L'ART Research Client

Release 0.4.1

Florian Breit, Marco Tamburelli

Apr 18, 2023

CONTENTS

The L'ART Research Client is a freely available open-source app that aids researchers in the collection, storage and transfer of data for research in bilingualism and language attitudes, especially in cases of bilinguals who speak a majority language and a regional / minority / minoritized language. This documentation is for version 0.4.1.

Features

- **Fieldwork-ready** — use the included versions of the LSBQe, AToL, MGT, and digital informed consent forms, or make your own versions.
- **Reproducible** — make your research reproducible by simply sharing version information.
- **Extensible** — contribute with JSON, Python, HTML and JavaScript or import the backend in your own Python project.
- **Free and open** — dual-licensed under the AGPLv3 and the EUPL.
- **Cross-platform** — runs on Windows, MacOS, and Linux.

Citing

If you use the L'ART Research Client (or parts of it), please cite the following document:

> Breit, F., Tamburelli, M., Gruffydd, I. and Brasca, L. (2023). *The L'ART Research Client: A digital toolkit for bilingualism and language attitude research* [Software, version 0.4.1]. Bangor University.

CHAPTER

ONE

ABOUT THE RESEARCH CLIENT

1.1 Introduction

The L'ART Research Client is a freely available open-source app to aid researchers in the collection, storage and transfer of data for research in bilingualism and language attitudes, with a particular focus on bilingual populations who speak a majority language and a regional / minority / minoritized language. The app aims to make research in bilingualism easier, more comparable and reproduceable. For a detailed discussion of the specific methodological choices, see [Breit-Tamburelli-EtAl-2023].

1.2 What the L'ART Reserch Client can do

The current version (L'ART Research Client 0.4.1) implements four tools (for a detailed discussion of methodological adaptations, please see Breit et al., 2023).

- **Participant consent**: A digital informed consent process, including participant information sheets & consent forms.
- **LSBQe**: A digital adaptation of the **Language and Social Background Questionnaire**, or LSBQ [Anderson-Mak-EtAl-2018], which we term the LSBQe ("e" for electronic).
- **AToL**: A digital implementation of the **Attitudes towards Languages Questionnaire** or AToL [Schoel-Roessel-EtAl-2013].
- **MGT** and **VGT**: A digital tool for measuring language attitudes via the speaker evaluation paradigm. This tool enables users to run several evaluations of audio guises such as the **Matched Guise Technique** [Lambert-Hodsgon-EtAl-1960] and the **Verbal Guise test** (e.g., [Markel-EtAl-1967]). Due to its flexibility as either MGT or VGT, we named this tool **'Audio Guise Test'**, or **AGT** for short.
- A simple memory game which can be employed as a general **distractor** in a series of tasks.

The main functionality of the L'ART Research Client resides in its format as a stand-alone app that can run on a large variety of desktop and laptop computers without the need for internet connectivity. This makes it highly usable both in lab environments and in the field, for example when collecting data in remote areas with inconsistent internet access.

The L'ART Research Client has been designed in such a way that it can be easily extended by researchers (or research groups) with just a basic knowledge of Python, JavaScript and HTML needed to implement additional tasks (see the *Developer Guide* for more info). Translating an existing task for a new language or language pair is even easier and can be done by just editing a simple JSON file in a text editor (see tutorials/translating-tasks).

1.3 Reasons to use the Research Client

- **Less work for the researcher:** With research tasks pre-implemented, preparation for a new study only involves translation/localisation of the interface where a suitable one is not yet available for the target population. There is also no need to manage forms and manually enter data after collecting responses.
- **Enhanced consistency and comparability within and across studies:** The translation/localisation of tasks is the only thing that varies within tasks. The presentation, data types and validation, coding, and output format stay constant across different use instances, whether as part of the same study or across different studies and research teams.
- **Improved transparency and reproducibility:** Because the entire source code for the L'ART Research Client is publicly available and version-controlled, it's easy to reference the specific version and task that was used, which allows other researchers to easily view and reconstruct the tasks exactly as they were administered at the time the research was carried out.

For detailed examples and more concrete illustrations of these advantages, see [Breit-Tamburelli-EtAl-2023].

1.4 Citing the Research Client

Breit, F., Tamburelli, M., Gruffydd, I. and Brasca, L. (2023). *The L'ART Research Client: A digital toolkit for bilingualism and language attitude research* [Software, version 0.4.1]. Bangor University.

1.5 Licensing

The L'ART Research Client and all the tools implemented within it are free and open source. The app is dual licensed under the terms of the Affero General Public License (the AGPL) and the European Union Public License (the EUPL). Dual licensing means that you are free to choose under which of the two license's terms you want to use it.

Both licenses allow you to:

- Use the app and its functionality freely (as in freedom) and for free (as in free beer) in your work, whether commercial or non-commercial.
- Modify or otherwise make adaptations to the app and its source code, as long as you yourself make those changes available to others under the same license terms (or the terms of another compatible license where this is expressly permitted by the AGPL or EUPL).
- Add yourself to the credits/copyright notice when you modify the software, as long as you do not remove, materially change, or misrepresent in any way the copyright and author attribution notes as they appear in the app, its source code, documentation, distributions (e.g. installers), etc.

Naturally, if you intend on modifying and/or improving the Research Client, we would appreciate it if you would share those developments with us so we can incorporate any improvements and enhancements into the official version of the app.

Where possible we would also strongly encourage you to retain the dual licensing model, as we believe this ensures maximal adoptability and reusability across a large variety of potential users in different parts of the world.

1.6 Contributors

The L'ART Research Client core developers are Florian Breit (Lead) and Marco Tamburelli.

We would like to thank the following for contributing (in alphabetical order):

- Chloe Cheung (Documentation)
- Lissander Brasca (Translation, Documentation)
- Ianto Gruffydd (User testing, Translation, Documentation)
- Athanasia Papastergiou (Translation)

1.7 Acknowledgements

The L'ART Research Client was developed by the Language Attitudes Research Team (GitHub) in the School of Arts, Culture and Language at Bangor University. Development of the app was supported by the Economic and Social Research Council [grant number ES/V016377/1].

CHAPTER

TWO

USER GUIDE

The L'ART Research Client User Guide covers everything you need to know to install the Research Client, set it up and begin data collection.

For users who wish to customise any of the tasks or add new translations, there are *Quick Tutorials* to help you through the process. For developers who may want to contrribute further developments to the Research Client or build additional tasks, a technical documentation can be found in the *Developer Guide*.

2.1 Compatibility and Requirements

2.1.1 Operating systems

The L'ART Research Client has been developed and tested primarily on Windows 10 and 11 (64-bit Intel and AMD architectures).

The installer and app will probably also work on 64-bit versions of Windows 8, but we do not officially support this. The app is in principle compatible with current versions of MacOS, Linux, and BSD, but this currently requires installation/building from source (see Developer's Guide).

We have plans to officially support these platforms with the version 1.0 stable release in the near future.

2.1.2 System requirements

Minimal requirements:

- 64-bit Intel, AMD, or otherwise x64 compatible processor, 2.4GHz.
- 65MB free disk space.
- 4GB RAM.

Recommended requirements:

- 64-bit Intel or AMD processor, 3.0GHz dual-core.
- 200MB free disk space.
- 8GB RAM.

2.1.3 Required Software

You must have a current version of the Google Chrome browser installed. The L'ART Research Client relies on this to provide its User Interface.

2.1.4 Tested System Configurations

We have successfully tested the app on the following systems:

Manufacturer	Model	Processor	RAM	Operating System
Dell	Inspiron 7620 2-in-1	Intel i7-1260P	16GB	Windows 11 64-bit
Dell	Latitude 5520	Intel i7-1165G7	16GB	Windows 11 64-bit
Dell	Latitude 5520	Intel i7-1165G7	16GB	Windows 10 64-bit
Dell	XPS 9520	Intel i7-9750H	16GB	Windows 10 64-bit
Lenovo	ThinkPad		8GB	Windows 10 64-bit

2.2 Installation

2.2.1 Installing on Windows 10/11

1. Download the official Windows installer for the L'ART Research Client on Windows.

   You can find the latest release (as well as earlier versions) at github.com/lart-bangor/research-client/releases.

2. Once downloaded to your device, open the *Downloads* dialogue in the browser and click *Open file*. Alternatively, navigate to your `Downloads` folder in File Explorer and double click on the installer file.

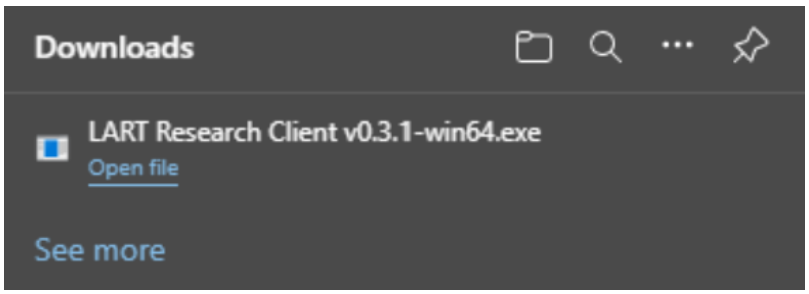


Fig. 2.1: Open file from downloads

Note: If you have Microsoft Defender active, you may be warned about running an unrecognised app.

This is expected behaviour for unsigned software downloaded from the internet. It is meant to get you to check that you've downnloaded the Software from a reputable source before running it.

This is fine if you've used our official download link above!

Click *Run anyway* to continue with the installation.

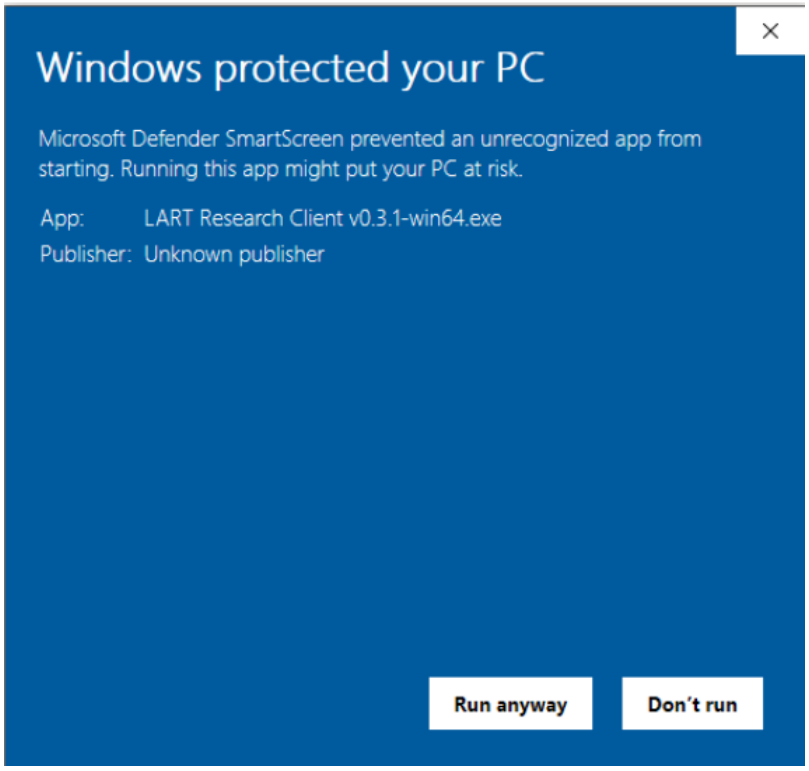


Fig. 2.2: Microsoft Defender SmartScreen

3. Select you preferred install mode.

   We recommend choosing *Install for me only* for most use cases, which will install the app only for the current user.

   However, you may wish to install the app for all users. For example, if you're installing on a shared university or lab computer and want to centrally manage the installation for all users (requires administrator privileges).

   If in any doubt, choose *Install for me only*.

4. Click *Yes* to allow L'ART Research Client to make changes to your device (namely, to install the app).
5. Read and accept the licence agreement.

   You must accept the agreement before installation can begin.

6. Select the destination location for your app.

   Normally you should be able to leave this at the path already suggested by the installer, which will be the default directory for app installation for your system and the chosen installation mode.

   Make sure you have at least 65MB of free disk space on your device.

   Click *Browse...* if you wish to change the installation path of the app.

7. Click *Install* to install the L'ART Research Client app on your device.
8. Complete setup by clicking *Finish* and enjoy!

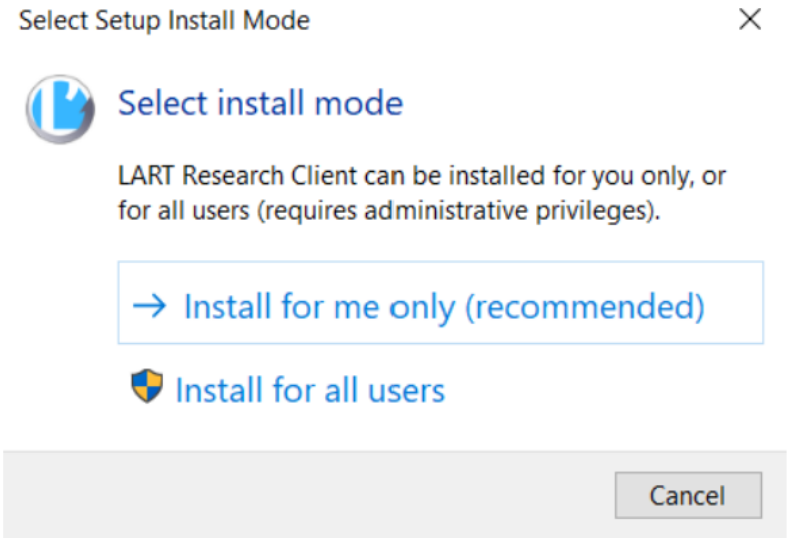


Fig. 2.3: Install mode setup

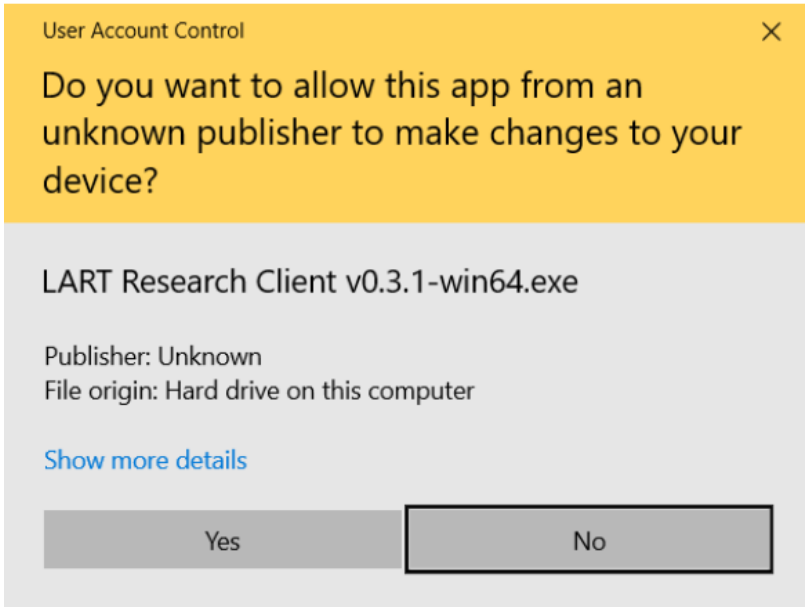


Fig. 2.4: User account control screen

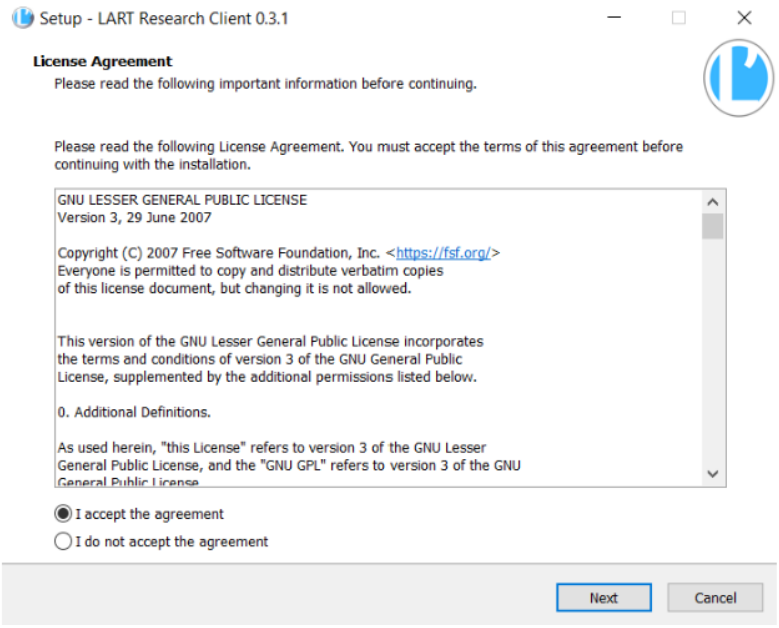


Fig. 2.5: License agreement

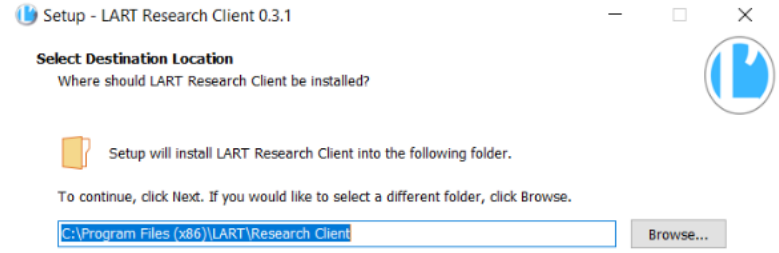


Fig. 2.6: Select destination location

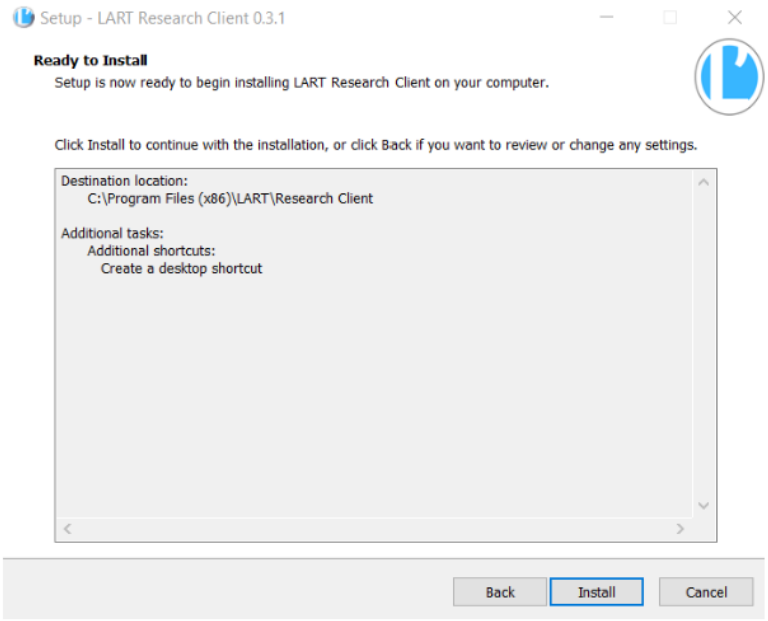


Fig. 2.7: Install Research Client app

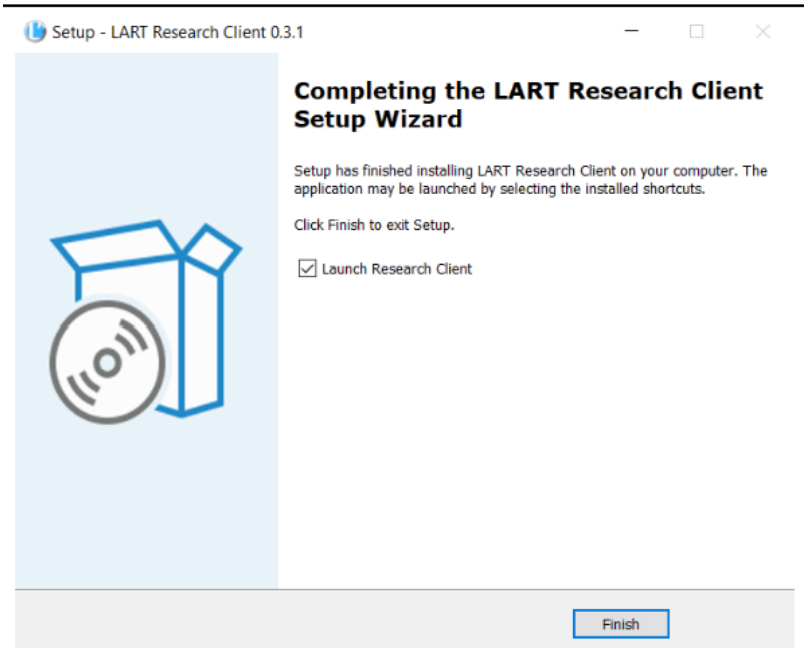


Fig. 2.8: Complete setup of Research Client app

2.2.2 Installing on Linux

This currently requires building from source or running as a Python package (requires Python 3.10), but should run if you have Chrome or Chromium installed.

Note: Help wanted!

We would welcome help for developing a sustainable workflow to build distributables for Linux. If you have any experience with packaging for one or more Linux distributions (e.g. as flatpaks, *.debs, snaps, etc.) and would be willing to help with that please do get in touch!

Running as a Python package

The easiest way is to run directly from source. On Ubuntu 22.04, follow the steps below the get the source code and all the dependencies installed. The last line will run the Research Client.

```
sudo apt install chromium-browser python3-pip python3-tk -y
python3.10 -m pip install pipenv
cd ~/
wget https://github.com/lart-bangor/research-client/archive/refs/tags/v0.4.1.tar.gz
tar -xf ./v0.4.1.tar.gz
rm ./v0.4.1.tar.gz
cd research-client-0.4.1
python3.10 -m pipenv install
python3.10 -m pipenv run python ./manage.py run
```

If you want to make an executable shortcut, create a file with the executable flag (+x) in your `~/.local/bin` directory. You can do this by following these steps:

```
$ cd ~/.local/bin
$ echo > research-client
$ chmod +x research-client
$ gedit research-client
```

In the editor that pops up, enter the following text and then save the file:

```
#!/usr/bin/env bash

cd ~/research-client-0.4.1
python3.10 -m pipenv run python manage.py
```

After saving the above, you can now launch the Research Client from the terminal by just typing in `research-client` and hitting `Enter`. (You may need to log out and log back in if this doesn't work straight away…)

Building from source

Alternatively, if you want to build the app properly for your system, you can follow the steps for *Setting up the development environment* from the *Developer Guide*, and then just run `python3.10 -m pipenv run python manage.py build` from inside the project's root directory.

This will produce a tarball (`*.tar.gz`) in the `./dist/linux/` directory containing the full set of binaries for the application, which can then be installed in the appropriate way for you system or run directly from the executable therein.

The only real advantage this might offer is if you want to install the Research Client on several machines, as you can just copy over the tarball, exctract it and run the app, without needing to worry about any dependencies (they are all packaged together when the executable is built). There is no real additional advantage over running as a Python package.

2.2.3 Installing on MacOS

This currently requires building from source or running as a Python package (requires Python 3.10), but should run if you have Chrome or Chromium installed.

> **Caution:** App termination issue on MacOS
>
> There is currently an issue where the app may not terminate correctly on MacOS after the main window has been closed. If the background Terminal window remains open after a few seconds, this may have to be closed manually and the user may have to confirm that they want to really terminate the process.
>
> This is not harmful beyond the annoyance value, as long as the user does not close the Terminal window *before* they have finished the data collection.
>
> For more information see #37.

To build from source follow the same instructions as for Linux above, with some adjustments necessary (such as using **`port`** instead of **`apt`**). Since we don't currently have full instructions that have been tested on MacOS for this, it will be preferable to run as a Python package unless you want to actively figure out any problems you might encounter during the build.

To run as a Python package, follow these instructions:

1. Install Chrome from the app store (or install Chromium using your preferred method).
2. Install the latest version of Python 3.10x (that is version 3.10.10 as of the time of writing not 3.11.x!) from the official Python releases for MacOS.

   After runnin the installer, open a Terminal and check that `python3 --version` prints something like "Python 3.10.10". This means python has installed correctly and you're ready to continue.
3. Install **`pipenv`** with the command `pip3 install pipenv`. This should print out a success message at the end of the process. You can ignore any messages it might print about updating pip itself (or follow the instructions it provides if you like).
4. Now run the following commands in your terminal to set up the package from source:

```
cd ~
curl -L https://github.com/lart-bangor/research-client/archive/refs/tags/v0.4.1.tar.
↪gz -o research-client.tar.gz
tar -xf ./research-client.tar.gz
rm research-client.tar.gz
mv research-client-0.4.1 research-client
cd research-client
```

```
python3 -m pipenv install
```

5. You can now launch the app from within a terminal, provided you are in the directory `~/research-client`, using the following command:

```
python3 -m pipenv run python3 ./manage.py run
```

Obviously, you will not want to open a Terminal, do `cd ~/research/client` and then type `python3 -m pipenv run python3 ./manage.py run` every time. You can create a shortcut which can be clicked to launch the app by following these additional steps:

1. Make an executable `.command` file inside the `~/research-client` directory by running the following in a Terminal:

```
cd ~/research-client
echo > research-client.command
chmod +x ./research-client.command
open -a TextEdit ./research-client.command
```

2. In the editor that popped up with the last command above, copy and paste the following code, then save and close the file.

```
#!/usr/bin/env bash

cd ~/research-client
python3 -m pipenv run python3 ./manage.py run
```

3. Once you've created the `research-client.command` file as per the two steps above, you can just locate it in Finder (e.g. launch Finder, then in the top click on *Go -> Home*, and open the `research-client` folder) and then drag and drop the `research-client.command` onto your dock.

