav'
59      repeat = 2
60
61
62  [part compare]
63    randomise = yes
64
65    [[intro]]
66      text = '''In the second part of the listening experiment, you will
       listen
67        to noise recordings of a wind turbine. Each screen will contain two
68        sound samples, which you will compare.
69
70        While listening, imagine that this is the sound situation
71        outdoors (e.g. your garden or a park).'''
72
73      time = 0
74
75    [[breaks]]
76      text = '''This is a short break from listening.
77        Please stay as long as you need.
78        When you continue, imagine that this is the sound situation
79        outdoors (e.g. your garden or a park).'''
80
81      interval = -2
82      time = 0
83
84    [[questions]]
85      [[[question which]]]
86        text = '''Which sample did you experience as more annoying?'''
87        type = MultipleChoice
88
89        choices = 'Left', 'Equally Annoying', 'Right'
90
91      [[[question rating]]]
92        text = '''On this scale from 1 to 10, how much MORE bothered,
       disturbed or annoyed were you by the most annoying sample, compared to
       the less annoying sample?'''
93        type = IntegerScale
94
95        min = 1
96        max = 10
```

```
      left note = 'Almost equal'
      right note = 'A lot more'

      unlocked by = which
      unlock condition = Left;Right


  [[audio comparison_1]]
    filename = 'tone500Hz.wav'
    filename_2 = 'tone500Hz.wav'

    repeat = 2
    max replays = 2

  [[audio comparison_2]]
    filename = 'tone500Hz.wav'
    filename_2 = 'tone500Hz.wav'

    repeat = 2
    max replays = 2


[part annoyance_2]
  randomise = yes

  [[intro]]
    text = '''
      In this final part of the listening experiment, you will listen
      to noise recordings of a wind turbine, once more.

      While listening, imagine that this is the sound situation
      outdoors (e.g. your garden or a park).'''

    time = 0

  [[questions]]
    [[[question annoyance]]]
      type = Annoyance

  [[audio sample_1]]
    filename = 'tone500Hz.wav'
    repeat = 2

  [[audio sample_2]]
    filename = 'tone500Hz.wav'
    repeat = 2

  [[audio sample_3]]
    filename = 'tone500Hz.wav'
    repeat = 2

  [[questionnaire]]
    [[[question wt_opinion_change]]]
      type = MultipleChoice
      text = '''Has this experiment changed your attitude
        towards wind turbines?'''
```

```
153       choices = 'Yes, positively', 'No', 'Yes, negatively'
154
155     [[[question wtn_acceptable]]]
156       type = MultipleChoice
157       text = '''Would you find any of the presented sound situations
158         acceptable in your daily life?'''
159       choices = 'None', 'Some', 'Most', 'All'
```