   When you click on the file in the dock now, it should launch a Terminal window together with the app.

2.3 Getting started

Upon starting the app on your device, both the L'ART Research Client app and a terminal window will open.

Figure XX - L'ART Research Client home screen

The terminal window prints out information that might be helpful with troubleshooting and should not be closed (the app will freeze/become unresponsive if the terminal window is closed).

When you close the app, the terminal will also close automatically.

However, if you close the terminal, the main app window will not close automatically.

On the app home screen, you will see the list of available tasks in the centre of your app, and a side menu containing several options, some of which are context-dependent. For example, the side menu contains options for Discarding Attempts, Collecting Consent, Unlocking the App, Exporting Data, App Settings and a dialogue providing information about the app.

These options are described in the linked sub-sections of the documentation.

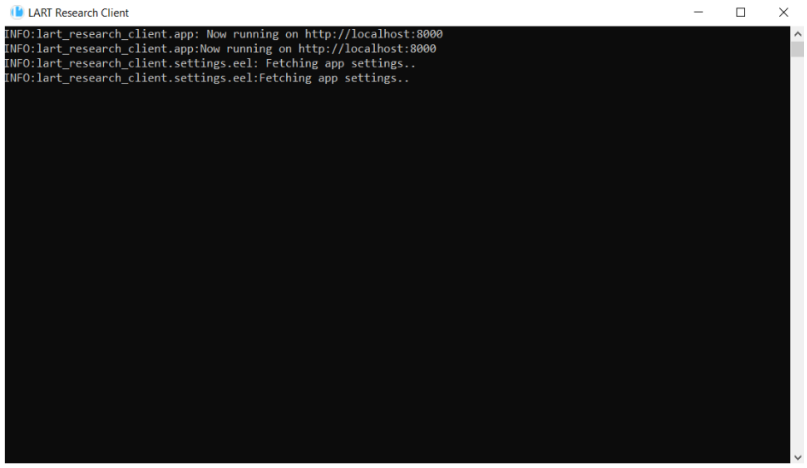


Fig. 2.9: L'ART Research Client Terminal

2.4 Setting up data collection and obtaining consent

There are two possible paths available to the researcher to collect data with the L'ART Research Client. Your preference will depend on how you wish to obtain informed consent from your participants.

Should you wish to obtain your participants' consent on paper for any reason (e.g., you require your participant's signature, or you prefer to work with physical copies of ethics-related documentation) then you will start data collection by clicking directly on the research tool you require under *Choose a task* on the app's home screen.

In the current version (0.4.1), the tasks available are: *LSBQe*, *AToL*, *AGT*, and *Memory Game*.

The second option offers the researcher an integrated digital avenue to obtain informed consent, which negates the need to handle physical information sheets, consent forms and signatures.

This can be done by using the generic consent form provided **(see Figures 16 & 17 below)** or by linking it to your own digital consent form.

To obtain consent digitally, open the side menu on the top left-handby clicking on the "burger menu" icon. Then click on *Informed Consent*.

After selecting the required language version and entering a unique participant ID, your study's consent form, information sheet and eligibility criteria will appear below.

If the participant gives their consent and confirms their eligibility through marking their respective boxes, they will be automatically advanced to the start screen for the first task (see Task Sequencing on how to set this).

The task start screen is the same start screen that researchers who opt for the LSBQe without digital informed consent will see after they select a task from the app's home screen.

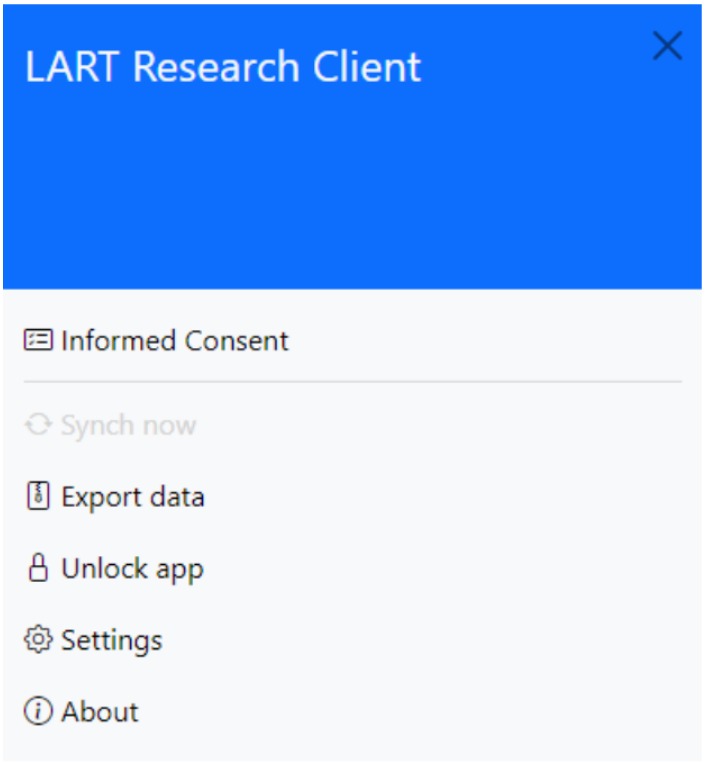


Fig. 2.10: The side menu of the L'ART app

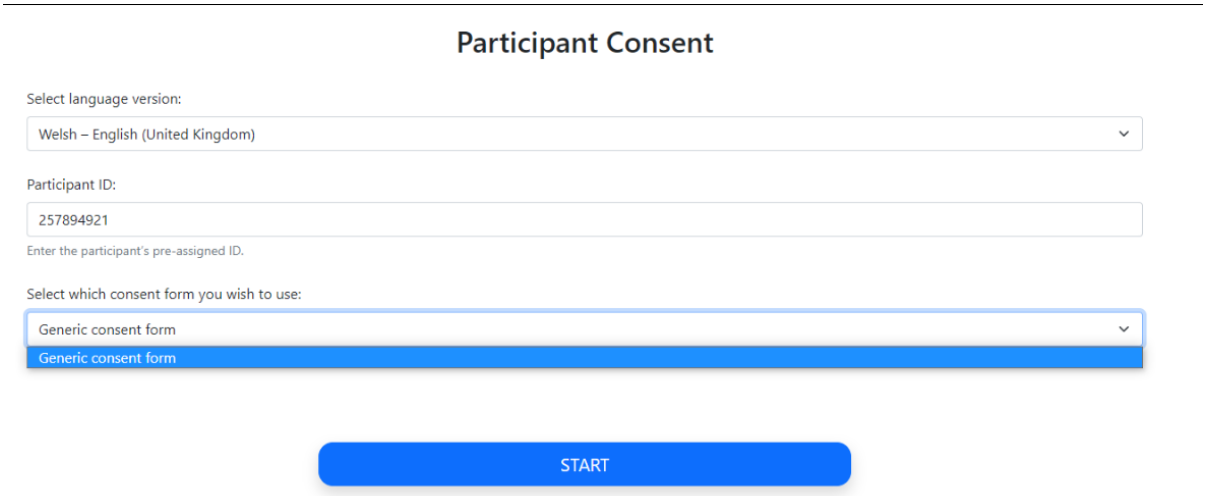


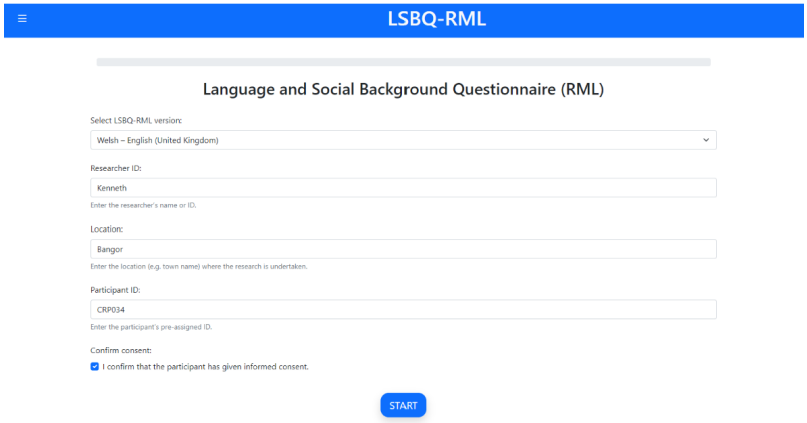
Fig. 2.11: Participant consent start screen

2.5 Collecting Responses

Upon starting the LSBQe, you are first asked to enter the relevant information for your study. These are:

1. Which version of the LSBQe you require, i.e. what localisation is pertinent to your study, which determines factors such as the primary (and possibly secondary) language displayed during the LSBQe, the suggested list of alternative languages, and the education level indicators.
2. A researcher ID, e.g., the name of the researcher conducting your experiment.
3. The location, i.e., the area, city, or town where the research is being undertaken.
4. A participant ID, i.e., the unique pre-assigned ID for your participant.
5. Consent confirmation, i.e. whether the participant has given their consent (either digitally or on paper).

Note: If you have collected consent via the digital consent form in-app, the app will pre-populate the information on this page using the information entered previously on the informed consent form.



2.5.1 User input

While all tasks within the L'ART Research Client can be completed with a touch interface or keyboard-only input, we strongly recommend that users are provided with access to both a keyboard and a pointing device (e.g. a mouse or trackpad).

This ensures the best user experience and the sliders used to collect continuous data are more accurate when used with a pointing device rather than a keyboard.

A significant portion of the tasks available on the L'ART Research Client make use of sliders. Sliders are displayed in a lighter shade with their indicator displayed in the middle by default, and they must be moved at least once for the answer to be valid.

Once moved, the slider will turn a darker shade of blue to show that the slider is active and has been moved by the participant.

Should the participant want to keep the slider in the middle of the bar, the slider will need to be moved once and then moved back to the middle point of the bar.

If the participant fails to move the slider, a red flag will appear **(see Figure 19)** and the participant will not be able to advance

Note: Some sliders provide a **"not applicable"** tickbox, which negates the need for the user to interact with that slider.

Fig. 2.12: The slider function, using the LSBQe as an example.

2.6 Research task: LSBQe

In the LSBQe, the task start screen is followed by the three main sections of the LSBQe on Language and Social Background, Language and Dialect Background, and Community Language Use Behaviour respectively.

For more details on the contents of the LSBQe and how this differs from the standard version of the LSBQ, see [Breit-Tamburelli-EtAl-2023].

Any mandatory fields that haven't been completed by the participant will be flagged up if the user attempts to continue to the next page without having fully completed any section of the LSBQe or the response entered in a field is invalid (e.g. text entered in a field expecting a date).

The user is given instructions on how they should complete the missing fields if this happens.

For researchers using the app, or a specific localisation of the LSBQe for the first time, it might be useful to complete the LSBQe and purposely leave all fields blank before trying to submit so they can read through and familiarise themselves with the user-feedback provided for each field.

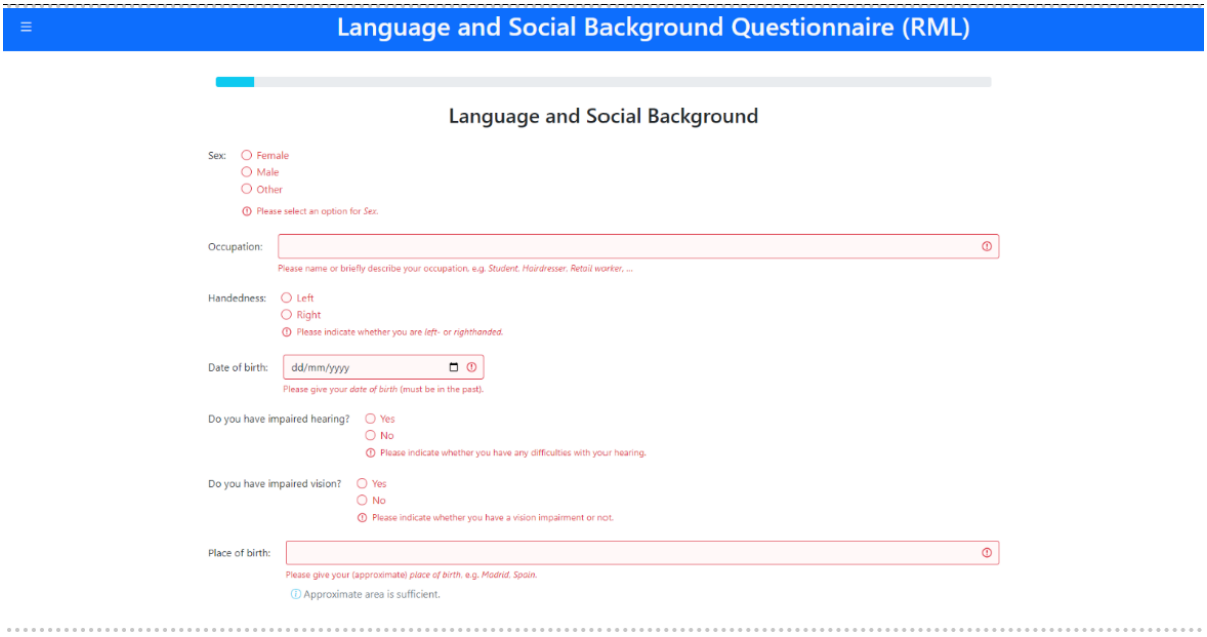


Fig. 2.13: Mandatory fields that remain unanswered or contain invalid input will be flagged in red

2.6.1 Loading a generic version of the LSBQe

Several generic versions of the LSBQe (e.g., English, German, Italian) are available for you to use if the languages pertinent to your research location are not available amongst our four LSBQe versions, or if you prefer a generic or customisable version of the LSBQe.

You can select a generic version of teh LSBQe from the dropdown list. For example, if you wish to use the generic version for British English, you would choose "English-generic (United Kingdom)".

This version of the LSBQe will give you English and "Other Language" at every juncture where both languages are named.

2.6.2 Customizing a generic version of the LSBQe

You may wish to customize a generic version of the LSBQe if you would like the LSBQe to present a specific language pair to use during your study.

Generic versions can be identified by the fact that the file name contains the sequence [Zzz], a placeholder code for "unknown language" (for example, the file for the generic version for British English is called `[EngZzz_Eng_GB]`).

If you wish to customise a generic version of the LSBQe, open the relevant file (e.g. `[EngZzz_Eng_GB]` for British English, or `[GerZzz_Ger_DE]` for German, and so on) by following the path below:

C:UsersusernameAppDataLocalProgramsLARTResearchClientlart_research_clientlsbqversions

Firstly, you **must "save as"**, following the ISO standard code sequence for standard code sequence generating) **(see Figure 22)**.

For example, if you wish to customize a version for English and Irish for use in Ireland through the medium of English, you will create a file called `[EngGle_Eng_IE]` **(see Figure 23)**.

After your new version is saved, you must change the **"versionID"** and **"versionName"** to reflect your customization. Your **"versionID"** should match your file name.

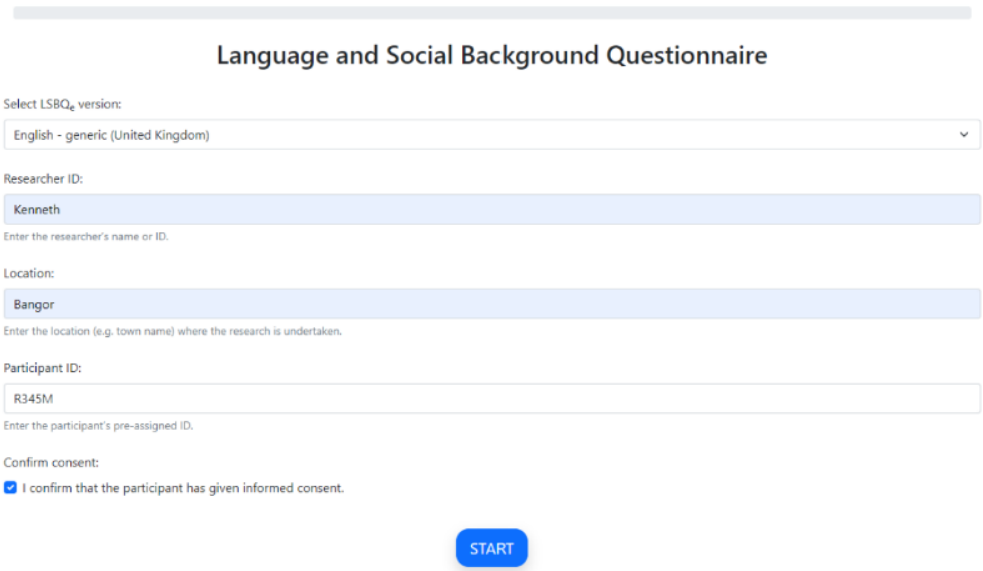


Fig. 2.14: Loading a generic version of the LSBQe

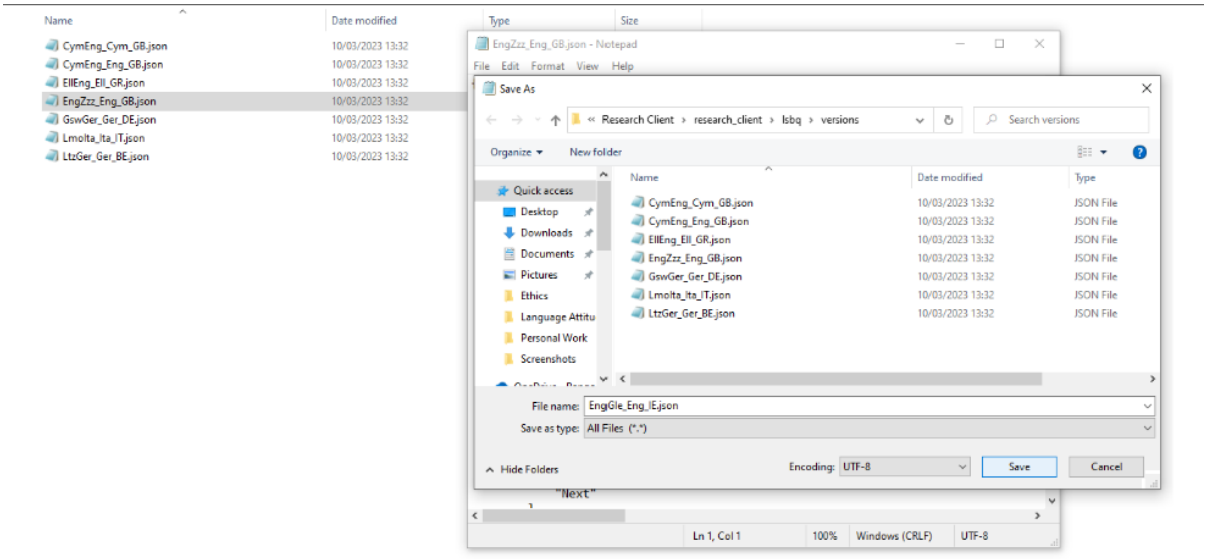


Fig. 2.15: Save the generic files as and follow the ISO code sequence

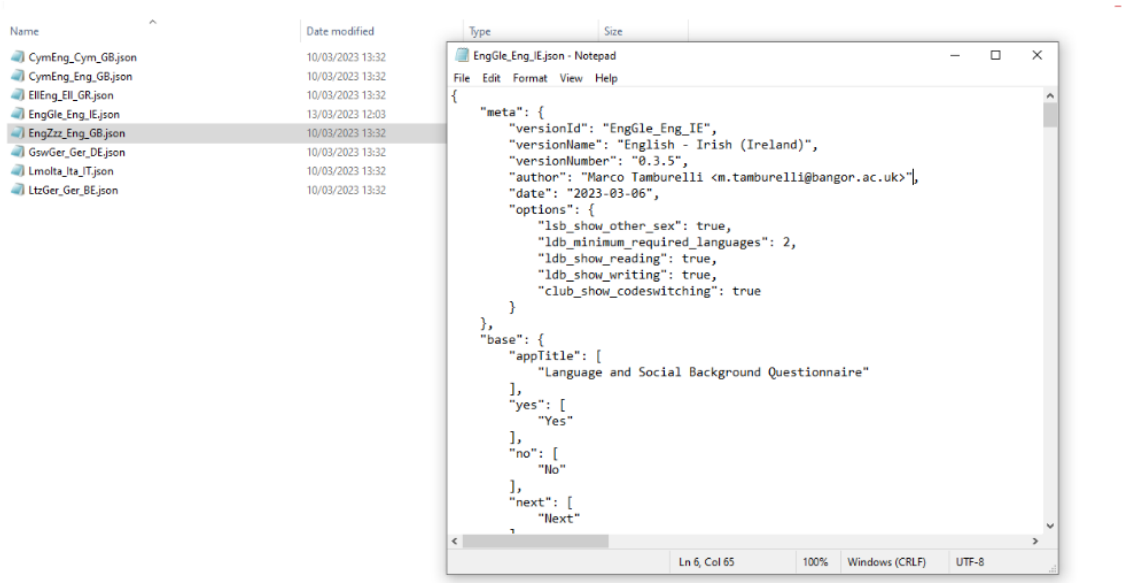


Fig. 2.16: New LSBQe file EngGle_Eng_IR

A further customization that you can make inside the file relates to how your LSBQe version will refer to the language you wish to include.

To do this, you must search for **"RML"** in your *[EngGle_Eng_IE]* and change **"the other language"** to the language name you wish to be displayed. In our current example that would be **"Irish"** as shown in Figure 24 below.

It is not mandatory to include English as one of the languages on your LSBQe version. For example, if you require an LSBQe version to study Ulster Scots and Irish in Northern Ireland, you would call the file `[ScoGle_Eng_GB]` and apply the relevant changes in Figure 22 and Figure 23.

Additionally, in order to change the default **"English"** in the LSBQe, you would have to search **"MajorityLanguage"** and change each instance of "English" to "Ulster Scots" **(see Figure 25)**

Note: Note that the third label in the file name `[ScoGle_Eng_GB]` remains **"Eng"**, as this refers to the language in which the LSBQe is presented, which in this case is still English.

See here for details on file naming and ISO codes.

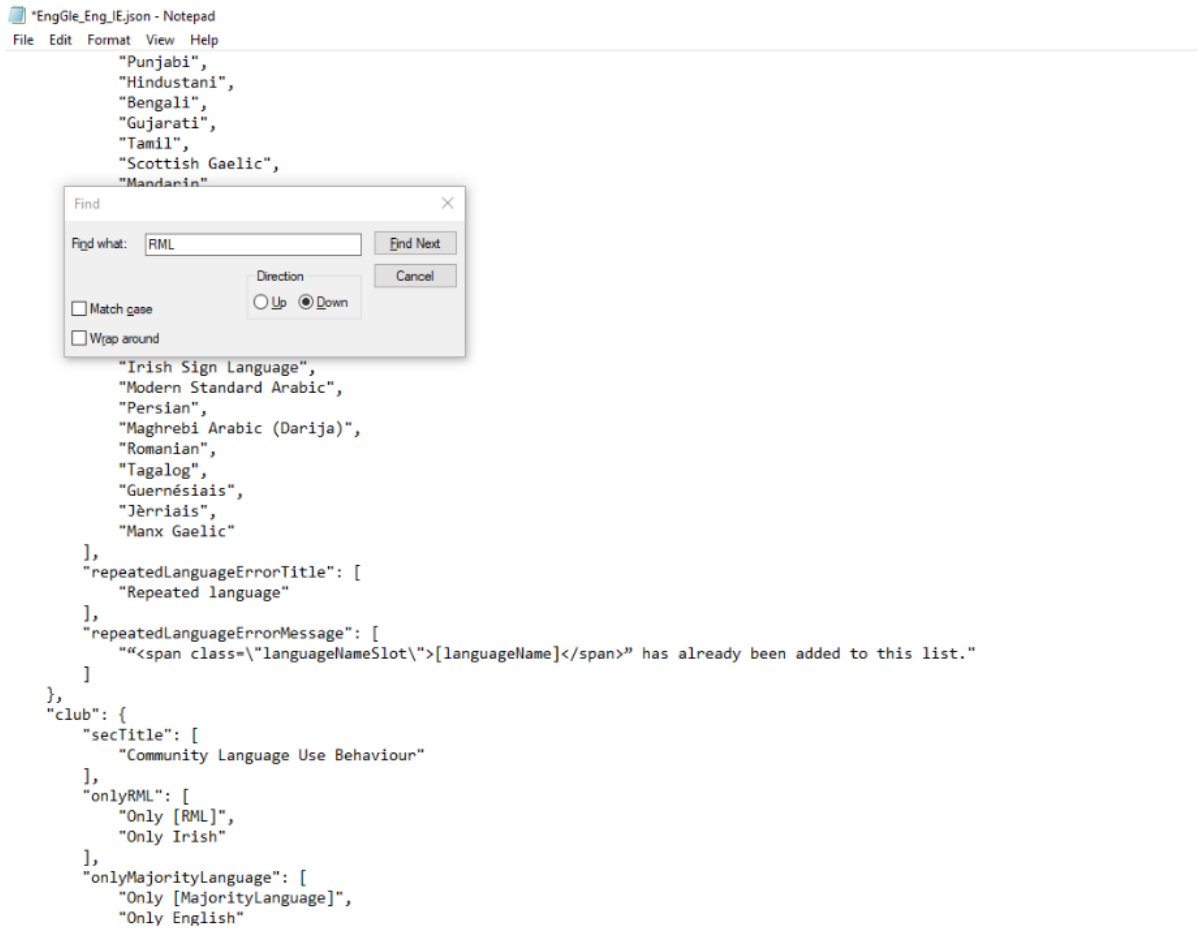


Fig. 2.17: Customizing inside your LSBQe file.

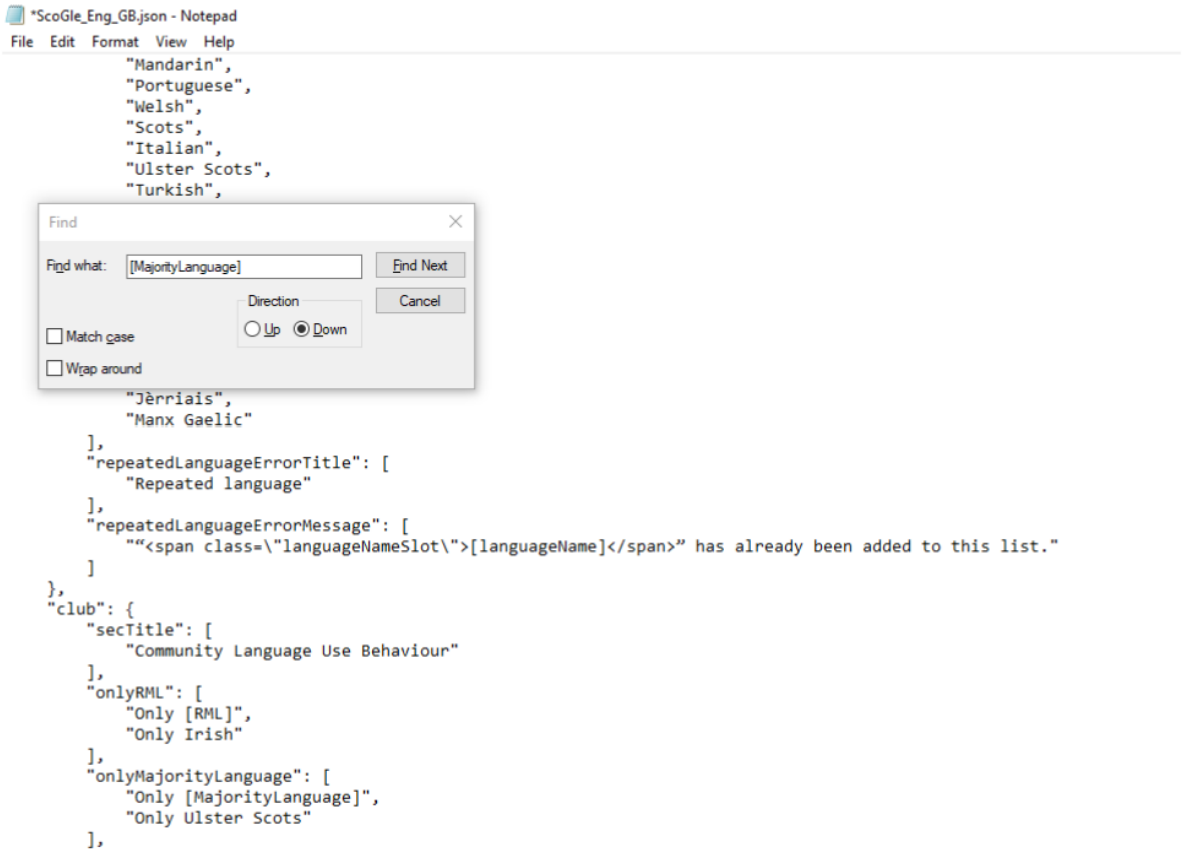


Fig. 2.18: Customizing both languages in your generic LSBQe file

2.6.3 Excludable Questions

The LSBQe allows users to include or exclude certain questions depending on the nature of the language communities to be researched (see Breit et al. 2023 for details on the rationale behind these choices).

Below you'll find instructions on which questions allow this option and how to go about excluding them.

"Other" Sex

As default, the LSBQe contains three options that a participant may select as their sex: "Female"; "Male"; "Other".

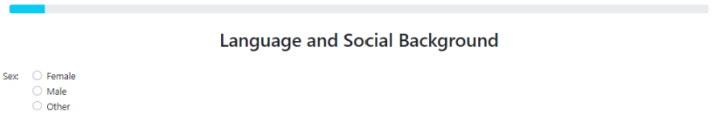


Fig. 2.19: Default options for sex on LSBQe

However, some researchers may prefer to use a binary choice (e.g., where biological sex is a research variable) and therefore exclude "Other" from the available options.

To do this, open your LSBQe version file from the following path:

`C:\Users\username\AppData\Local\Programs\LART\ResearchClient\lart_research_client\lsbq\versions`

With the file open, you will see that below the section "options" the line labelled **"**`lsb_show_other_sex`**"**` is set to `true`:

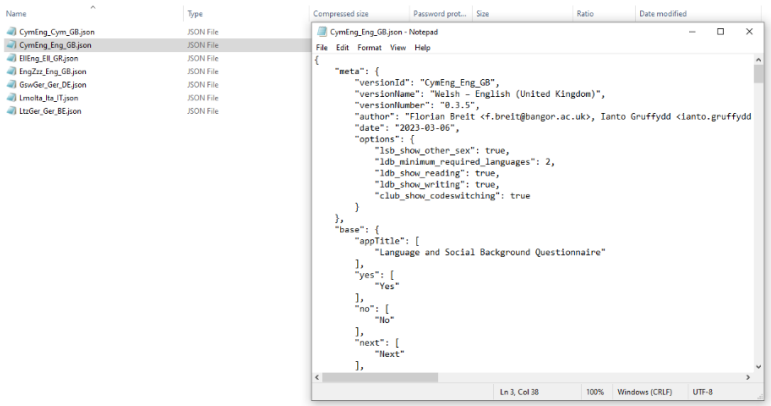


Fig. 2.20: The feature "lsb_show_other_sex" set to "true"

To exclude the **"Other"** option in your version of the LSBQe you simply need to set that option to **"**`false`**"**.

Note: Make sure to restart the app so that the change can take effect.

If you wish to change it back to including **"Other"**, you must reverse the above procedure and change the setting back to `"true"`.

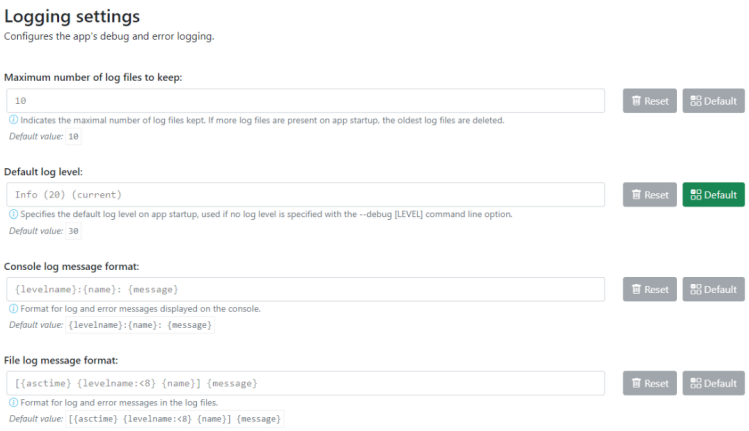


Fig. 2.21: The feature "lsb_show_other_sex" has been changed to "false"

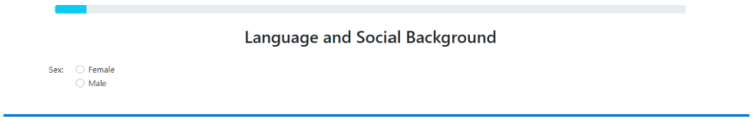


Fig. 2.22: How the question appears in the app after removing "Other"

Minimum required languages

In the "Language and Dialect Background" section, the opening question asks participants to list all the languages and dialects that they speak and give information regarding where they learned each of them, when they learned them, and if there were significant periods where the participant did not use any of them.

By default, the LSBQe requires a minimum of two required language names, by presenting participants with two blank lines that must be filled before continuing.

While participants have the option of adding more language varieties via the *Add Line* button (i.e. for participants who are multilingual), only two lines will appear as default **(see Figure 30 below)**.

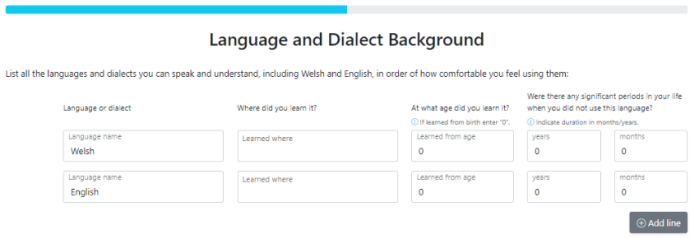


Fig. 2.23: The opening question on the Language and Dialect Background section set to two minimum required languages

Should you wish to make three or more languages the default without having to add more lines, for instance if you're researching trilingualism within a community, you may set the minimum required languages to three.

To do this, firstly, open your LSBQe version file from the following path:

`C:\Users\username\AppData\Local\Programs\LART\ResearchClient\lart_research_client\lsbq\versions`

With the file open, you will see that below the section "`options`" the line labelled "`ldb_minimum_required_languages`" is set to **"2"**:

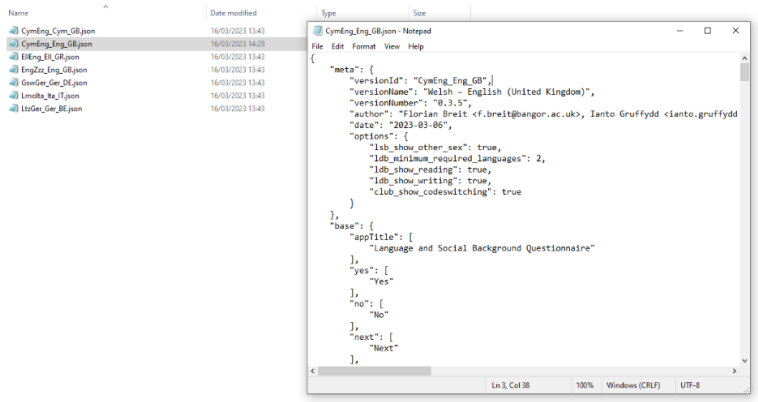


Fig. 2.24: The feature "ldb_minimum_required_languages" set to "2"

To change this to a different number, e.g., 3, you simply type "3" in place of "2":

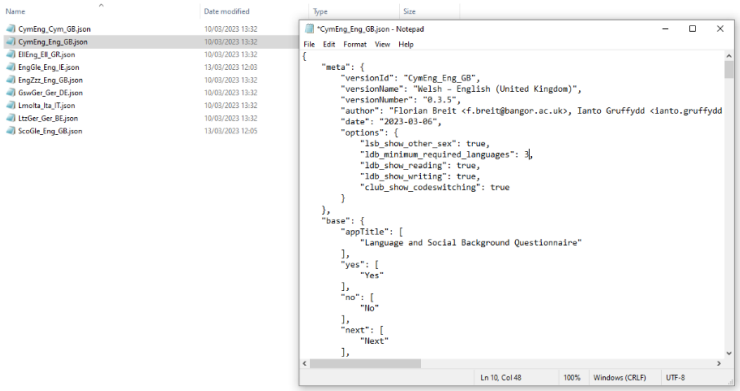


Fig. 2.25: Changing the minimum required languages to three

Note: Make sure to restart the app so that the change can take effect.

If you wish to change the option back to two languages, you must reverse the above procedure and change the setting back to **"2"**.

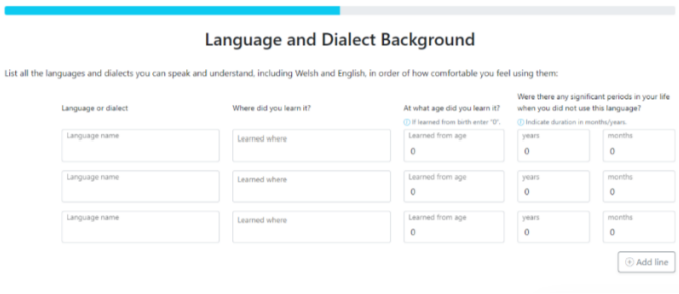


Fig. 2.26: How the question appears in the app with a minimum of three required languages

Reading and Writing:

In the "Language and Dialect Background" section, participants are asked how much time they spend engaged in speaking, listening, reading, and writing in each of their languages.

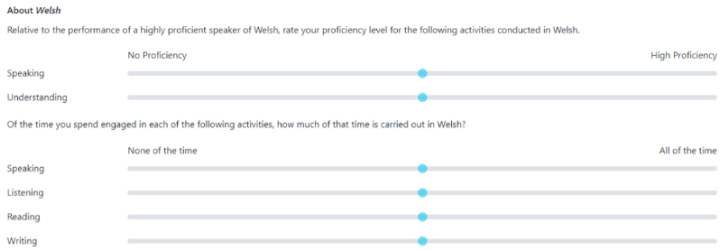


Fig. 2.27: How the question appears in the app with "Reading" and "Writing" options

The "reading" and "writing" parts of the questions can be removed. For example when researching a community whose one or more languages is only/mostly oral or doesn't have an accepted orthographic system, making the "reading" and "writing" options irrelevant to participants.

To remove the "reading" and "writing" options, firstly, open your LSBQe version file from the following path:

`C:\Users\username\AppData\Local\Programs\LART\ResearchClient\lart_research_client\lsbq\versions`

With the file open, you will see that below the section **"options"** the lines labelled **"ldb_show_reading"** and **"ldb_show_writing"** are set to **"true"**`:

To exclude these options from your version of the LSBQe, simply change the values to **"false"**:

Note: Make sure to restart the app so that the change can take effect.

If you wish to change it back to including "reading" and "writing", you must reverse the process and change the values back to **"false"**.

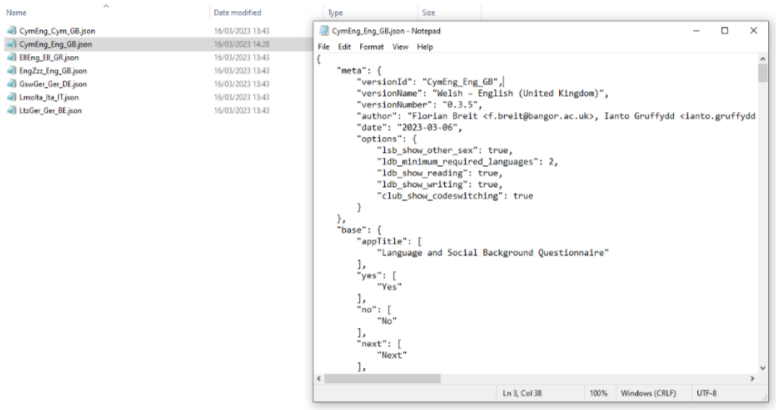


Fig. 2.28: The features "ldb_show_reading" and "ldb_show_writing" are set to "true" by default

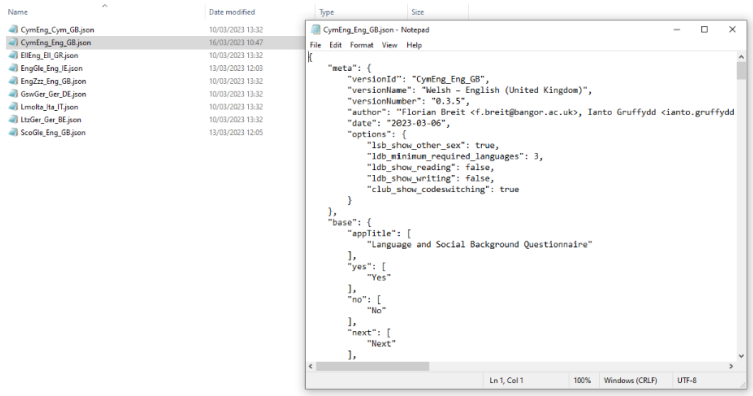


Fig. 2.29: Setting the "reading" and "writing" options to "false"

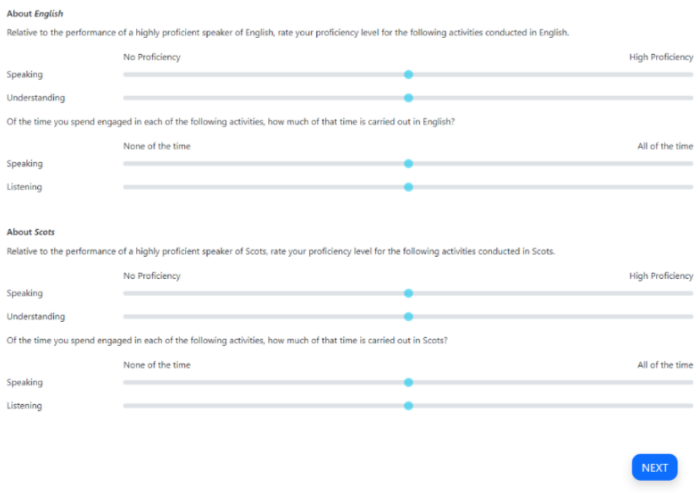


Fig. 2.30: How the question appears in the app with "Reading" and "Writing" options removed

Show code-switching

The LSBQe's Community Language Use Behaviour section contains a final section on code-switching where participants are asked how often they code-switch in different contexts **(see Figure 38)**.

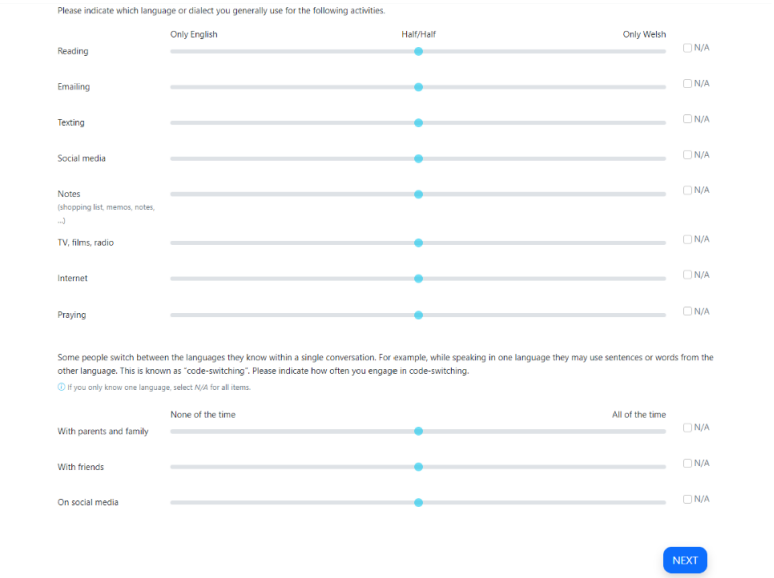


Fig. 2.31: CLUB section with code-switching question included

The code-switching question can be removed if this information is not required in your study.

To remove the code-switching question, firstly, open your LSBQe version file from the following path:

`C:\Users\username\AppData\Local\Programs\LART\ResearchClient\lart_research_client\lsbq\versions`

With the file open, you will see that below the section **"`options`"** the line labelled **"`club_show_codeswitching`"**` is set to **"`true`" (see Figure 39)**

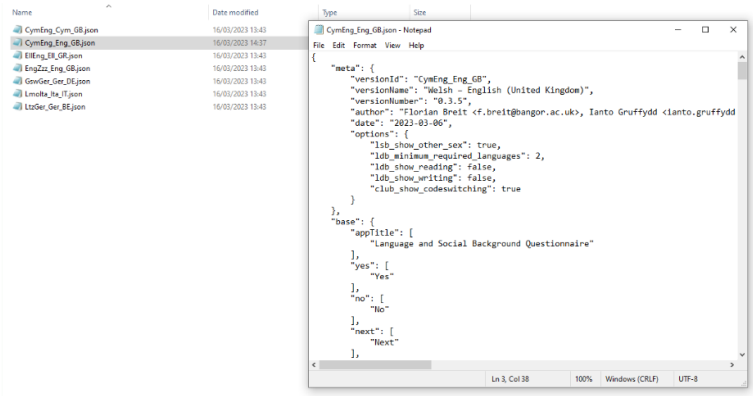


Fig. 2.32: The feature "club_show_codeswitching" is set to "true" by default

To exclude the code-switching question from your version of the LSBQe, simply change the value to **"`false`" (see Figure 40)**

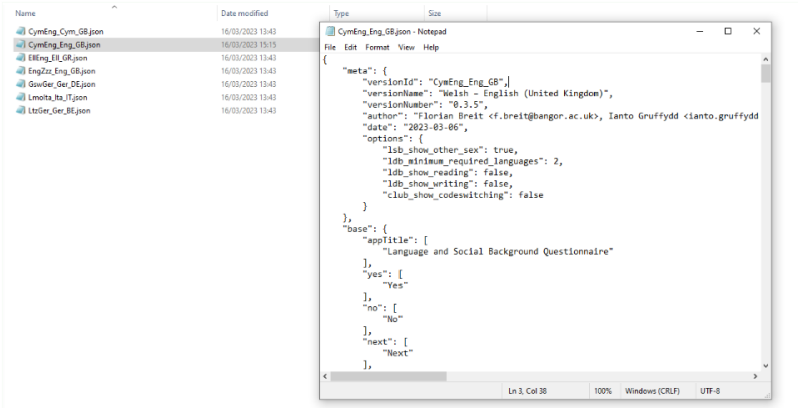


Fig. 2.33: Setting the codeswitching option to "false"

Note: Make sure to restart the app so that the change can take effect.

After removing the code-switching section, the CLUB section finishes on the question prior to the code-switching question that asks participants to indicate which language or dialect they generally use for various activities **(see Figure 41)**.

If you wish to change it back to including the code-switching question, you must reverse the process and change the value back to `"true"`.

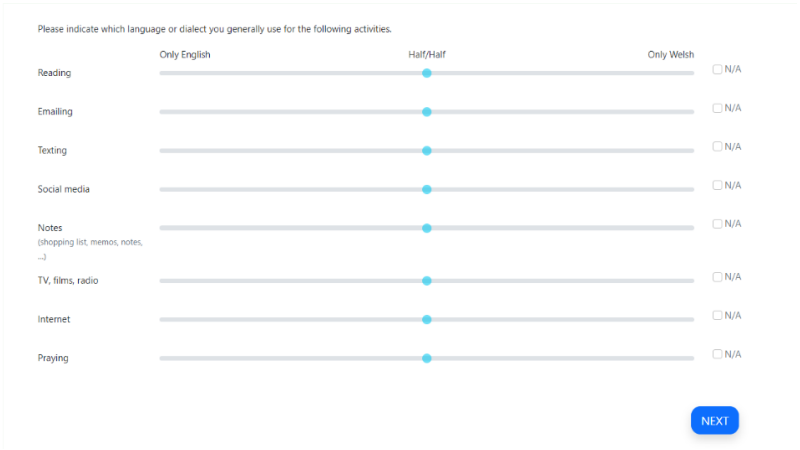


Fig. 2.34: How the question appears in the app with "Reading" and "Writing" options removed

2.7 Research task: AToL

The AToL begins with a start screen where you must select a version, input Researcher ID, Location, Participant ID and confirm that consent has been obtained by ticking the relevant box.

You will not be able to advance without completing each respective part of the start screen **(see Figure 42)**.

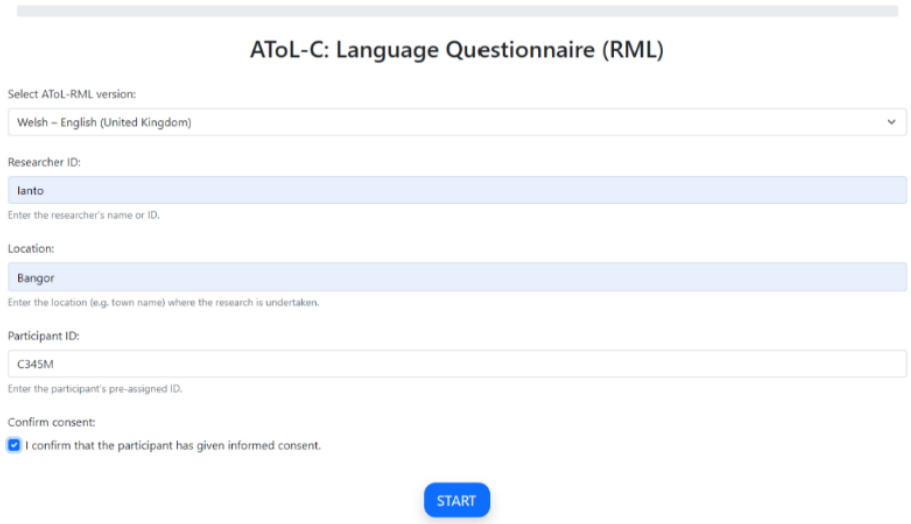


Fig. 2.35: AToL Start Screen

The next screen begins the AToL proper, asking the participant to rate the relevant languages, depending on the AToL version selected.

The majority language always appears first due to sociolinguistic plausibility, for instance, because all instructions appear in the majority language in the original AToL versions for the L'ART research client app.

The bipolar adjective pairs are always generated in a random order (the specific order for each participant is recorded in that participant's the data file). The AToL presents the statement **"The X language is…"** followed by the AToL's bipolar adjective pairs which are rated by using the sliders as seen in Figure 43 below.

The AToL is a task that involves exclusively using sliders, and the order of the adjective pairs is randomised for each participant.

For ease of analysis, your result file for a given participant **(see Figure 59** here) reports the order in which the adjectives were presented for that participant.

After activating each slider and providing a rating along each bipolar adjective pair, the next button activates in a darker shade of blue, indicating that you may advance to the next part of the AToL.

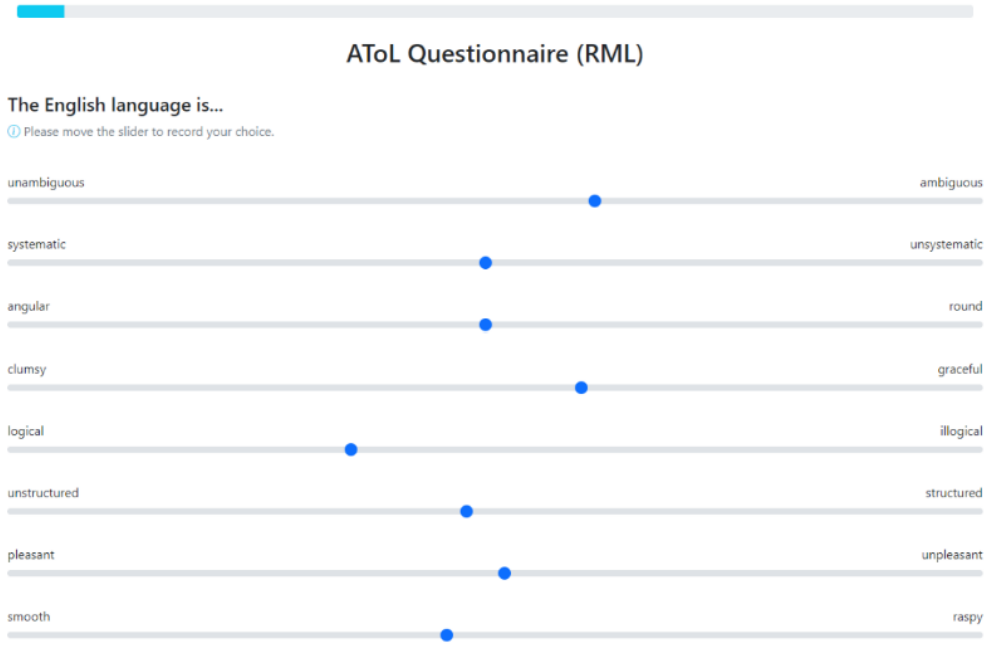


Fig. 2.36: AToL rating

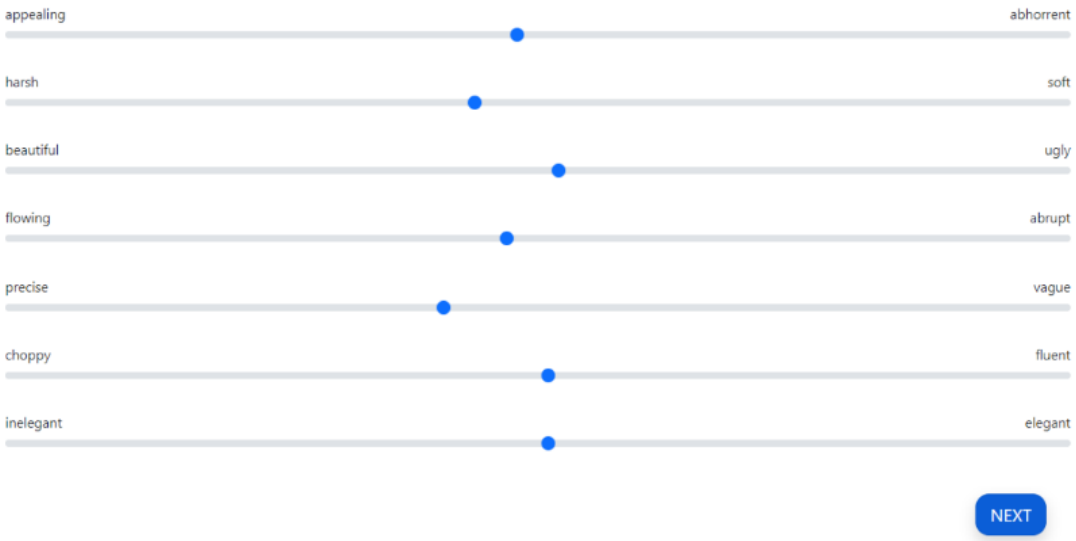


Fig. 2.37: Completed AToL section with an activated "next" button

2.7.1 Loading and customizing a generic version of the AToL

As for the LSBQe, several generic versions of the AToL are available (e.g., English, German, Italian). Unlike the LSBQe, however, it is not possible to load a generic version of the AToL without customizing it.

This is due to the fact that while the LSBQe may refer to "the other language", the AToL is dependent on naming each language under investigation at the top of every page (see Figure 43 "the **English** Language is…")

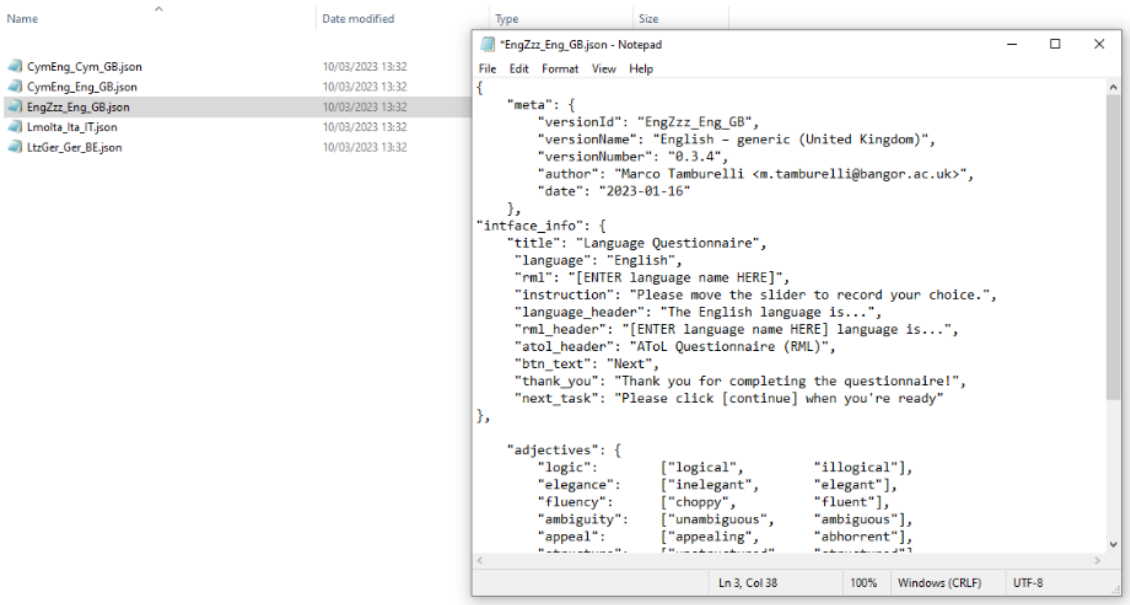


Fig. 2.38: Generic AToL file EngZzz_Eng_GB

Firstly, as seen in Figure 22 you must open the generic file and **"save as"** in order to make a copy ready for customisation.

Note: Generic versions can be identified by the fact that the file name contains the sequence [Zzz], a placeholder code for "unknown language" (for example, the file for the generic version for British English is called `[EngZzz_Eng_GB]`).

After that, change the "versionID" and "versionName" to reflect your customization. Following the English and Scots example presented here, this would be `[EngSco_Eng_GB]`

Your file name should match your "versionID", which must follow the ISO standard code sequence (see the note)

In order to produce a customized version of the AToL, you must also change both "rml" and "rml_header" to indicate the language(s) pertinent to your AToL version.

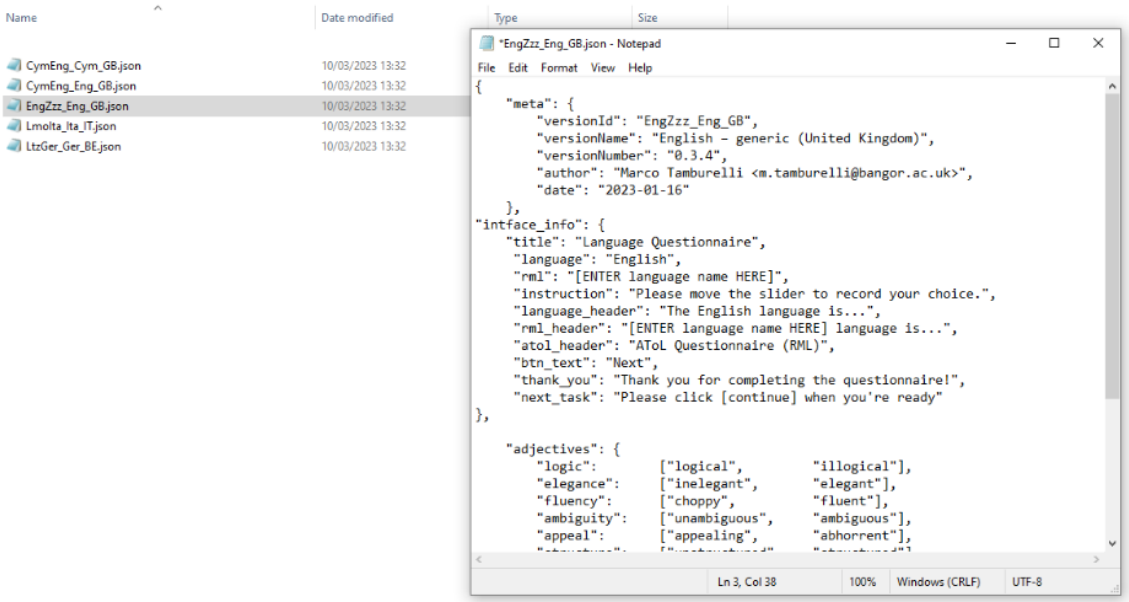


Fig. 2.39: New customized AToL file EngSco_Eng_GB

2.8 Research task: AGT

The Audio Guise Tool (AGT) allows users to run either a Matched Guise Technique (MGT; [Lambert-Hodsgon-EtAl-1960]) or a Verbal Guise Technique (VGT; [Markel-EtAl-1967]) (see [Breit-Tamburelli-EtAl-2023] for details).

Differentiation between MGT or VGT is executed via your audio recordings, and it is explained in some detail below.

2.8.1 Loading recordings for the AGT

The AGT requires **thirteen** recordings in order to function. Eight recordings are classed as experimental recordings, four are classed as filler recordings and one is a practice recording.

For an MGT setup, you **must** load eight experimental recordings from four speakers, with each speaker providing a recording in either language variety.

For a VGT setup, you would load eight experimental recordings from eight speakers, with four speakers providing a recording in one language variety, and the other four speakers providing a recording in the other language variety.

What to do with the four fillers is left up to you. For instance, in an MGT setup, continuity with experimental stimuli may be preferred, so the four fillers could be recorded by two speakers, with both speakers providing a recording in each language variety.

The practice guise is presented first during an AGT and allows the participant to familiarise themselves with an AGT without testing experimental stimuli. Practice stimulus design is decided by the researcher, for instance, you may wish to produce a recording of yourself talking about a neutral topic for the same length as the experimental and filler recordings.

Sound files must be labelled appropriately in order for the AGT to execute the audio correctly. The audio recording for the practice guise **must** be named "practice.mp3"; audio files for fillers **must** be named beginning with "f" plus the number of the filler (i.e., f1.mp3 to f4.mp3); and experimental guises **must** be marked "s" plus number to denote

your speaker, then underscored before either "maj" or "rml" to mark the different language varieties (see Figure 48 for example).

These labels must be assigned **consistently** to the file names, but it does not matter which variety you choose to label "maj" and which "rml" (though if working with a majority language and a regional/minority language it may help analysis if you use "maj" for majority and "rml" for minority language). What matters for app functionality is that you assign the "maj" label to one language/variety and the "rml" label to the other, keeping it consistent throughout your set-up.

In the example below in Figure 47, `"maj"` indicates Chinese recordings and `"rml"` indicates English recordings. Do note that this is done **consistently** for all recordings.

> **Warning:** File names are **case-sensitive** and **must** be written identically to how they appear here in order for the AGT to function.
>
> All sound files must be in mp3 format for the AGT to function.

To load your own recordings for the AGT, open the folder `[mgt]` by following the path below:

`C:\Users\username\AppData\Local\Programs\LART\ResearchClient\lart_research_client\web\audio\agt`

Create a new folder which follows the ISO standard code sequence (see the note here for standard code sequence generating) to store the sound files for your AGT. For example, for an AGT set-up to work with Chinese-English bilinguals in Singapore and use English as the language of presentation, you would create a folder called `"ZhoEng_Eng_SG"`, as follows:

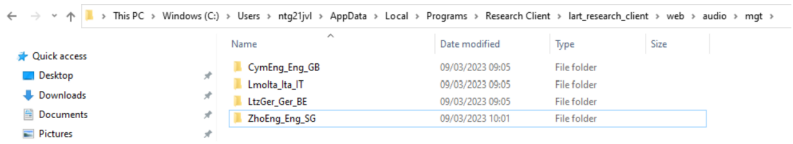


Fig. 2.40: New folder **ZhoEng_Eng_SG** created following the ISO standard code sequence

Inside your folder, paste your own sound files but copy the standard code sequence described as above for file names when naming your sound files.

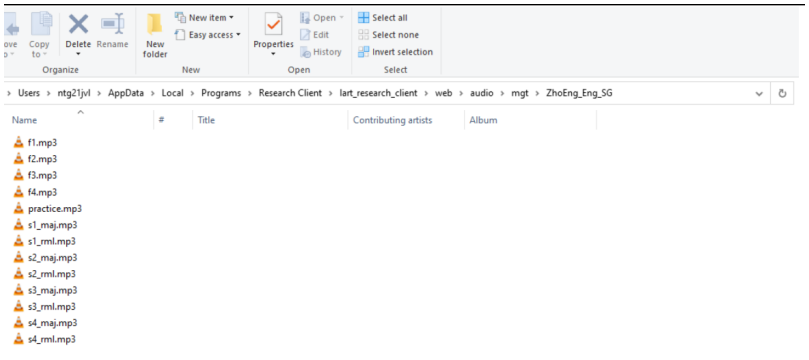


Fig. 2.41: Sound file names for AGT following the standard code sequence

Your files will now play when you start the AGT and select your AGT version on the start menu.

insert screenshot once agt version has been implemented in the app

2.8.2 Loading a generic version of the AGT

Similarly to the LSBQe (here) and AToL (here) an "English-generic" version of the AGT is made available. Select the generic version of the AGT by selecting the "English – generic (United Kingdom)" version from the LSBQe version drop down list.

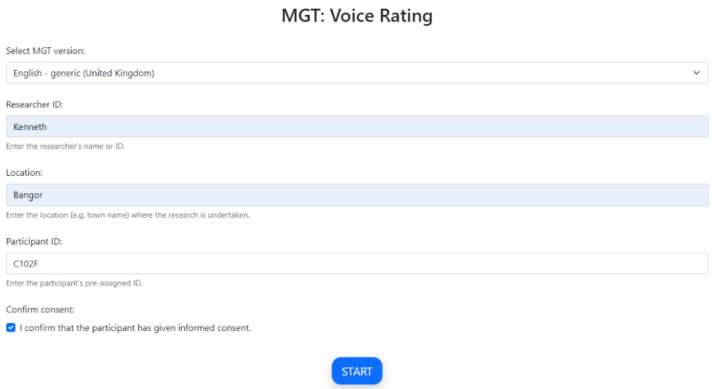


Fig. 2.42: Loading a generic version of the AGT

2.8.3 Customizing a generic version of the AGT

You may wish to customize a generic version of the AGT if you would like the AGT home screen to list a specific language pair to use during your study. There are several generic versions available (e.g., English, German, Italian), all of which can be customized. Generic versions can be identified by the fact that the file name contains the sequence [Zzz], a placeholder code for "unknown language" (for example, the file for the generic version for British English is called `[EngZzz_Eng_GB]`).

If you wish to customise a generic version of the AGT, open the relevant file (e.g. `[EngZzz_Eng_GB]` for British English, or `[GerZzz_Ger_DE]` for German, and so on) by following the path below:

C:\Users\username\AppData\Local\Programs\LART\Research Client\research_client\agt\versions

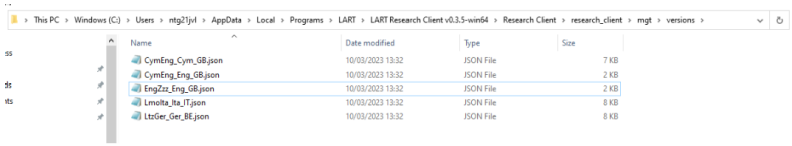


Fig. 2.43: File path and AGT version files

Firstly, you **must** ensure that you **save as**, following the ISO standard code sequence for standard code sequence generating).

Secondly, you must change the **"versionID"** and **"versionName"** to reflect your customization. Your **"versionID"** should match your file name.

Figure 51 follows the same example presented in Figure 17.

It is not mandatory to include English as one of the languages on the "English – generic" AGT version. For example, if you are a dialectologist who requires a VGT to study Ulster Scots and Irish in Northern Ireland, you would call your file `[ScoGle_Eng_GB]` and apply the changes to versionName and VersionID as above.

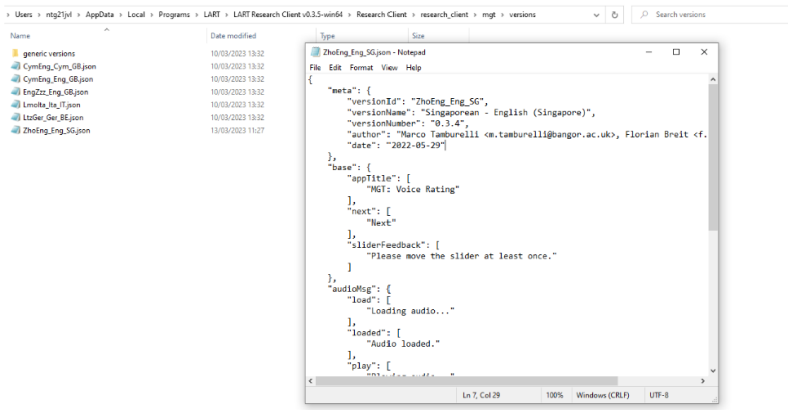


Fig. 2.44: New AGT file ZhoEng_Eng_SG

Note: Note that the third label in the file name `[ScoGle_Eng_GB]` remains **"Eng"**, as this refers to the language in which the AGT is presented, which in this case is still English.

Seeing as no language names are mentioned anywhere in the AGT, the only other component where the languages you choose to study are important is the recordings you load onto the AGT (see here). Everything else can remain the same as in the "English – generic" version.

2.9 Locking and unlocking the app

The app is always in a "locked" state when it is first started. The locked state prevents the user from (accidentally or purposely) carrying out certain actions, such as inspecting the logic behind the forms they see or using right-click context menus to reload or revert to an earlier screen.

Done unintentionally, this could lead to invalid, corrupted, or duplicate responses, and/or might give the participants information about the administered tasks that the researchers might not want them to have (at the point of data collection).

Researchers might find it useful however to unlock the app and access such functionality from time to time. For example, to go back to the previous screen if an error was inadvertently made, or to reload the current screen if for any reason something isn't rendered correctly. Unlocking the app is also useful for researchers who develop new localisations of a task (see here).

To unlock the app, open the side menu and click *Unlock app*. After unlocking the app, right click and these options (as well as a few more) will be available to you. If intervening during data collection, it is good practice to lock the app again once the necessary intervention has been carried out.

Do this by following the same steps as for unlocking.

To unlock the app, open the side menu and click "Unlock app". After unlocking the app, right click and these options (as well as a few more) will be available to you. If intervening during data collection, it is good practice to lock the app again once the necessary intervention has been carried out. Do this by following the same steps as for unlocking.

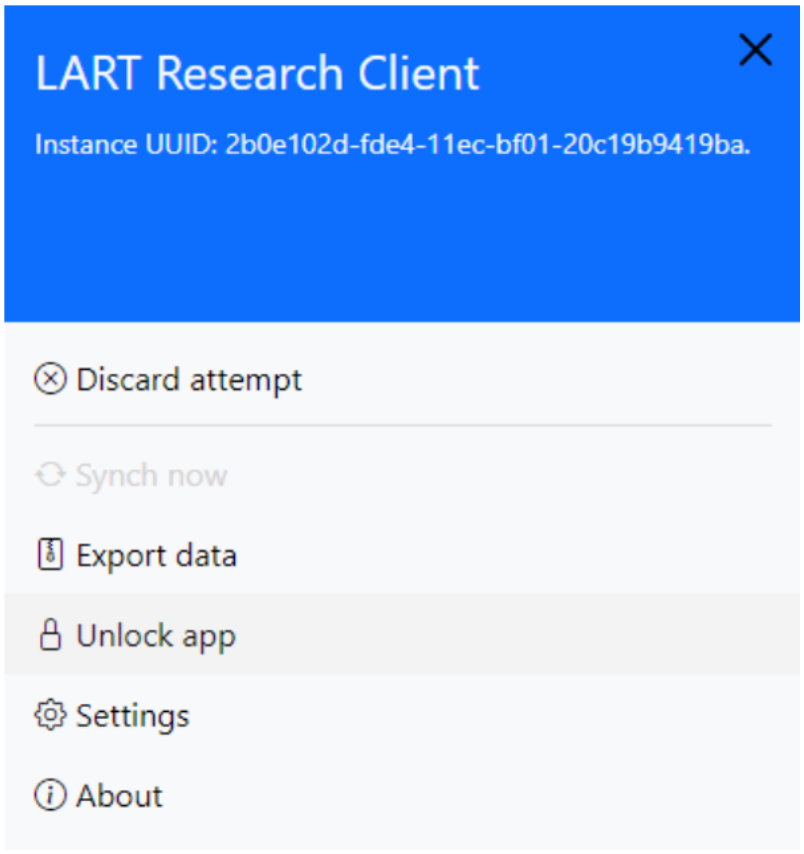


Fig. 2.45: Open the sidebar to unlock the app

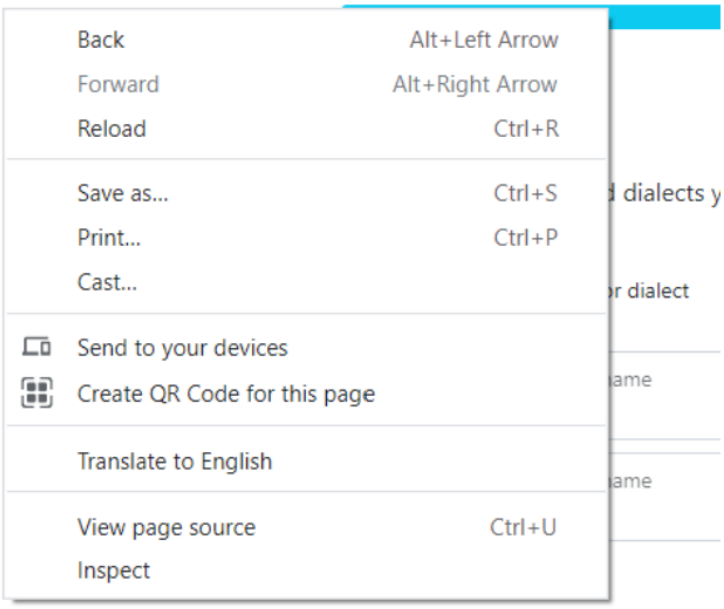


Fig. 2.46: Right click will reveal options available after unlocking app

2.10 Exporting data

Data for each task (e.g., informed consent, LSBQe) will be automatically stored in a JSON file on the computer running the app once the participant has submitted their response.

However, if you want to export the data manually, e.g., to make a backup to a different location or collate data collected on various devices, you can do this straightforwardly on the app.

For the simplest way to do this, open the side menu, and click on *Export Data*.

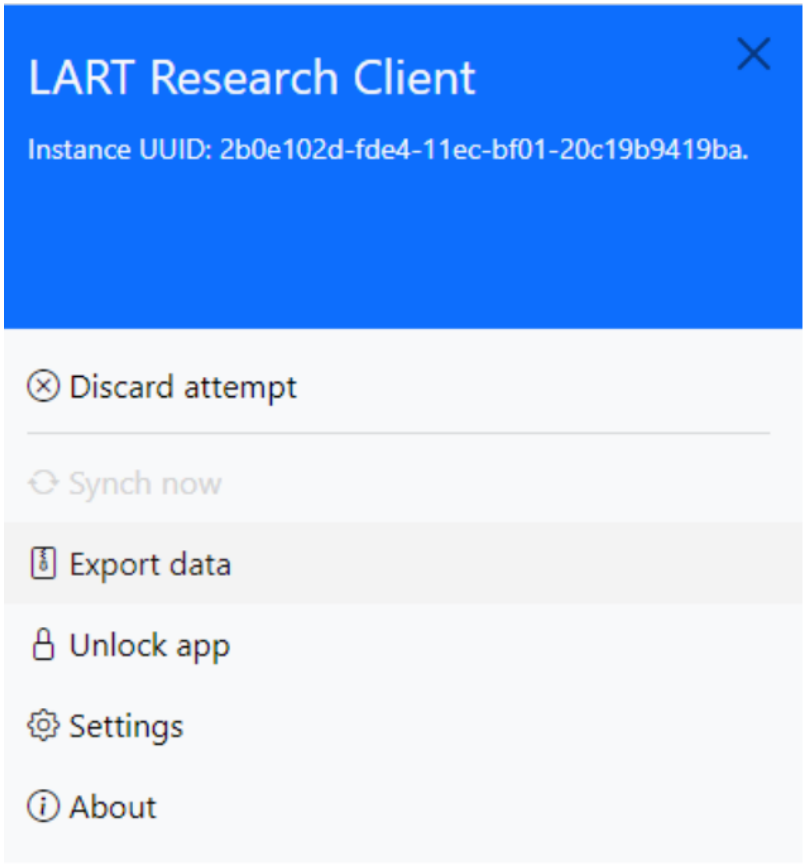


Fig. 2.47: Figure 54- Open the side bar to export data

Once you have clicked on export data, a dialogue will appear which allows you to save a ZIP archive containing all the responses currently stored on the computer in a location of your choice.

Note: Identify and remember your file path before saving, so that you know where to find your ZIP archive containing the exported data.

Your next step will be to verify that all the data collected appears as it should in your exported ZIP Archive file.

To find your data in order to verify that the data has been backed up correctly, firstly, follow the file path that your ZIP Archive file is located in.

In order to follow the file path, you must show hidden items in the *"view"* section your **"File Explorer"**.

Next, discover your ZIP Archive exported data file via the following path:

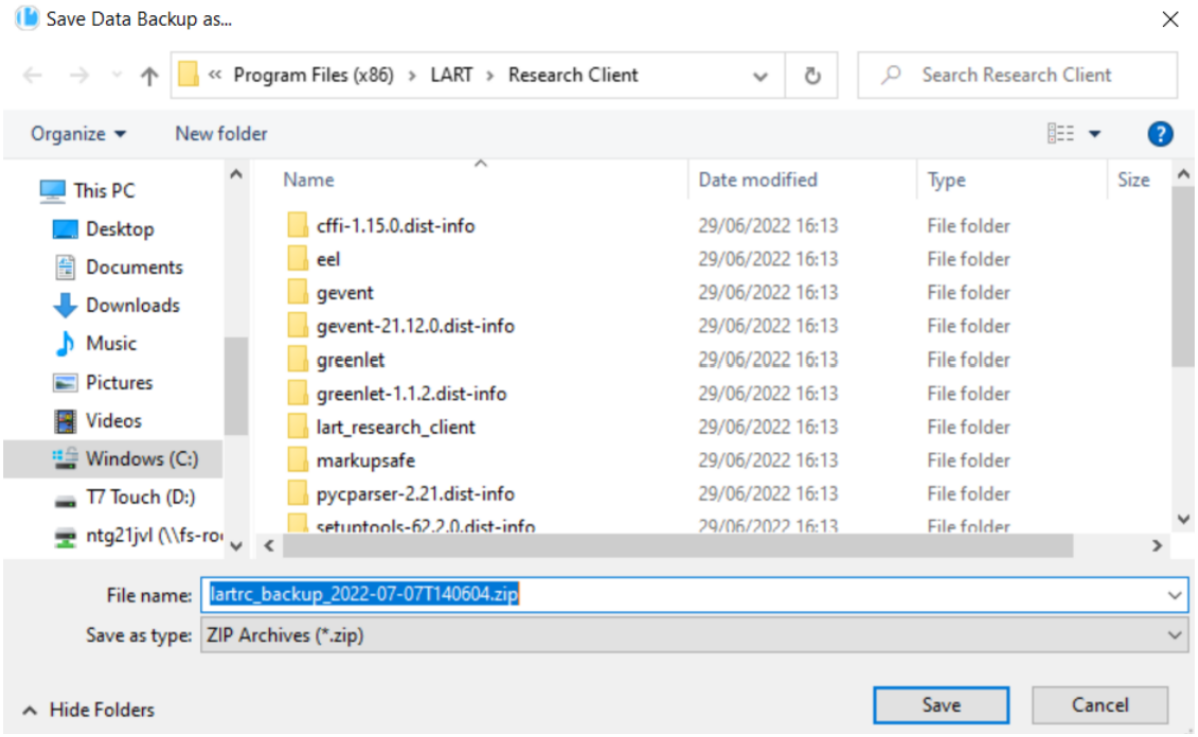


Fig. 2.48: Figure 55 - Saving the exported data

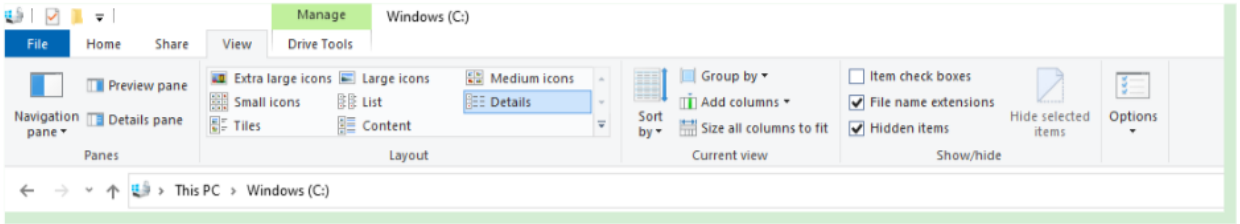


Fig. 2.49: Tick the "hidden items" box in order to display all of your file path

`C:\Users\username\AppData\Local\Programs\LART\ResearchClient\`

Open up your ZIP Archive back up and open the folder that corresponds to the task used in your study.

For instance, if you're verifying data that you collected from the AToL, open **"AToL-C" (see Figure 57)**.

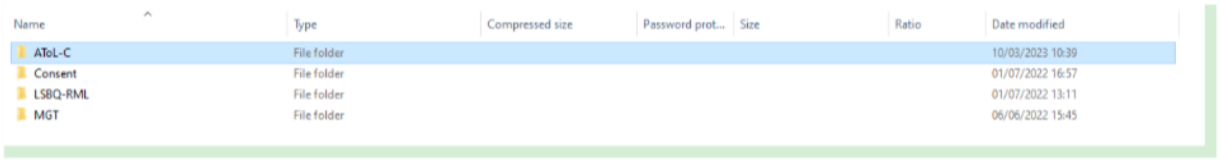


Fig. 2.50: Task folders located in your exported ZIP Archive

Located inside the file will be your **.json data** files, labelled in **"participant_date_time"** format **(see Figure 58)**.

Fig. 2.51: .json data file in participant_date_time format

File name accuracy should indicate if your data has been collected and exported properly, but you can open the file to verify completely that data was collected and exported accurately.

For instance, in the example below **(Figure 59)**, by observing that each AToL adjective pair, per language, equates to a **number** (how your participant rated the language for the trait via the slider mechanism).

For advanced users who want to automate export or backup of the responses, or monitor and integrate these files with some other system, the files can be accessed directly on the system.

On Windows these are stored in the Roaming profile by default, identified by the path `%AppData%LARTResearch Clientdata`.

On Mac OS X the default path will be `~/Library/Application Support/Research Client/data`.

For most Linux distributions the default path will be `~/.local/share/Research Client/data`.

Note: It is best practice to not modify or work with the original data files where this is avoidable.

This is expected behaviour for unsigned software downloaded from the internet. It is meant to get you to check that you've downloaded the Software from a reputable source before running it.

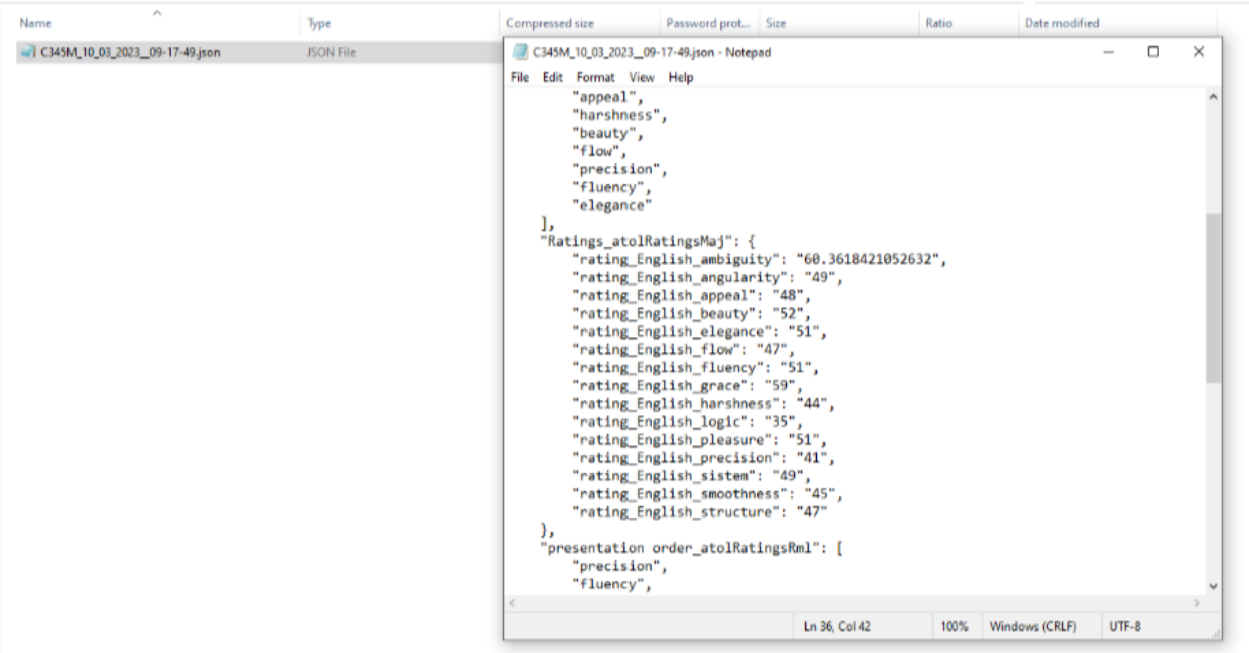


Fig. 2.52: Check that your data file contains data inputed by your participant

2.11 Discarding an attempt in progress

> **Warning:** The procedure described below might lead to the irretrievable loss of data.
>
> Please ensure that you read this information carefully and understand the ramifications of discarding an attempt before making use of this functionality.

Discarding a participant's attempt at a task can be done straightforwardly at any stage of the data collection process, including during the consent stage. To do so, open the side menu and click **"Discard attempt"**. Once the attempt has been discarded, the app will return to the app home screen.

Note that discarded responses will **not** be saved on the device and any information a participant may already have entered for that task will be irretrievably lost.

Discarding an attempt will usually be the appropriate choice where a participant choses to withdraw their consent during the study, and should align with the ethical procedures in place for your research. If a response has already been submitted successfully and consent is withdrawn retrospectively, the researcher **must** manually ensure that data is deleted where that might be required.

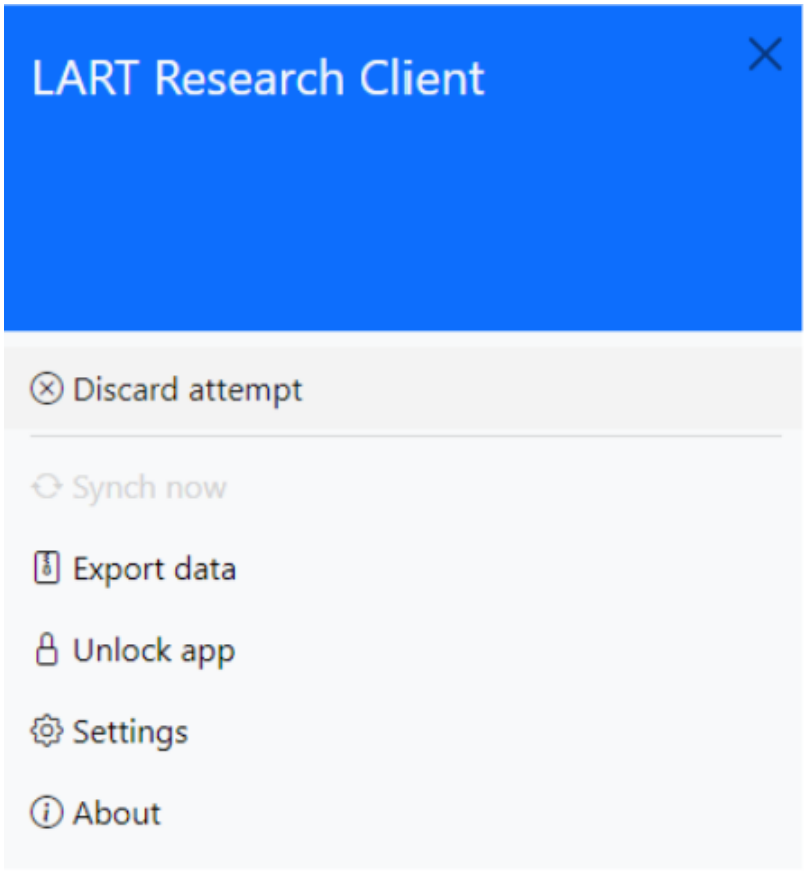


Fig. 2.53: Open sidebar to discard attempt

2.12 App Settings

The app's settings can be accessed via the app side menu. There are numerous aspects of the app that can be changed on the settings page. The various options are discussed in some detail below.

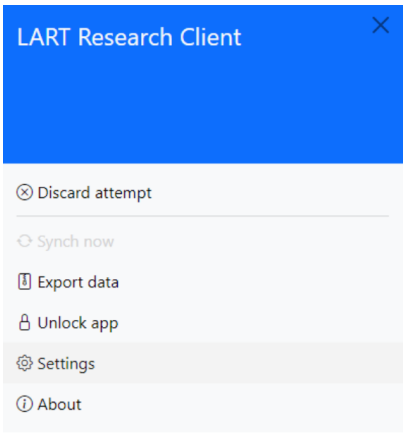


Fig. 2.54: Open the sidebar to enter settings

Remember to click *Save changes* and restart the app in order for the changes you make to take effect.

Tip: If you modify a setting, it will show up in gold. Pressing the red *Reset* button will revert to the previous setting. Pressing the green *Default* button will revert to the default setting. This is illustrated in Figure 10:

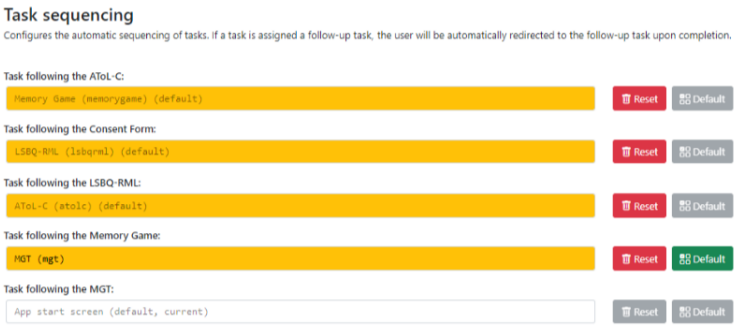


Fig. 2.55: Settings interface with modified task sequencing

2.12.1 General settings

The general settings section is used to configure basic running parameters of the app. These typically do not need to be adjusted.

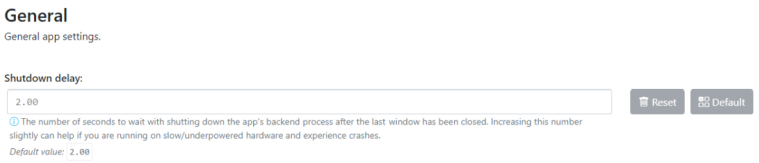


Fig. 2.56: Editing shutdown delay in the *general settings*

Shutdown delay

Type: Real number

Default: `2.00`

Shutdown delay is the amount of time (in seconds) that the app's backend process (basically, what you can see in the terminal window) waits before closing after you close the main app window.

Under normal circumstances there should be no need to adjust this. However, it can be beneficial to increase the shutdown delay when using an underpowered device e.g., a 4GB Surface Tab Go or some other device not meeting the recommended system requirements (see *Compatibility and Requirements* for more information).

Problems with limited system resources can lead to the app freezing or becoming unresponsive. Increasing the shutdown delay means that the app will wait longer in case the system temporarily delays the processing of expected signals and information.

2.12.2 Logging settings

Logging settings involves the app's debug and error logging functionality. While you will not usually have to access these files, they can contain useful information for researchers developing an extension for the app, those creating a new localisation of a task, or generally for diagnostic information if an unexpected error occurs.

You may be asked for information from the log files if you report a bug which will help us to reconstruct what happened when the error occurred on your computer.

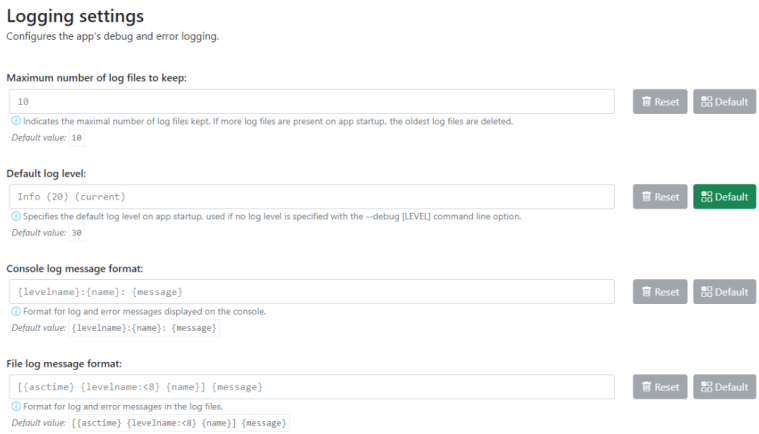


Fig. 2.57: Logging settings

Warning: The log files may potentially contain any of the information that a user/researcher/participant enters into the app while it is running.

For this reason, **you must apply the same information security policies to the log files as you do to the response data itself.**

If you share log files with a third party, you should ensure that they do not contain identifiable data which you would not otherwise share with that party.

You may want to "sanitise" your log files *(by manually removing any sensitive/identifiable data)* before sharing them and/or make sure that the other party is aware and capable of keeping this data secure in line with your policies.

Maximum number of log files to keep

Type: Integer

Default: `10`

The maximum number of log files to keep determines how many logs from previous runs of the app are kept, and once this number is reached old logs are deleted. By default, the app keeps logs files for the last 10 times it was started.

Default log level

Type: Integer

Default: `30`

The log level determines how detailed the log files are. The lower the numeric level, the more detail is stored in the log files.

Lowering the log level might be useful if you try to diagnose an error or bug and it is not apparent what led to the undesired behaviour from the existing logs (however, we recommend **not** doing this "just in case", as the amount of information might be overwhelming with log levels below 30).

Console log message format

Type: String

Default: `{levelname}:{name}:  {message}`

Modifies the format of log messages shown in the console window that runs in the background of the app.

The log message format is only relevant for advanced users and developers who may want to format logs in a specific way for working with their preferred analysis tools. If you are not sure what this is or how it works, there is no need for you to modify it.

For details on the formatting see the documentation of the `logging` package in the Python standard library.

File log message format

Type: String

Default: `[{asctime} {levelname:<8} {name}] {message}`

Modifies the format of log messages stored in the log files while the app is running.

The log message format is only relevant for advanced users and developers who may want to format logs in a specific way for working with their preferred analysis tools. If you are not sure what this is or how it works, there is no need for you to modify it.

For details on the formatting see the documentation of the `logging` package in the Python standard library.

2.12.3 Path and directory settings

The path and directory settings configure the paths used by the app for storing and reading various files, such as the data collected from participants, the app settings, and the log files.

If paths are modified it is best to always restart the app and fully test that everything is working as expected, including inspecting the stored data files after running a task.

On Windows, the app by default uses paths in the so-called *roaming* profile to store settings and data. This means that if you install the app on a networked domain computer, it's settings and data will transfer across to other computers in the domain where you log in with the same credentials. This is of course provided your system administrators have not modified the behaviour for roaming profiles on the domain, so it's a good idea to check for yourself that this works when you log in to other computers if you plan on relying on this feature in some way (we always recommend making your own backups and not overly relying on system backup features — those should be seen more as a second-line defense or 'backup of the backup' if anything).

> **Warning:** It is strongly recommended that you do not modify any of the app paths unless you are positively confident that you know what you are doing. Incorrect path information could lead to unstable behaviour and in the worst case even data loss.

Path for configuration files

Type: Path to a directory

Default: `%AppData%\LART\Research Client` (on Windows)

This is the path where the app will look for configuration files (such as `settings.json`, the file in which these settings are stored). As opposed to the other paths, changes to the value here will have no discernible effect and will revert automatically upon start-up. The path for configuration files thus mainly has informational value.

Path for data files

Type: Path to a directory

Default: `%AppData%\LART\Research Client\Data` (on Windows)

This is the path where data files, i.e. the participants' responses, from the app tasks are stored.

Path for log files

Type: Path to a directory

Default: `%AppData%\LART\Research Client\Logs` (on Windows)

This is the path where the app's log files are stored and will be handy to know if you ever have to debug or report an error. However, note the potential data security policy implications noted in the *Logging settings* section above.

Path for temporarily cached data and files

Type: Path to a directory

Default: `%LocalAppData%\LART\Research Client\Cache` (on Windows)

This is the path to a directory where the app may temporarily cache (sore, modify, delete) various filed during operation.

2.12.4 Task sequencing

The task sequencing settings allows you to configure which tasks (if any) should follow the completion of a specific task. This facilitates a more convenient data collection process where the user is automatically directed to the next task without the need for researcher intervention. This also negates the need to re-enter participant details (and the associated margin for error) at the start of each task, as these are transferred across tasks automatically.

Example: The deafault LSBQe sequence

For example, with the default settings, when the informed consent task is completed the participant will be automatically advanced to the LSBQe, and when the LSBQe is complete they will be sent to the conclusion screen before then being redirected to the app home screen (see Figure 12).

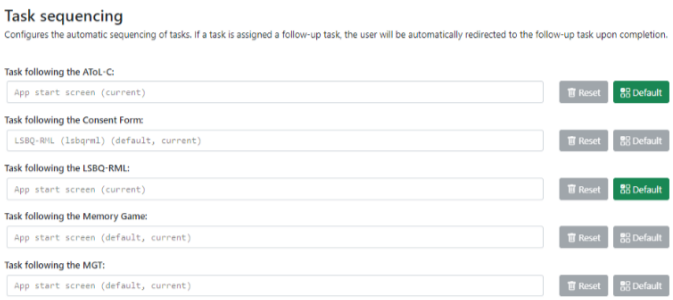


Fig. 2.58: Default sequencing: *Consent Form* > *LSBQe* > *Conclusion Screen* > *App start screen*

Note also that the sequencing doesn't rely on the entry point. If the participant is starting directly with the *LSBQe* in the default sequence, they will then still follow the remainder of that sequence, i.e. the *Conclusion Screen* followed by the *App start screen*.

Example: A custom sequence

You could decide to use any possible sequence consisting of available tasks, though note that you should only use the *Conclusion Screen* for the end of the sequence.

For instance, you may not want to require an electronic consent form for your study, thus removing the consent form from the sequence, and may want the LSBQe to advance into the AGT as is typical in linguistic studies where a background questionnaire precedes the main research method, followed by a conclusion screen to inform the participant they have completed all tasks and that they should await further instruction from the researcher. This sequencing is demonstrated in Figure 13:

Should you require every available task to be sequenced, you may also do so, as shown in Figure 14:

Fig. 2.59: Customised sequence: *LSBQe* > *AGT* > *Conclusion Screen* > *App start screen*

Fig. 2.60: *Consent Form* > *LSBQe* > *AToL-C* > *Memory Task* > *AGT* > *Conslusion Screen* > *App Start screen*

CHAPTER

THREE

QUICK TUTORIALS

3.1 Localisation and Adding Translations

The LSBQe is designed to allow easy implementation of interfaces in any language you choose.

At the moment, the languages available are **English, German, Greek, Italian,** and **Welsh**. The setup is for four bilingual communities: **Welsh-English, Lombard-Italian, Moselle Franconian-German**, and **Greek-English:**

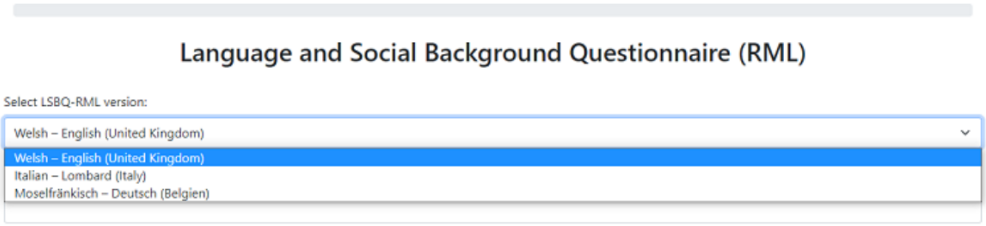


Fig. 3.1: Selecting a version of the LSBQe

However, both the working languages and the setup for specific bilingual communities can be easily changed by providing a translated version to suit your own research settings.

To do this, you will need to create a new file, provide a translation for each interface item and then save it with a specific naming convention. Each step is outlined below.

3.1.1 Creating and Naming your file

To create a new file for your translation, go to the location where the L'ART app is installed, and open the `Versions` folder.

Below is the path you need to follow in order to find it. The path your app is located in depends on whether you installed the app for a single user or for all users (you will have made this choice on installation).

Below is an example of the path when the app is installed for **a single user**:

Sometimes Windows hides the folder `AppData` from view. To make it visible, click on the *View* tab and ticking the box labelled *Hidden items* as follows:

However, if you installed the app **for all users**, you will find the `Versions` folder by following a different path, as below:

To have an interface in the language of your choosing, open the file called `CymEng_Eng_GB.json`. This is the British-English version of the interface built to work with Welsh-English bilinguals.

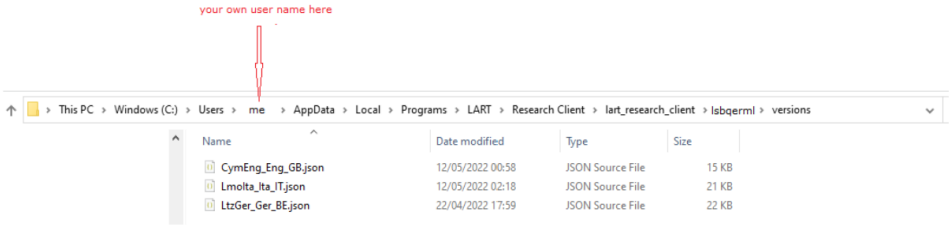
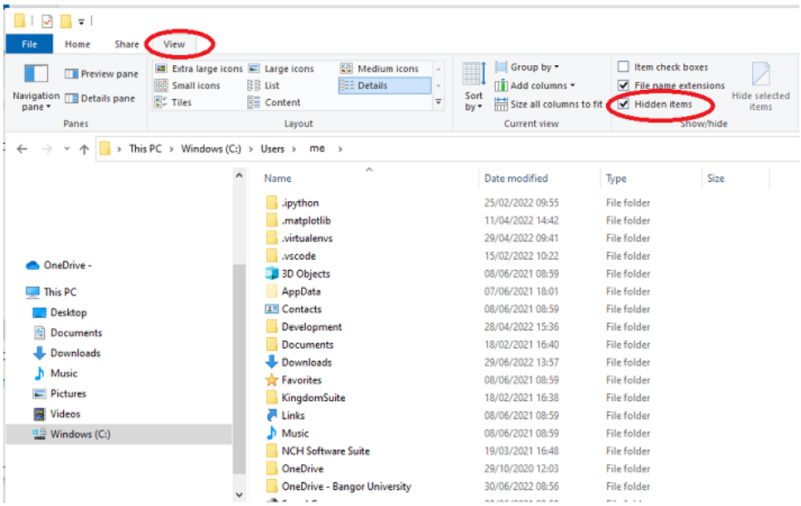


Fig. 3.2: Finding the "versions" folder if you installed L'ART Research Client for a single user



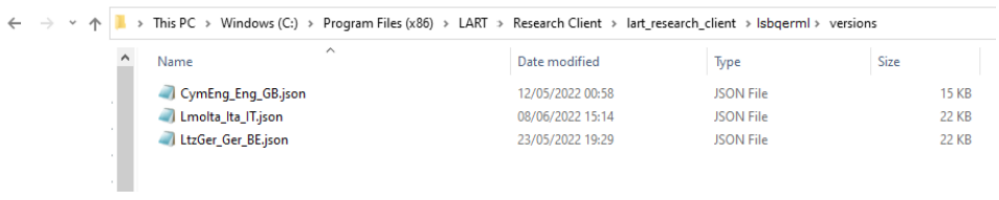


Fig. 3.3: Finding the `Versions` folder if you installed L'ART Research Client for all users

You can open this in **Notepad**, or any text editor of your choice.

Go to *File* and then *Save as*, and save it with a new name that includes the **language** and a **label** for the bilingual community you plan to study.

Note: Naming **must** be done in a specific manner so that the app can find and read the translation you provide.

The naming convention adopted in the L'ART Research Client is based on ISO 639 codes for the languages (a full list is found here) and on ISO 3166-1 alpha-2 codes for the countries (click here for a list), but uses capital letters for the language codes in keeping with CamelCase (see here) as follows:

Description:	Heritage language	Major language	underscore	Language of translation	underscore	Country of interest	Extension
Example:	Lmo	Ita	_	Ita	_	IT	.json
Gloss:	Lombard	Italian		Italian		Italy	

Therefore, the Italian language file to be used for research with the Lombard-Italian bilingual community based in Italy is named: `LmoIta_Ita_IT.json`.

In the instance where you would like to provide a Modern Standard Arabic translation **(Iso code: arb)** to study a bilingual community in Morocco **(ISO code: MA)** whose native languages are Moroccan Arabic **(ary)** and Berber **(ber)** you would label your file **"`BerAry_Arb_MA.json`"**.

Similarly, if you would like to provide a Spanish translation **(Iso code: spa)** to study a bilingual community in Spain **(ISO code: ES)** whose native languages are Galician **(glg)** and Spanish **(spa)**, you would label your file **"`GlgSpa_Spa_ES.json`"**.

3.1.2 Adding your translation

Your newly created file will now be identical to the original British-English file (except for its name)! Now it's time to add your translation. The translation file involves **two** main pieces of information: a **set of labels** and a **language output**. The labels are what the L'ART Research Client needs in order to function, while the language output is what you will see in your interface.

To provide your translated version, you need to highlight each bit of language output and replace it with your translation. Make sure you **do not** change the labels though, otherwise the app will not find your translation and instead, will output the default English version.

First, you will need to provide some basic information about the file. This is the information under the header **"meta"**. With your new file open in a text editor, begin by highlighting the language output for the label `versionId`, as follows:

Then, replace it with the code for your translation. Using our Galician-Spanish example above, this will look as follows:

Now go through each item and provide the relevant information for the header **"meta"**, namely:

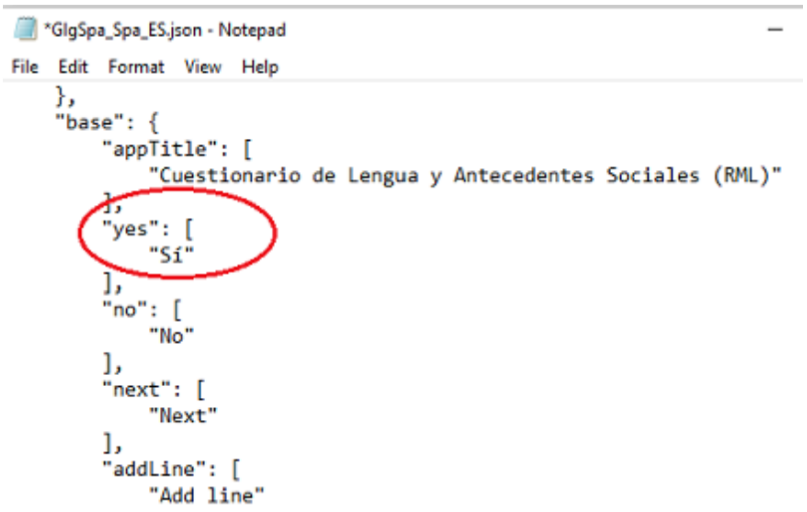
1. The version name
2. The authors' / author's name(s) and email address(es)
3. The date that the file is created.

Once you've completed that, you may begin the translation properly.

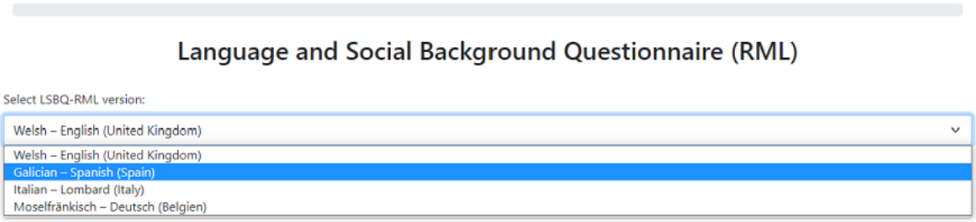

GlgSpa_Spa_ES.json - Notepad
File Edit Format View Help
{
"meta": {
"versionId": "CymEng_Eng_GB",
"versionName": "Welsh – English (United Kingdom)",
"author": "Florian Breit <f.breit@bangor.ac.uk>, Ianto Gruffydd <iant
"date": "2022-04-10"
},
"base": {
"appTitle": [
"Language and Social Background Questionnaire (RML)"
],
"yes": [
"Yes"
],
"no": [
"No"
],
"next": [
"Next"
],
"addLine": [
"Add line"
],
"levelOfEducation1": [
"EQF Level 1",
"No formal qualifications."
],
"levelOfEducation2": [
"EQF Level 2-3",
"GCSE(s), NVQ Levels 1-2."
],
"levelOfEducation3": [
"EQF Level 4",
"A-Level(s), AS-Level(s), NVQ Diploma, HNC, Apprenticeship."
],
Ln 3, Col 36
100%
Windows (CRLF)
UTF-8


*GlgSpa_Spa_ES.json - Notepad
File Edit Format View Help
{
"meta": {
"versionId": "GlgSpa_Spa_ES",
"versionName": "Welsh - English (United Kingdom)",
"author": "Florian Breit <f.breit@bangor.ac.uk>, Ianto Gruffydd <iant
"date": "2022-04-10"
},
"base": {
"appTitle": [
"Language and Social Background Questionnaire (RML)"
],
"yes": [
"Yes"
],
"no": [
"No"
],
"next": [
"Next"
],
"addLine": [
"Add line"
],
"levelOfEducation1": [
"EQF Level 1",
"No formal qualifications."
],
"levelOfEducation2": [
"EQF Level 2-3",
"GCSE(s), NVQ Levels 1-2."
],
"levelOfEducation3": [
"EQF Level 4",
"A-Level(s), AS-Level(s), NVQ Diploma, HNC, Apprenticeship."
],
Ln 27, Col 11
100%
Windows (CRLF)
UTF-8

Ensure that you highlight each language output for each item and provide your translation! For example, under the label *"yes"*, you would replace the output "yes" with **"Sí"**, taking care not to change the label, which **must remain "yes"**, as follows:



And that's it! Once you have replaced all items with your translations, **restart the app** and you will see your Galician-Spanish version, like so:



CHAPTER

FOUR

DEVELOPER GUIDE

Welcome to the L'ART Research Client Developer Guide. This is our "essential reading" for anyone who wants to contribute, enhance, integrate or otherwise modify the Research Client. It will cover everything from contribution guidelines and setting up the development environment to understanding how the code is organised and how to build the app and the documentation from source.

You may also want to have a look at the *API Documentation*, which covers both the Python APIs for the backend and the JavaScript APIs for the frontend.

4.1 Contributing

4.1.1 What do I need to know to help?

If you are looking to helo with a code contribution, our project uses Python 3.10 and Eel for the backend, JavaScript for the frontend logic, and HTML & CSS (specifically, Bootstrap) for the frontend design (i.e., the user interface). If you don't feel ready to make a code contribution yet, no problem! You can also checkout Documentation issues, help by *Localisation and Adding Translations* adding localised/translated versions for the research tasks we have already implemented, or thoroughly test the latest release in the app and submit high-quality bug reports to our issue tracker.

If you are interested in making a code contribution and would like to learn more about the technologies that we use, check out the list of free resources below.

- The official Python Tutorial – Best if you've done some programming before but not necessarily in Python 3.
- Sebastiaan Mathôt's Video Tutorials for Python – Good if you're still relatively unsure about programming and especially Object-Oriented Programming.
- Learn Python 3 The Hard Way – No need to purchase the book, just open the "Contents" menu on the top left and work through the exercise chapters.
- The Modern JavaScript Tutorial – Pretty comprehensive introduction to all things JavaScript.
- Mozilla's JavaScript Guide – Can be worked through quite readily and well-integrated with the relevant API documentation, very good starting point if you've done some programming before but not necessarily (modern) JavaScript.
- Mozilla's JavaScript API Documentation – All the API documentation you will need for JavaScript.
- Mozilla's HTML Guide – Good starting point whether you're new to HTML or just a bit rusty.
- Mozilla's CSS Guide – Definitely much more than you'll need to know for this.
- Official Bootstrap Documentation – As long as you know some HTML and a little bit of CSS, this will cover most of the rest you'll need to know to work on the design of the frontend.

4.1.2 How do I make a contribution?

Never made an open source contribution before? Wondering how contributions work in our project? Here's a quick rundown!

1. Find an issue that you are interested in addressing or a featue that you would like to add.
2. Fork our lart-bangor/research-client repository.

   This means that you will have a copy of the repository under `your-GitHub-username/research-client`.
3. Clone the repository to your local machine using `git clone https://github.com/your-GitHub-username/research-client.git`.
4. Create a new branch for your fix using `git checkout -b branch-name-here`.

   The branch name should ideally follow the schema `active/category/description-of-issue`. For example to make improvements to the backend logic for the LSBQ, this could be `active/lsbq/improve-backend-logic`. For a new tutorial in the documentation it could be `active/docs/basketweaving-tutorial`.

   We generally follow the principle of using the `active` prefix for branches that have active development happening (except the three core branches `main`, `dev`, and `docs`). Once an issue developed on one of these "active" branches has either been merged into one of the core branches or abandoned (e.g. because someone coded themselves into a knot) we rename them with the prefix `obsolete` instead of `active`. This helps us to keep a good picture of what is happening on the repository.

   If you're not sure about the 'category' or it doesn't neatly fit to one sub-part of the project, you can just write `active/general/description-of-issue`.
5. If you need to run the app, make test builds, or build the documentation locally, then install the required dependencies with `pipenv install --dev` (run from inside the project directory, where the file called `Pipfile` is located).

   If you don't need to run the app, make test builds, or build the documentation locally, you can omit this.
6. Make the appropriate changes for the issue you are trying to address or the feature that you want to add.
7. Use `git add insert-paths-of-changed-files-here` to add the file contents of the changed files to the "staging area" git uses to manage the state of the project, also known as the index.
8. Use `git commit -m "Insert a short summary message of the changes made here"` to store the contents of the changes in your "staging area" together with a descriptive message.
9. Push the changes back to your remote repository (`your-GitHub-username/research-client`) by using `git push origin branch-name-here` (using the same branch name you decided on above).
10. Submit a pull request to our upstream repository (lart-bangor/research-client).
11. Title the pull request with a short description of the changes made, and if applicable the issue or bug number associated with your change.

    For example, you could title a pull request "Added additional check to LSBQ data model to resolve issue #1234" or "Add a basketweaving tutorial to the documentation".
12. In the description of the pull request, explain the changes you have made, any issues you think exist with the contribution you're submitting, and ask any questions you have for the maintainer(s).

    It's completely OK if your pull request is not perfect (no pull request is!) — your pull request will be reviewed and the reviewer will be able to help you fix any problems it might have and help you improve it if needed.

    In case you explicitly do not want to be credited for your contribution for any reason you should also mention this in your pull request — otherwise we will assume by default that you are happy for us to add your name and a link to your GitHub profile to the *Contributors* in furture versions of the *User Guide*.

13. Wait for the pull request to be reviewed by a maintainer.
14. Make any changes to the pull request that the reviewing maintainer recommends. They might ask you some questions to clarify some aspect of your pull request, and it's totally okay for you to ask questions during this process as well.
15. Celebrate your success after your pull request is merged!

For a more detailed guide on getting set up to work on the codebase, including if you need to install the dependencies (like **`git`**, **`python`**, etc.) so that you can test run and build the app locally, see our guide on *Setting up the development environment*.

4.1.3 Where can I go for help?

If you need help, you can ask questions on one of our GitHub Discussions sections. We'll be happy to help where we can!

4.1.4 Code of Conduct

We currently have a very simple Code of Conduct:

1. You are responsible for treating everyone on the project with respect and courtesy, regardless of who they are or what their attributes are.
2. If you are the victim of any inappropriate behaviour or comments, we are here for your and will do the best to ensure that any abusers are reprimanded and/or removed, as may be appropriate in the situation.
3. If you are abusive to anyone on the project we reserve the right to reprimand you or remove you from the project, as we may judge appropriate in the situation.
4. Always remember that this is a community we build together .

Note: These contributing guidelines have been adapted from a very neat template provided by Safia Abdalla.

4.2 Setting up the development environment

This article will guide you through the setup of the development environment, aimed primarily at those with relatively little previous experience of software development.

Tip: If you're a seasoned developer, this guide will probably be a bit verbose for your taste, so if you know what you're doing you might just want to install any of the tools listed below (if you don't have them already), fork the repo, and run `pipenv install --dev` in the root of the source tree to get going.

4.2.1 Installing the pre-requirements

To work on the L'ART Research Client codebase, you need to have at least the following:

- git – The version management system we use
- python (version >= 3.10) – The primary programming language of the app
- pipenv – The python package we use for virtual environments and dependency management
- chrome – The browser we use to display the app's user interface

If you want to build the documentation locally, you will also need to have jsdoc installed.

Below are some examples of how you could install this software on various systems, e.g. Windows 10 or 11 if you use `winget`, and Ubuntu Linux (note that the newest releases of Ubuntu will have Python > 3.10 already installed). Please apply these with care. If you're not sure about any of this, it's best to go to the website of the respective software (linked above) and just follow their installation instructions!

```
# Windows Terminal / PowerShell
# Install Google Chrome
winget install -e --id Google.Chrome
# Install Git
winget install -e --id Git.Git
# Install Python 3.10+
winget install -e --id Python.Python.3.10
# Install pipenv
pip install pipenv
# Optional: Install npm and jsdoc (only needed to generate documentation)
winget install -e --id OpenJS.NodeJS.LTS
npm install -g jsdoc
```

```
$ # Ubuntu Linux < 22.04
$ # First make sure the system's packaged and package index are up-to-date
$ sudo apt update && sudo apt upgrade -y
$ # Install Google Chrome
$ sudo apt install libxss1 libappindicator1 libindicator7
$ wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
$ sudo apt install ./google-chrome*.deb
$ rm ./google-chrome*.deb
$ # Install Git
$ sudo apt install git -y
$ # Check your current python version
$ python3 --version
Python 3.9.4
$ # !! You should only run the next command if your python version is below 3.10.x !!
$ sudo apt install software-properties-common
$ sudo add-apt-repository ppa:deadsnakes/ppa
$ sudo apt update
$ sudo apt install python3.10 -y
$ # Now check that you have python3.10 running:
$ python3.10 --version
Python 3.10.5
$ # Install pip and tkinter for Python3.10
$ # (Ff you already had python 3.10 show above, just use "python3" instead of "python3.10
↪")
```

(continues on next page)

(continued from previous page)

```
$ sudo apt install python3.10-pip and python3.10-tk
$ # Install pipenv
$ python3.10 -m pip install pipenv
$ # Optional: Install jsdoc (only needed to generate documentation)
$ sudo apt install npm -y
$ sudo npm install -g jsdoc
```

```
$ # Ubuntu Linux >= 22.04
$ # First make sure the system's packaged and package index are up-to-date
$ sudo apt update && sudo apt upgrade -y
$ # Install Google Chrome
$ sudo apt install libxss1 libappindicator1 libindicator7
$ wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb
$ sudo apt install ./google-chrome*.deb
$ rm ./google-chrome*.deb
$ # Install Git
$ sudo apt install git -y
$ # Check your current python version is >= 3.10.0
$ python3 --version
Python 3.10.5
$ # Install pip and tkinter for Python3
$ sudo apt install python3-pip python3-tk -y
$ # Install pipenv
$ python3 -m pip install pipenv
$ # Optional: Install jsdoc (only needed to generate documentation)
$ sudo apt install npm -y
$ sudo npm install -g jsdoc
```

Following the installation of the above, make sure that both **python** and **pipenv** are on your PATH environment variable. You may need to re-start your terminal, or log out and log back in for this to be the case. To test, just open a new terminal window and type both `python --version` and `pipenv --version`. If this does not work, you need to find out how to add them to the PATH environment variable on your system before proceeding.

Important: Know your machine!

For most of what follows we will assume you have the above software installed and know the correct commands to use. This is especially important for Python, which depending on your installation may go by different names.

If you aren't sure which Python command to use, open a command-line/terminal window and try the following commands in order:

- `py --version`
- `python --version`
- `python3 --version`
- `python3.10 --version`

The first one of these that doesn't give you an error message and prints a Python version that is at least 3.10.0 is the command you should use for everything else.

For simplicity, unless specifying something OS-specific, we will just use `python` throughout the documentation – it's *your responsibility* to adapt accordingly.

If you have the pre-requirements above out of the way, you can follow these steps to get the source code and all dependencies set up.

4.2.2 Get a copy of the source code

These are the steps you need to follow to get a current copy of the sourcecode:

1. Open a terminal (console / command-line prompt)
2. Go to (or make) your prefered directory for development.

   For example `cd C:\Users\florian\Development` (Windows) or `cd /home/florian/development` (Linux). If you don't have a directory you use for software development yet, you can use the **`mkdir`** command to create it, then **`cd`** into it.
3. Clone the repository with `git clone https://github.com/lart-bangor/research-client.git`.

   This will make a local copy of the remote git repository, to which you can then make local changes and which you can sync back and forth with the remote repository (called *pulling* and *pushing*).

   Tip: Fork the repository before cloning it…

   You might want to make a *fork* of our repository on GitHub and work on that fork, so that your own work benefits from the added security of having the version control history in the cloud even if you do not have write permissions to our repository.

   You will also have to make a fork if you want to make a *pull request* later, which is what you would do to have your modifications adopted in our official repository and included in future builds of the L'ART Research Client.

   For more information, check out how to fork a repo in the GitHub Quickstart Guide.

   ---
4. Enter the project's root directory.

   You can do this with the command `cd ./research-client`. If you now type `ls` (Linux) or `dir` (Windows), you should see a list of files including one called `manage.py` – if you see that you know that your code has cloned successfully and you are in the project's root directory.

4.2.3 Set up pipenv and install dependencies

We use **`pipenv`** to manage the environment and dependencies. This makes it very easy to ensure that everyone working on the app can keep their dependencies up-to-date and have the same, stable environment for development.

After cloning the source code repository, there are just two steps to get this all set up. We're assuming you're still in the same terminal session as above, inside the project's root directory (see the last step above).

1. Run `pipenv install --dev`.

   This will set up a new virtual environment (so it doesn't get polluted by any other packages or changes on your system's Python installation, and vice-versa), and then install all the Python packages you need. The `--dev` switch is quite important here, because without it you will be able to run the app from the terminal, but you won't be able to build the app binaries or the documentation for example.
2. Activate the pipenv environment with `pipenv shell`.

   You now have to actually activate the virtual environment, so your terminal knows to use the isolated copy of Python it made for this project instead of the system installation. You activate the environment by typing `pipenv shell` (normally, after this you will see something like `(research-client)` at the start of your command prompt.)

Important: Remember pipenv!

It's important to remember to activate and use **`pipenv`** whenever you start working on the project. If you don't, you'll probably get error messages, and if you then just use regular `pip` to try and resolve these you'll mess up your system-wide installation and run the risk of introducing new dependencies that can break the code, without other people being able to later see what these dependencies were. It might also prevent you from being able to build the binaries from the source.

So, every time you open a terminal to work on the project, remember to use `pipenv shell` first. Every time you install a package, remember to use `pipenv install <pkgname>` or `pipenv install <nobr>--dev</nobr> <pkgname>` (if the package is only needed for development, but not for the version the end-user gets).

4.2.4 Running the app from the source

Now let's test that things are working as they should. Open a terminal and go to the directory to which you've cloned the source code, e.g. `C:\Users\florian\Development\research-client` (Windows) or `/home/florian/development/research-client` (Linux). You know that you are in the right directory if you type `ls` (Linux) or `dir` (Windows) and the list shown contains a file named `manage.py`.

Now just type `python manage.py run` in your terminal and hit `Enter`. If you get an error, something in the above steps probably went wrong — check which of the steps the error message seems to relate to and try again from there. If you see the app's main window and some text on the terminal telling you that it is running, then you should be good to go.

Tip: Use a dedicated code editor…

If you use VS Code as your editor, you can tell it to automatically activate the **`pipenv`** environment when you open your source code.

Just install the Python extension in VS Code. Then press `Ctrl+Shift+P` and type *Python: Select Interpreter*, then select the one showing "(PipEnv)" in parentheses at the end.

Similar extensions are available for most other editors and IDEs, it's worth consulting their documentation on this.

4.2.5 Bonus: Consider using a specialised source code editor

If you have only written a few lines of Python, HTML, or JavaScript here and there in the past, chances are that you've just used a general purpose text editor in the past, such as *notepad* or *gedit*.

We recommend that you consider a modern specialised source code editor or IDE (Integrated Development Environment) instead. The extra features they offer, such as running terminal commands from within the editor, integrating with git, showing type-error hints in your code, etc. will pay of quickly on a codebase like this.

Some free options you might want to consider:

- VS Code: Lightweight, responsive, platform-independent. Used by most people on our team.
- Geany: Super-lightweigt, responsive, platform-independent. A popular choice for those that don't want to run just a 'free' Microsoft product or otherwise don't like VS Code.
- Spyder: Medium-weight, aimed primarily at scientific computing, a bit like RStudio. Worth considering if you want to also run data analysis in Python.

- PyCharm: A more heavy-weight IDE with many features, quite popular and probably a bit more than what is needed. It's commercial software, but there is a free community version you can download, and if you're an academic or student you can get a free full license.
- vim: Lightweight, super-fast, very powerful terminal-based editor. If you prefer not to use a graphical user interface and stay on the command line this is probably for you, but the learning curve is rather steep.

4.3 The *manage.py* utility

The project includes its own little script to simplify many common project management tasks. From inside the project directory, you can run the following commands:

```
python manage.py -h          # show the available commands
python manage.py run         # runs the app from the source files
python manage.py debug       # runs the app in debug mode
python manage.py clean       # cleans up your project directory
python manage.py build       # builds an executable with PyInstaller
python manage.py docs        # build the documentation
```

4.4 Building from source

This guide will describe how to build the app for distribution with PyInstaller and Inno Setup (if you're building on Windows). We assume that you have successfully followed the steps to *Setting up the development environment* and *Running the app from the source* works already.

4.4.1 Additional build dependencies

The L'ART Research Client app is built with PyInstaller, and Inno Setup is used on Windows to package it up as an executable installer.

PyInstaller

PyInstaller will already have been installed when you have installed the development dependencies with `pipenv install --dev` while *Setting up the development environment*. If you get an error saying that PyInstaller could not be found, just run that command again.

Inno Setup

Inno Setup is only needed if you're building on Windows — so if you're on Linux or MacOS you can just ignore anything to do with Inno Setup.

On Windows, Inno Setup needs to be installed on your system or the build process will fail. Follow these simple steps to install it:

1. Download the latest version of Inno Setup from **`https://jrsoftware.org/isdl.php`_**.
2. Run the Inno Setup installer on your system to install it.
3. Check whether the command **`iscc`** is on your system's `Path` by opening a terminal window (`Windows+R`, type `cmd`, hit `Enter`) and then entering the command `iscc` followed by `Enter`.

4. If you see soem text starting with something like "Inno Setup 6 Command-Line Compiler" followed by instructions, you're good to know and can skip to the next section. In the (very likely) event that you get an error instead, continue with the next step here.
5. Locate `ISCC.exe` and note the path to the directory where it is located. This is probably `C:\Program Files\Inno Setup 6` or `C:\Program Files (x86)\Inno Setup 6`.
6. Either type "environment variables" in the Start search box and open the "Edit the system environment variables" shortcut that shows up, or go to *Settings -> System -> Advanced system settings*.
7. On the bottom of the dialog box, click on *Environment Variables...*.
8. Highlight the variable called `Path` in the top half of the dialogue window, then click on *Edit*.
9. Click on *New* and add the path to the directory where Inno Setup (`ISCC.exe` from before) is installed, e.g. `C:\Program Files (x86)\Inno Setup 6`.
10. Click *OK* repeatedly until all the dialogue windows are closed.
11. Start a *new* terminal window (it will not work in any terminal windows that were opened before you edited the `Path` environment variable) and try running `iscc` again — it should work now, meaning you're ready to build the app on Windows (if it still doesn't work, you probably entered the wrong path two steps earlier).

4.4.2 Building the app and the installer

Building the app is super simple. Just go to the folder containing the `manage.py` file, make sure you're running in the **`pipenv`** shell (if you're not sure, just run `pipenv shell` again), and then run the command `py ./manage.py build` (on Windows) or `python3.10 manage.py build` (on Linux and MacOS).

The folder `./build` will contain all the build artificats and direct outputs from PyInstaller.

The folder `./dist` will contain the distributables for the app, in a subfolder named after the system on which they were built. For example on Linux, there will be a tarball (`*.tar.gz`) in `./dist/linux`, while on Windows there will be both a ZIP archive and an executable (`.exe`) installer in `./dist/win64`, which can be used to install the app.

4.4.3 Building the documentation

The documentation is built automatically on Read the Docs whenever a pull request, push or merge succeeds on the `docs` branch of the repository. Even so, if you're updating the documentation (even in the inline documentation in the Python and JavaScript files) it might be desirable to build it locally to make sure any changes are reflected as they should be and nothing breaks.

Additional documentation dependencies

To build the documentation, you need to additionally install jsdoc, as shown as an optional step in *Installing the prerequirements*.

`jsdoc` is used to extract documentation from within the JavaScript files that provide the app's APIs in the frontend.

You can check whether **`jsdoc`** works by opening a terminal and typing `jsdoc`. If it is installed correctly and available on your `Path`, it should print something like "There are no input files to process." — otherwise you will need to install it and make sure it is available on the `Path` before you can build the documentation.

Building the documentation

Building the documentation is just as simple as building the app. Like with building the app, make sure you're in the directory containing the file `manage.py` and that you're in the **`pipenv`** shell (any doubt, just run `pipenv shell`). Then just run the command `py ./manage.py docs` (on Windows) or `python3.10 manage.py docs` (on Linux and MacOS).

The folder `./dist/docs/html` will contain the HTML version of the documentation (we do not currently build the latex/PDF version offline, as this has too many dependencies and quirks to work reliably from one person to the next).

4.4.4 Cleaning up after yourself

Just like with your bedroom, it's important to keep your development environment tidy. So once you've completed your builds and inspected that everything is as it should be, you probably want to clean up all the artifacts, local documentation and distributables generated by the build process…

Just run `py ./manage.py clean` (on Windows) or `python3.10 manage.py clean` (on Linux or MacOS), and the *manage.py* utility will make everything nice and tidy again .

4.4.5 Known issues with building

- Building fails with Python version 3.10.0 due a bug in Python that affects PyInstaller (issue). If your Python version is 3.10.0 then update to 3.10.1 or later (but not 3.11.x, for which nothing has been tested, … yet).

4.5 Roadmap

The below roadmap should give an idea of what features and enhancements to the L'ART Research Client that we envision and plan to implement in the future. It also gives some idea of the various stages at which we would like to implement these — though these are not necessarily set in stone and may be adjusted as our available development time (and demand for the features) dictates.

The main purpose of this roadmap is for us as a development team to be able to visualise where we are heading, what we want to achieve, and how we might go about that, but we explicitly choose to make it public here also so that our users and collaborators can get a sense for where things might be heading — and of course have a chance to influence that.

If you would like to dicuss any of the planned enhancements or additional ideas you have, we would very much welcome your input via the Ideas section on GitHub Discussions.

Milestone	Research Tasks	Usability	Data Management	Integrations
1.0.0	• Participant Notes	• Modularisation of task versions	• Pydantic backend • JSON Schemas	
1.1.0	• SPIN Test			
1.2.0			• More export options • In-app data synching	• FastAPI
Future	• HeLeX and/or LSBQ-H • LSBQ for children • A simple BG Questionnaire	• Option to run online		

CHAPTER

FIVE

API DOCUMENTATION

5.1 Backend API (Python)

agt	Package implementing the Audio Guise Task (AGT).
app	LART Research Client App.
atolc	Data structures for the AToL Questionnaire (RML).
booteel	Utilities to work with Python Eel and Bootstrap.
conclusion	Conclusion screen for end of task series.
config	Configuration handler for L'ART Research Client.
consent	Informed consent from user.
datavalidator	Simple data validation tools.
lsbq	Package implementing the Language and Social Background Questionnaire.
memorygame	Package implementing the Memory Game for the LART Research Client.
settings	Package implementing the Settings UI for the LART Research Client.
utils	Utility functions for the LART Research Client app.

5.1.1 research_client.agt

Package implementing the Audio Guise Task (AGT).

Functions

expose_to_eel()	Expose the AGT API to Python Eel.

expose_to_eel()

Expose the AGT API to Python Eel.

Modules

research_client.agt.dataschema	Data schema implementing the AGT.
research_client.agt.eel	Exposes the AGT to Python Eel.
research_client.agt.patterns	Additional validation patterns for AGT.
research_client.agt.versions	Version implementations and translations for the AGT.

research_client.agt.dataschema

Data schema implementing the AGT.

Classes

Response	Class for representing the data of an AGT questionnaire response.

class Response

Bases: *DataSchema*

Class for representing the data of an AGT questionnaire response.

__init__(*id_=None*)

Instantiates a new LSBQ-RML response object.

Parameters
id_ (*Optional[str]*)

generate_trial_order(*fillers=None*, *speakers=None*, *languages=None*, *sep='_'*)

Produce a pseudo-randomised AGT presentation order.

Given four speakers, four fillers, and two languages, produce a presentation order for Matched Guise Task, based on the following grid:

Speaker	Language	Example
F1	Either	Filler 1
S1	L1	Speaker 1, Language 1
S2	L2	Speaker 2, Language 2
F2	Either	Filler 2
S3	L2	Speaker 3, Language 2
S4	L1	Speaker 4, Language 1
F3	Either	Filler 3
S1	L2	Speaker 1, Language 2
S2	L1	Speaker 2, Language 1
F4	Either	Filler 4
S3	L1	Speaker 3, Language 1
S4	L2	Speaker 4, Language 2

The function randomises:

(a) the order of the fillers (regardless of filler language),

(b) the order in which speakers are presented (distance kept constant)

(c) whether L1 or L2 are presented first (keeping alternation constant)

Return type
tuple[str, ...]

Parameters

- **fillers** (*Optional[list[str]]*)
- **speakers** (*Optional[list[str]]*)
- **languages** (*Optional[list[str]]*)
- **sep** (*str*)

getratings(*trial*)

Get the ratings for the trial labelled by *trial*.

Return type
dict[str, float]

Parameters
trial (*str*)

setratings(*trial*, *ratings*)

Set the ratings for the trial labelled by *trial*.

Parameters

- **trial** (*str*)
- **ratings** (*dict[str, float]*)

```
__schema = {'f1_ratings': {'ambitious': {'constraint': (0, 100), 'type_': <class
'float'>, 'typedesc': 'rating of ambitious'}, 'amusing': {'constraint': (0, 100),
'type_': <class 'float'>, 'typedesc': 'rating of amusing'}, 'attractive':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
attractive'}, 'competent': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of competent'}, 'cool': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of cool'}, 'educated': {'constraint': (0,
100), 'type_': <class 'float'>, 'typedesc': 'rating of educated'}, 'friendly':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
friendly'}, 'honest': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of honest'}, 'ignorant': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of ignorant'}, 'influential': {'constraint':
(0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of influential'},
'intelligent': {'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc':
'rating of intelligent'}, 'international': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of international'}, 'likeable':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
likeable'}, 'natural': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of natural'}, 'open-minded': {'constraint': (0, 100),
'type_': <class 'float'>, 'typedesc': 'rating of open-minded'}, 'polite':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
polite'}, 'pretentious': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of pretentious'}, 'trustworthy': {'constraint': (0, 100),
'type_': <class 'float'>, 'typedesc': 'rating of trustworthy'}}, 'f2_ratings':
{'ambitious': {'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc':
'rating of ambitious'}, 'amusing': {'constraint': (0, 100), 'type_': <class
'float'>, 'typedesc': 'rating of amusing'}, 'attractive': {'constraint': (0,
100), 'type_': <class 'float'>, 'typedesc': 'rating of attractive'}, 'competent':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
competent'}, 'cool': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of cool'}, 'educated': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of educated'}, 'friendly': {'constraint':
(0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of friendly'}, 'honest':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
honest'}, 'ignorant': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of ignorant'}, 'influential': {'constraint': (0, 100),
'type_': <class 'float'>, 'typedesc': 'rating of influential'}, 'intelligent':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
intelligent'}, 'international': {'constraint': (0, 100), 'type_': <class
'float'>, 'typedesc': 'rating of international'}, 'likeable': {'constraint': (0,
100), 'type_': <class 'float'>, 'typedesc': 'rating of likeable'}, 'natural':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
natural'}, 'open-minded': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of open-minded'}, 'polite': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of polite'}, 'pretentious': {'constraint':
(0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of pretentious'},
'trustworthy': {'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc':
'rating of trustworthy'}}, 'f3_ratings': {'ambitious': {'constraint': (0, 100),
'type_': <class 'float'>, 'typedesc': 'rating of ambitious'}, 'amusing':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
amusing'}, 'attractive': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of attractive'}, 'competent': {'constraint': (0, 100),
'type_': <class 'float'>, 'typedesc': 'rating of competent'}, 'cool':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of cool'},
'educated': {'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc':
'rating of educated'}, 'friendly': {'constraint': (0, 100), 'type_': <class
'float'>, 'typedesc': 'rating of friendly'}, 'honest': {'constraint': (0, 100),
'type_': <class 'float'>, 'typedesc': 'rating of honest'}, 'ignorant':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
ignorant'}, 'influential': {'constraint': (0, 100), 'type_': <class 'float'>,
```

Type: `dict`

research_client.agt.eel

Exposes the AGT to Python Eel.

Functions

discard(instid)	Discards a Response.
end(instid[, data])	Redirect participant in right sequence after AGT end screen.
get_traits()	Return the list of AGT stimuli.
getmissing(instid)	Gets a list of missing fields.
getversions()	Retrieves the available versions of the AGT.
init(data)	Initialises a new AGT Response.
iscomplete(instid)	Checks whether a Response is complete.
load_version(instid, sections)	Load specified sections of an AGT version implementation.
setratings(instid, data)	Adds the ratings for a given trial and redirects to the next trial.
store(instid)	Submits a (complete) Response for long-term storage.

_expose(*func*)

Wraps, renames and exposes a function to eel.

Return type
`TypeVar(F, bound= Callable[..., Any])`

Parameters
func (*F*)

_getinstance(*instid*)

Return type
Response

Parameters
instid (*str*)

_getnexttrial(*instid*, *current_trial*)

Return the trial following *current_trial* on an AGT Response.

Return type
`Optional[str]`

Parameters

- **instid** (*str*)
- **current_trial** (*str*)

_handleexception(*exc*)

Passes exception to exceptionhandler if defined, otherwise continues raising.

Return type
None

Parameters
exc (*Exception*)

discard(*instid*)

Discards a Response.

Return type
bool

Parameters
instid (*str*)

end(*instid*, *data=None*)

Redirect participant in right sequence after AGT end screen.

Return type
str

Parameters

- **instid** (*str*)
- **data** (*Optional[dict[str, str]]*)

get_traits()

Return the list of AGT stimuli.

getmissing(*instid*)

Gets a list of missing fields.

Return type
list[str]

Parameters
instid (*str*)

getversions()

Retrieves the available versions of the AGT.

Return type
dict[str, str]

init(*data*)

Initialises a new AGT Response.

Return type
str

Parameters
data (*dict[str, Any]*)

iscomplete(*instid*)

Checks whether a Response is complete.

Return type
bool

Parameters
instid (*str*)

load_version(*instid*, *sections*)

Load specified sections of an AGT version implementation.

Return type
dict[str, dict[str, Any]]

Parameters

- **instid** (*str*)
- **sections** (*list[str]*)

setratings(*instid*, *data*)

Adds the ratings for a given trial and redirects to the next trial.

Return type
None

Parameters

- **instid** (*str*)
- **data** (*dict[str, str]*)

store(*instid*)

Submits a (complete) Response for long-term storage.

Return type
bool

Parameters
instid (*str*)

F = TypeVar(F, bound=Callable)

Type: TypeVar

Invariant TypeVar bound to typing.Callable[..., typing.Any].

research_client.agt.patterns

Additional validation patterns for AGT.

research_client.agt.versions

Version implementations and translations for the AGT.

_get_versions()

Loads all available AGT versions into memory.

Return type
dict[str, dict[str, Any]]

5.1.2 research_client.app

LART Research Client App.

An app to collect survey-type data for research on regional and minority languages, developed by the Language Attitudes Research Team at Bangor University.

Functions

atol_rating(data)	Retrieve atol rating and print to screen.
close(page, opensockets)	Callback when an app socket is closed.
export_data_backup()	Non-blocking eel wrapper for the app's `export_backup()` function.
main()	App main function called on app launch.
shutdown([sig, frame])	Shut down the app.

atol_rating(*data*)

Retrieve atol rating and print to screen.

Parameters
data (*dict[Any, Any]*)

close(*page*, *opensockets*)

Callback when an app socket is closed.

Parameters

- **page** (*str*)
- **opensockets** (*list[Any]*)

export_data_backup()

Non-blocking eel wrapper for the app's `export_backup()` function.

main()

App main function called on app launch.

shutdown(*sig=None*, *frame=None*)

Shut down the app.

5.1.3 research_client.atolc

Data structures for the AToL Questionnaire (RML).

Functions

alphabetise(dictionary)	Return a dict in alphabetised order.
arrange_data(data)	Orders data so that meta info is consistent with other tasks in the app.
arrange_order(dict, source)	Records order in which traits were presented for a given trial, then orders them alphabetically for write readability
atol_c_get_items(version)	Get label pairs for each AToL item depending on language selection.
atol_end(data)	Redirect participant in right sequence after AToL-C end screen.
fetch_location(source_file, version)	Finds which html page to load next.
get_id(dict)	Returns a participant ID
grab_atol_ratings(myData, source, ...)	Does the same as init_atol, but for ratings
init_atol(myData)	Retrieves initial info from index.html and prints to file & to terminal.
key_list(dic)	Return the keys of a dictionary.
randomize(dictionary)	Return a dict in randomized order.

Classes

Response	Class for representing the data of an LSBQ-RML questionnaire response.

class Response

Bases: *DataSchema*

Class for representing the data of an LSBQ-RML questionnaire response.

__init__(*id_=None*)

Instantiates a new LSBQ-RML response object.

Parameters
id_ (*Optional[int]*)

setmeta(*data*)

Sets the metadata for the response.

Return type
None

Parameters
data (*dict[str, Any]*)

```
__schema = {'id': {'constraint': (0, 9223372036854775807), 'type_': <class
'int'>, 'typedesc': 'LSBQ-RML Response ID'}, 'language1': {'ambiguous':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
ambiguous'}, 'appealing': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of appealing'}, 'ballsy': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of ballsy'}, 'beautiful': {'constraint': (0,
100), 'type_': <class 'float'>, 'typedesc': 'rating of beautiful'}, 'elegant':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
elegant'}, 'flowing': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of flowing'}, 'fluent': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of fluent'}, 'graceful': {'constraint': (0,
100), 'type_': <class 'float'>, 'typedesc': 'rating of graceful'}, 'logical':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
logical'}, 'pleasant': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of pleasant'}, 'precise': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of precise'}, 'round': {'constraint': (0,
100), 'type_': <class 'float'>, 'typedesc': 'rating of round'}, 'smooth':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
smooth'}, 'soft': {'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc':
'rating of soft'}, 'structured': {'constraint': (0, 100), 'type_': <class
'float'>, 'typedesc': 'rating of structured'}, 'systematic': {'constraint': (0,
100), 'type_': <class 'float'>, 'typedesc': 'rating of systematic'}}, 'language2':
{'ambiguous': {'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc':
'rating of ambiguous'}, 'appealing': {'constraint': (0, 100), 'type_': <class
'float'>, 'typedesc': 'rating of appealing'}, 'ballsy': {'constraint': (0, 100),
'type_': <class 'float'>, 'typedesc': 'rating of ballsy'}, 'beautiful':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
beautiful'}, 'elegant': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of elegant'}, 'flowing': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of flowing'}, 'fluent': {'constraint': (0,
100), 'type_': <class 'float'>, 'typedesc': 'rating of fluent'}, 'graceful':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
graceful'}, 'logical': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of logical'}, 'pleasant': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of pleasant'}, 'precise': {'constraint': (0,
100), 'type_': <class 'float'>, 'typedesc': 'rating of precise'}, 'round':
{'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of
round'}, 'smooth': {'constraint': (0, 100), 'type_': <class 'float'>, 'typedesc':
'rating of smooth'}, 'soft': {'constraint': (0, 100), 'type_': <class 'float'>,
'typedesc': 'rating of soft'}, 'structured': {'constraint': (0, 100), 'type_':
<class 'float'>, 'typedesc': 'rating of structured'}, 'systematic': {'constraint':
(0, 100), 'type_': <class 'float'>, 'typedesc': 'rating of systematic'}}, 'meta':
{'consent': {'constraint': ({'true', True, 'on', '1', 'yes'}, {False, 'off', '0',
'false', 'no'}), 'type_': typing.Union[tuple[typing.Iterable[typing.Any],
typing.Iterable[typing.Any]], list[typing.Iterable[typing.Any]]], 'typedesc':
'consent confirmation'}, 'date': {'constraint':
'[0-9]{1,4}\\-(0?[1-9]|1[0-2])\\-(0?[1-9]|[12][0-9]|3[01])', 'type_': <class
'str'>, 'typedesc': 'current date'}, 'participant_id': {'constraint':
'[A-Za-z0-9]{3,10}', 'type_': <class 'str'>, 'typedesc': 'Participant ID'},
'research_location': {'constraint': "[\\w,' \\(\\)\\.\\-]{1,50}", 'type_': <class
'str'>, 'typedesc': 'location name'}, 'researcher_id': {'constraint':
'[A-Za-z0-9]{3,10}', 'type_': <class 'str'>, 'typedesc': 'Researcher ID'},
'version': {'constraint': '\\w{13,17}', 'type_': <class 'str'>, 'typedesc':
'LSBQ-RML version identifier'}}}
```

Type: dict

_atol_getversions()

Retrieves the available versions of the AToL.

Return type
dict[str, str]

alphabetise(*dictionary*)

Return a dict in alphabetised order.

Return type
dict[str, Any]

Parameters
dictionary (*dict[str, Any]*)

arrange_data(*data*)

Orders data so that meta info is consistent with other tasks in the app.

arrange_order(*dict*, *source*)

Records order in which traits were presented for a given trial, then orders them alphabetically for write readability

atol_c_get_items(*version*)

Get label pairs for each AToL item depending on language selection.

Return type
Optional[dict[str, tuple[str, str]]]

Parameters
version (*str*)

atol_end(*data*)

Redirect participant in right sequence after AToL-C end screen.

Return type
str

Parameters
data (*dict[str, str]*)

fetch_location(*source_file*, *version*)

Finds which html page to load next.

Return type
Optional[str]

Parameters

- **source_file** (*str*)
- **version** (*str*)

get_id(*dict*)

Returns a participant ID

grab_atol_ratings(*myData*, *source*, *version_id*, *partId*, *versN*)

Does the same as init_atol, but for ratings

Parameters

- **myData** (*dict[Any, Any]*)

- **source** (*str*)
- **version_id** (*str*)
- **partId** (*str*)
- **versN** (*str*)

init_atol(*myData*)

Retrieves initial info from index.html and prints to file & to terminal.

Return type
None

Parameters
myData (*dict[str, str]*)

key_list(*dic*)

Return the keys of a dictionary.

Return type
Iterable[Any]

Parameters
dic (*dict[str, Any]*)

randomize(*dictionary*)

Return a dict in randomized order.

Return type
dict[str, Any]

Parameters
dictionary (*dict[str, Any]*)

Modules

research_client.atolc.patterns	Additional validation patterns for AToL-C.
research_client.atolc.testPyt	
research_client.atolc.versions	dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's (key, value) pairs dict(iterable) -> new dictionary initialized as if via: d = {} for k, v in iterable: d[k] = v dict(**kwargs) -> new dictionary initialized with the name=value pairs in the keyword argument list. For example: dict(one=1, two=2).

research_client.atolc.patterns

Additional validation patterns for AToL-C.

research_client.atolc.testPyt

Functions

atol_test()	Get label pairs for each AToL item depending on language selection.

atol_test()

Get label pairs for each AToL item depending on language selection.

research_client.atolc.versions

versions = {'CymEng_Cym_GB': {'adjectives': {'ambiguity': ['diamwys', 'amwys'], 'angularity': ['onglog', 'crwn'], 'appeal': ['apelgar', 'atgas'], 'beauty': ['prydferth', 'hyll'], 'elegance': ['anghywrain', 'cywrain'], 'flow': ['llifo', 'swta'], 'fluency': ['toredig', 'rhugl'], 'grace': ['trwsgl', 'gosgeiddig'], 'harshness': ['llym', 'meddal'], 'logic': ['rhesymegol', 'afresymegol'], 'pleasure': ['dymunol', 'annymunol'], 'precision': ['manwl', 'amhendant'], 'sistem': ['trefnus', 'di-drefn'], 'smoothness': ['llyfn', 'cryg'], 'structure': ['distrwythur', 'strwythuredig']}, 'intface_info': {'atol_header': 'Holiadur AToL', 'btn_text': 'Nesaf', 'instruction': 'Defnyddiwch y llythrydd i ateb', 'language': 'Saesneg', 'language_header': 'Mae'r Saesneg yn...', 'next_task': 'Cliciwch Parhau pan fyddwch yn barod', 'rml': 'Cymraeg', 'rml_header': 'Mae'r Gymraeg yn...', 'thank_you': "Diolch am lenwi'r holiadur!", 'title': 'Holiadur Iaith'}, 'meta': {'author': 'Ianto Gryffudd <ianto.gryffudd@bangor.ac.uk>, Marco Tamburelli <m.tamburelli@bangor.ac.uk>', 'date': '2023-01-30', 'versionId': 'CymEng_Cym_GB', 'versionName': 'Cymraeg - Saesneg (Y Deyrnas Unedig)', 'versionNumber': '0.3.4'}}, 'CymEng_Eng_GB': {'adjectives': {'ambiguity': ['unambiguous', 'ambiguous'], 'angularity': ['angular', 'round'], 'appeal': ['appealing', 'abhorrent'], 'beauty': ['beautiful', 'ugly'], 'elegance': ['inelegant', 'elegant'], 'flow': ['flowing', 'abrupt'], 'fluency': ['choppy', 'fluent'], 'grace': ['clumsy', 'graceful'], 'harshness': ['harsh', 'soft'], 'logic': ['logical', 'illogical'], 'pleasure': ['pleasant', 'unpleasant'], 'precision': ['precise', 'vague'], 'sistem': ['systematic', 'unsystematic'], 'smoothness': ['smooth', 'raspy'], 'structure': ['unstructured', 'structured']}, 'intface_info': {'atol_header': 'AToL Questionnaire', 'btn_text': 'Next', 'instruction': 'Please move the slider to record your choice.', 'language': 'English', 'language_header': 'The English language is...', 'next_task': "Please click Continue when you're ready", 'rml': 'Welsh', 'rml_header': 'The Welsh language is...', 'thank_you': 'Thank you for completing the questionnaire!', 'title': 'Language Questionnaire'}, 'meta': {'author': 'Marco Tamburelli <m.tamburelli@bangor.ac.uk>', 'date': '2023-01-16', 'versionId': 'CymEng_Eng_GB', 'versionName': 'English - Welsh (United Kingdom)', 'versionNumber': '0.3.4'}}, 'EngZzz_Eng_GB': {'adjectives': {'ambiguity': ['unambiguous', 'ambiguous'], 'angularity': ['angular', 'round'], 'appeal': ['appealing', 'abhorrent'], 'beauty': ['beautiful', 'ugly'], 'elegance': ['inelegant', 'elegant'], 'flow': ['flowing', 'abrupt'], 'fluency': ['choppy', 'fluent'], 'grace': ['clumsy', 'graceful'], 'harshness': ['harsh', 'soft'], 'logic': ['logical', 'illogical'], 'pleasure': ['pleasant', 'unpleasant'], 'precision': ['precise', 'vague'], 'sistem': ['systematic', 'unsystematic'], 'smoothness': ['smooth', 'raspy'], 'structure': ['unstructured', 'structured']}, 'intface_info': {'atol_header': 'AToL Questionnaire', 'btn_text': 'Next', 'instruction': 'Please move the slider to record your choice.', 'language': 'English', 'language_header': 'The English language is...', 'next_task': "Please click Continue when you're ready", 'rml': '[ENTER language name HERE]', 'rml_header': '[ENTER language name HERE] language is...', 'thank_you': 'Thank you for completing the questionnaire!', 'title': 'Language Questionnaire'}, 'meta': {'author': 'Marco Tamburelli <m.tamburelli@bangor.ac.uk>', 'date': '2023-01-16', 'versionId': 'EngZzz_Eng_GB', 'versionName': 'English - generic (United Kingdom)', 'versionNumber': '0.3.4'}}, 'LmoIta_Ita_IT': {'adjectives': {'ambiguity': ['chiaro', 'ambiguo'], 'angularity': ['spigoloso', 'arrotondato'], 'appeal': ['attraente', 'ripugnante'], 'beauty': ['bello', 'brutto'], 'elegance': ['non elegante', 'elegante'], 'flow': ['fluido', 'brusco'], 'fluency': ['frammentato', 'scorrevole'], 'grace': ['goffo', 'aggraziato'], 'harshness': ['duro', 'morbido'], 'logic': ['logico', 'illogico'], 'pleasure': ['piacevole', 'spiacevole'], 'precision': ['preciso', 'vago'], 'sistem': ['sistematico', 'non sistematico'], 'smoothness': ['liscio', 'ruvido'], 'structure': ['non strutturato', 'strutturato']}, 'intface_info': {'atol_header': 'Questionario sulle varietà linguistiche', 'btn_text': 'Avanti', 'instruction': 'Si prega di spostare il cursore per registrare la propria scelta.', 'language': 'Italiano', 'language_header': "A mio avviso, l'italiano è...", 'next_task': 'Si prega di cliccare Continua appena possibile', 'rml': 'lombardo', 'rml_header': 'A mio avviso, il lombardo è...', 'thank_you': 'Grazie per aver completato il questionario!', 'title': 'AToL - RML'}, 'meta': {'author': 'Marco Tamburelli <m.tamburelli@bangor.ac.uk>', 'date': '2023-01-30', 'versionId': 'LmoIta_Ita_IT', 'versionName': 'Italian - Lombard (Italy)', 'versionNumber': '0.3.4'}}, 'LtzGer_Ger_BE': {'adjectives': {'ambiguity':

dict() -> new empty dictionary dict(mapping) -> new dictionary initialized from a mapping object's

(key, value) pairs

dict(iterable) -> new dictionary initialized as if via:
d = {} for k, v in iterable:

d[k] = v

dict(kwargs) -> new dictionary initialized with the name=value pairs**
in the keyword argument list. For example: dict(one=1, two=2)

5.1.4 research_client.booteel

Utilities to work with Python Eel and Bootstrap.

Functions

buildquery(params)	Build a URL query string based on a set of key-value pairs of parameters.
displayexception(exc)	Display an exception as a dismissable client-side bootstrap modal.
modal(title, body[, options, primary, ...])	Display a client-side modal via bootstrap.
setlocation(location)	Direct the user to a new address.
setloglevel(level)	Set the log level for the booteel module in both Python and JavaScript.

_booteel_handlemodal(*modal_id*, *choice*)

Parameters

- **modal_id** (*str*)
- **choice** (*str*)

_booteel_log(*level*, *message*, *args*)

Expose logging interface to booteel.js.

Parameters

- **level** (*int*)
- **message** (*str*)
- **args** (*list[Any]*)

_booteel_logger_getlevel()

Grant access to the module's loglevel to booteel.js.

buildquery(*params*)

Build a URL query string based on a set of key-value pairs of parameters.

Return type
str

Parameters
params (*dict[str, str]*)

displayexception(*exc*)

Display an exception as a dismissable client-side bootstrap modal.

Parameters
exc (*Exception*)

modal(*title*, *body*, *options=None*, *primary='ok'*, *dismissable=True*, *callback=None*, *icon=None*)

Display a client-side modal via bootstrap.

Return type
str

Parameters

- **title** (*str*)
- **body** (*str*)
- **options** (*Optional[dict[str, str]]*)
- **primary** (*str*)
- **dismissable** (*bool*)
- **callback** (*Optional[Callable[[str, str], bool]]*)
- **icon** (*Optional[str]*)

setlocation(*location*)

Direct the user to a new address.

Parameters
location (*str*)

setloglevel(*level*)

Set the log level for the booteel module in both Python and JavaScript.

Parameters
level (*int*)

5.1.5 research_client.conclusion

Conclusion screen for end of task series.

This module implements a configurable screen for the conclusion of a task or series of tasks in the Research Client app.

The functionality provided does not implement any data models, but simply provides text in different language versions which can be used to indicate to a user that they have reached the end of their assigned tasks in the app.

This is nicer than just sending the user straight back to the home screen of the app, which might not provide sufficient closure to leave the user confident that everything was successfully concluded.

Functions

expose_to_eel()	Expose the Conclusion screen API to Python Eel.

expose_to_eel()

Expose the Conclusion screen API to Python Eel.

Modules

research_client.conclusion.eel	Exposes the Conclusion screen to Python Eel.
research_client.conclusion.versions	Version implementations and translations for the conclusion screen.

research_client.conclusion.eel

Exposes the Conclusion screen to Python Eel.

Functions

end(version_id)	Redirect following sequence logic after conclusion screen ends.
getversions()	Retrieves the available versions of the Conclusion screen.
init(data)	Initialises a new Conclusion screen instance.
load_version(version_id, sections)	Load specified sections of a Conclusion screen version implementation.

_expose(*func*)

Wraps, renames and exposes a function to eel.

Return type
TypeVar(F, bound= Callable[..., Any])

Parameters
func (*F*)

_handleexception(*exc*)

Passes exception to exceptionhandler if defined, otherwise continues raising.

Return type
None

Parameters
exc (*Exception*)

end(*version_id*)

Redirect following sequence logic after conclusion screen ends.

Return type
bool

Parameters
version_id (*str*)

getversions()

Retrieves the available versions of the Conclusion screen.

Return type
dict[str, str]

init(*data*)

Initialises a new Conclusion screen instance.

Return type
str

Parameters
data (*dict[str, Any]*)

load_version(*version_id*, *sections*)

Load specified sections of a Conclusion screen version implementation.

Return type
dict[str, dict[str, Any]]

Parameters

- **version_id** (*str*)
- **sections** (*list[str]*)

F = TypeVar(F, bound=Callable)

Type: TypeVar

Invariant TypeVar bound to typing.Callable[..., typing.Any].

research_client.conclusion.versions

Version implementations and translations for the conclusion screen.

_get_versions()

Loads all available conclusion screen versions into memory.

Return type
dict[str, dict[str, Any]]

5.1.6 research_client.config

Configuration handler for L'ART Research Client.

This package provides an API to read, modify, and store app configuration. The configuration is stored in a JSON file and relevant paths (unless explicitly specified) are determined based on the Operating System (using the *AppDirs* package).

The configuration package only exposes the actual interface to the configuration, which is done via the singleton *config* object, an instantiation of the *Config* class.

To access and/or modify the configuration of the running app, you should import only *config*. The other classes and objects in the package will not typically be needed, perhaps with the exceptions of functions that deal with system updates and the like (as for instance the functionality in the *research_client.utils* module).

Example

Let's imagine you want to ensure that the *shutdown_delay* setting is always at least three seconds. The following example shows how you would load the current app configuration, check the current value, and if it is below the threshold increase it to *3.00* and then save the modified configuration (so that it will persist when the app is restarted):

```
from .config import config

if config.shutdown_delay < 3.00:
    config.shutdown_delay = 3.00
    config.save()
```

Module Attributes

config	The default configuration object for the app.

Classes

Config	Class for keeping track of App configuration data.
DataclassDictMixin	Mixin adding asdict() and fromdict() methods to a dataclass.
DataclassDocMixin	Mixin adding a getdocs() method to a dataclass.
JSONPathEncoder	JSON Encoder capable of serialising pathlib Path objects.
Logging	Class for Logging configuration.
Paths	Class for configuration of App paths.
Sequences	Class for app-task sequencing configuration.

class Config

Bases: *DataclassDictMixin*, *DataclassDocMixin*

Class for keeping track of App configuration data.

__init__(*logging=Logging(max_files=10, default_level=30, stream_format='{levelname}:{name}: {message}', file_format='[{asctime} {levelname:<8} {name}] {message}'), paths=Paths(config=PosixPath('/home/docs/.config/Research_Client'), data=PosixPath('/home/docs/.local/share/Research_Client/Data'), logs=PosixPath('/home/docs/.local/state/Research_Client/log'), cache=PosixPath('/home/docs/.cache/Research_Client')), sequences=Sequences(agt='', atolc='memorygame', conclusion='', consent='lsbq', lsbq='atolc', memorygame=''), shutdown_delay=2.0*)

Parameters

- **logging** (Logging)
- **paths** (Paths)
- **sequences** (Sequences)
- **shutdown_delay** (*float*)

Return type

None

classmethod load(*filename='settings.json'*)

Load configuration from a file or return default Config().

Return type
Config

Parameters
filename (*str*)

save(*filename='settings.json'*)

Save configuration to a file.

Parameters
filename (*str*)

appauthor = 'L'ART'

Type: str

appname = 'Research Client'

Type: str

appversion = '0.3.4'

Type: str

logging = Logging(max_files=10, default_level=30, stream_format='{levelname}:{name}: {message}', file_format='[{asctime} {levelname:<8} {name}] {message}')

Type: *Logging*

paths = Paths(config=PosixPath('/home/docs/.config/Research_Client'), data=PosixPath('/home/docs/.local/share/Research_Client/Data'), logs=PosixPath('/home/docs/.local/state/Research_Client/log'), cache=PosixPath('/home/docs/.cache/Research_Client'))

Type: *Paths*

sequences = Sequences(agt='', atolc='memorygame', conclusion='', consent='lsbq', lsbq='atolc', memorygame='')

Type: *Sequences*

shutdown_delay = 2.0

Type: float

class DataclassDictMixin

Bases: object

Mixin adding asdict() and fromdict() methods to a dataclass.

asdict()

Return a deep copy of the dataclass as a dictionary.

Return type
dict[str, Any]

classmethod fromdict(*d*, *ignorefaults=False*)

Recursively converts a dictionary to a dataclass instance.

Parameters

- **d** (*dict[str, Any]*)
- **ignorefaults** (*bool*)

class DataclassDocMixin

Bases: object

Mixin adding a getdocs() method to a dataclass.

This enables adding additional documentation to dataclass fields using the field's *metadata* property. The following metadata properties are read by the DataclassDocMixin:

- doc_label: A human-friendly label/short description of a field.
- doc_help: A human-friendly explanation of what a field does / why it's there.
- doc_values: A dictionary of label-value pairs, which can be used to give an indication of specific values the field can take and what they mean.

getdocs(*recurse=True*, *include_undocumented=False*)

Return a dictionary with documentation for the dataclass's fields.

Parameters

- **recurse** (bool, default: True) – Whether to also fetch the field-docs for fields that are dataclasses.
- **include_undocumented** (bool, default: False) – Whether to include fields that do not have at least *doc_label* assigned in their metadata.

Return type
tuple[list[dict[str, Any]], list[dict[str, Any]]]

Returns: A two-tuple, where the first member is a list of dataclass
fields, the second member a list of non-dataclass fields. Each list contains a dictionary with information about the field.

class JSONPathEncoder

Bases: JSONEncoder

JSON Encoder capable of serialising pathlib Path objects.

default(*o*)

Encodes pathlib Path objects as strings, otherwise uses default JSONEncoder.

Parameters
o (*Any*)

class Logging

Bases: *DataclassDictMixin*, *DataclassDocMixin*

Class for Logging configuration.

__init__(*max_files=10*, *default_level=30*, *stream_format='{levelname}:{name}: {message}'*, *file_format='[{asctime} {levelname:<8} {name}] {message}'*)

Parameters

- **max_files** (*int*)
- **default_level** (*int*)
- **stream_format** (*str*)
- **file_format** (*str*)

Return type
None

_get_file_path(*name*, *path*)

Determine log file path and clear old log files.

Scans *path* for logfiles named after *name* (*name**_.log). If there are more than *config.logging.max_files - 1*, removes the oldest until that value is reached, and returns a path to a new log file (without creating it yet).

Return type
Path

Parameters

- **name** (*str*)
- **path** (*Path*)

get_file_handler(*name*, *path=None*)

Return a *logging.FileHandler* object for logging.

Return type
FileHandler

Parameters

- **name** (*str*)
- **path** (*Optional[Union[Path, str]]*)

get_stream_handler(*stream=None*)

Return a *logging.StreamHandler* object for logging.

Parameters
stream (*Any*)

Return type
logging.StreamHandler[*Any*]

default_level = 30

Type: int

file_format = '[{asctime} {levelname:<8} {name}] {message}'

Type: str

max_files = 10

Type: int

stream_format = '{levelname}:{name}: {message}'

Type: str

class Paths

Bases: *DataclassDictMixin*, *DataclassDocMixin*

Class for configuration of App paths.

__init__(*data=PosixPath('/home/docs/.local/share/Research_Client/Data')*, *logs=PosixPath('/home/docs/.local/state/Research_Client/log')*, *cache=PosixPath('/home/docs/.cache/Research_Client')*)

Parameters

- **data** (*Path*)
- **logs** (*Path*)
- **cache** (*Path*)

Return type
None

`cache = PosixPath('/home/docs/.cache/Research_Client')`

Type: Path

`config = PosixPath('/home/docs/.config/Research_Client')`

Type: Path

`data = PosixPath('/home/docs/.local/share/Research_Client/Data')`

Type: Path

`logs = PosixPath('/home/docs/.local/state/Research_Client/log')`

Type: Path

`class Sequences`

Bases: *DataclassDictMixin*, *DataclassDocMixin*

Class for app-task sequencing configuration.

`__init__`(*agt=''*, *atolc='memorygame'*, *conclusion=''*, *consent='lsbq'*, *lsbq='atolc'*, *memorygame=''*)

Parameters

- **agt** (*str*)
- **atolc** (*str*)
- **conclusion** (*str*)
- **consent** (*str*)
- **lsbq** (*str*)
- **memorygame** (*str*)

Return type
None

`_sequence_options = {'AGT': 'agt', 'AToL-C': 'atolc', 'App start screen': '', 'Conclusion Screen': 'conclusion', 'Consent Form': 'consent', 'LSBQe': 'lsbq', 'Memory Task': 'memorygame'}`

Type: ClassVar[dict[str, str]]

`agt = ''`

Type: str

`atolc = 'memorygame'`

Type: str

`conclusion = ''`

Type: str

`consent = 'lsbq'`

Type: str

`lsbq = 'atolc'`

Type: str

`memorygame = ''`

Type: str

```
config = Config(appname='Research Client', appauthor='L'ART', appversion='0.3.4',
logging=Logging(max_files=10, default_level=30, stream_format='{levelname}:{name}:
{message}', file_format='[{asctime} {levelname:<8} {name}] {message}'),
paths=Paths(config=PosixPath('/home/docs/.config/Research_Client'),
data=PosixPath('/home/docs/.local/share/Research_Client/Data'),
logs=PosixPath('/home/docs/.local/state/Research_Client/log'),
cache=PosixPath('/home/docs/.cache/Research_Client')), sequences=Sequences(agt='',
atolc='memorygame', conclusion='', consent='lsbq', lsbq='atolc', memorygame=''),
shutdown_delay=2.0)
```

Type: Final[*Config*]

The default configuration object for the app.

This is an instance of *Config* loaded from the users' stored settings upon module initialisation (if available), otherwise it is populated with (sensible) default values.

When saved with *config.save()* it will automatically save in the correct file and location depending on the user's system and installation type.

5.1.7 research_client.consent

Informed consent from user.

Functions

consent_getversions()	loops through the folder [versions] inside [consent] and finds all unique language versions, regardless of test type.
fetch_file_info(file)	takes a json file and returns info from inside the file as a list in the form of [version_name, version_ID, version_data]
fetch_study_info(filename)	takes a filename and returns the json data from that file
record_consent(data)	Takes in data from index.html and prints it to file && to console.
set_options(selected_version)	Takes a language version as arg and finds all consent forms available for that version.

consent_getversions()

loops through the folder [versions] inside [consent] and finds all unique language versions, regardless of test type. Returns a list of lists in the form: [versionId, languageInfo]

fetch_file_info(*file*)

takes a json file and returns info from inside the file as a list in the form of [version_name, version_ID, version_data]

Parameters
file (*str*)

fetch_study_info(*filename*)

takes a filename and returns the json data from that file

Parameters
filename (*str*)

record_consent(*data*)

Takes in data from index.html and prints it to file && to console.

Parameters
data (*dict[Any, Any]*)

set_options(*selected_version*)

Takes a language version as arg and finds all consent forms available for that version. Returns a list in the form of [version_name, version_ID, version_data]

Parameters
selected_version (*str*)

Modules

research_client.consent.versions

research_client.consent.versions

5.1.8 research_client.datavalidator

Simple data validation tools.

This package contains a set of classes providing a simple interface for data validation and validation-based user feedback via Validator objects.

Modules

research_client.datavalidator.exceptions	Exceptions for the datavalidator package.
research_client.datavalidator.patterns	Patterns/ranges/enums/etc.
research_client.datavalidator.schemas	Prototypes to conveniently define data classes with built-in validation.
research_client.datavalidator.types	Type definitions for the datavalidator package.
research_client.datavalidator.validation	Validation tools for the datavalidator package.

research_client.datavalidator.exceptions

Exceptions for the datavalidator package.

Exceptions

DataValidationError	Exception raised when one or more data validations have failed.

exception DataValidationError

Bases: Exception

Exception raised when one or more data validations have failed.

__init__(*message*, *validator*)

Constructs a new DataValidationError exception.

Parameters

- **message** (str) – The message to be shown to the user.
- **validator** (*Validator*) – The list of the ValidationResults that have failed validation.

errors

Type: list[*ValidationResult*]

The list of the ValidationResults that have failed validation.

message

Type: str

The message to be shown to the user.

validator

Type: *Validator*

The Validator object from which the DataValidationError originated.

research_client.datavalidator.patterns

Patterns/ranges/enums/etc. for validating different common data input types.

research_client.datavalidator.schemas

Prototypes to conveniently define data classes with built-in validation.

Classes

CField	Defines a manually validated data field for DataSchema classes.
CFieldList	Defines a field containing an arbitrary number of CField data.
DataField	Base class for data schema fields.
DataFieldList	Defines a field containing an arbitrary number of DataField data.
DataGroup	Defines a data group for DataSchema classes.
DataSchema	Abstract base class to define auto-validating data classes.
VField	Defines an auto-validated data field for DataSchema classes.
VFieldList	Defines a field containing an arbitrary number of VField data.

class CField

Bases: *DataField*

Defines a manually validated data field for DataSchema classes.

__init__(*name*, *type_*, *typedesc*, *vmethod*, *forcecast=None*, *required=True*)

Instantiates a new CField.

Parameters

- **name** (*str*)
- **type_** (*str*)
- **typedesc** (*str*)
- **vmethod** (*Callable[[Any, Any], Any]*)
- **forcecast** (*Optional[bool]*)
- **required** (*bool*)

fieldparams = [('name', <class 'str'>, 'The name of the DataField'), ('type', typing.Any, 'The data type for the DataField'), ('typedesc', <class 'str'>, 'A user-intelligible description of the data type'), ('vmethod', typing.Callable[[typing.Any, typing.Any], typing.Any], 'A callable accepting a two arguments: the first is the type_ of the DataField and the second the value. The callable should raise either a TypeError or a DataValidationError if the value is invalid, and must return a (possibly processed) version of the value it was passed which fits the type_ it was passed.'), ('forcecast', typing.Optional[bool], 'Whether to force casting of data during validation', None), ('required', <class 'bool'>, 'Whether the field is required', True)]

Type: list[Union[tuple[str, Any, str], tuple[str, Any, str, Any]]]

forcecast

Type: Optional[bool]

vmethod

Type: Union[str, Callable[[Any, Any], Any]]

class CFieldList

Bases: *CField*, *DataFieldList*

Defines a field containing an arbitrary number of CField data.

class DataField

Bases: object

Base class for data schema fields.

To instantiate, use VField, CField, or shorthand notation in a DataSchema declaration.

__init__(*name*, *type_*, *typedesc*, *required=True*)

Instantiates a new DataField.

Parameters

- **name** (*str*)
- **type_** (*Any*)
- **typedesc** (*str*)
- **required** (*bool*)

classmethod fieldparams()

Returns the parameter list for a DataField of this type.

Return type
list[Union[tuple[str, Any, str], tuple[str, Any, str, Any]]]

fieldspecs()

Returns the values for each parameter of the DataField.

Return type
dict[str, Any]

fieldparams = [('name', <class 'str'>, 'The name of the DataField'), ('type', typing.Any, 'The data type for the DataField'), ('typedesc', <class 'str'>, 'A user-intelligible description of the data type'), ('required', <class 'bool'>, 'Whether the field is required', True)]

Type: list[Union[tuple[str, Any, str], tuple[str, Any, str, Any]]]

name

Type: str

required

Type: bool

type_

Type: Any

typedesc

Type: str

class DataFieldList

Bases: *DataField*

Defines a field containing an arbitrary number of DataField data.

class DataGroup

Bases: object

Defines a data group for DataSchema classes.

__init__(*name*, *fields*)

Instantiates a new DataGroup.

Parameters

- **name** (*str*)
- **fields** (*dict[str, Union[DataField, dict[str, Any]]]*)

getfield(*key*)

Gets the data field with the name indicated by key.

Return type
Union[dict[str, Any], *DataField*]

Parameters
key (*str*)

class DataSchema

Bases: object

Abstract base class to define auto-validating data classes.

@TODO: - __getattr__(self, name) - to retreive data

- __setattr__(self, name, value) - to set data (with validation)
- __delattr__(self, name) - to remove/clear a datapoint
- Move some __new__ stuff to __init_subclass__(cls)?
- JSON import/export

__delfieldfactory(*fieldname*, *fieldspecs*)

Return type
Callable[[*DataSchema*], None]

Parameters

- **fieldname** (*str*)
- **fieldspecs** (*dict[str, Any]*)

__delfieldlistfactory(*fieldname*, *fieldspec*)

Return type
Callable[[*DataSchema*], None]

Parameters

- **fieldname** (*str*)
- **fieldspec** (*dict[str, Any]*)

__delgroupfactory(*gname*, *fieldspecs*)

Return type
Callable[[*DataSchema*], None]

Parameters

- **gname** (*str*)
- **fieldspecs** (*dict[str, dict[str, Any]]*)

classmethod __functionalize()

Dynamically creates and attaches methods to get/set values.

Return type
None

__getfieldfactory(*fieldname*, *fieldspecs*)

Return type
Callable[[*DataSchema*], Any]

Parameters

- **fieldname** (*str*)
- **fieldspecs** (*dict[str, Any]*)

__getfieldlistfactory(*fieldname*, *fieldspec*)

Return type
Callable[[*DataSchema*], list[Any]]

Parameters

- **fieldname** (*str*)
- **fieldspec** (*dict[str, Any]*)

classmethod __getfieldspecs(*key*)

Get the specifications for a single DataField or fields in a DataGroup.

Return type
dict[str, dict[str, Any]]

Parameters
key (*str*)

__getgroupfactory(*gname*, *fieldspecs*)

Return type
Callable[[*DataSchema*], dict[str, Any]]

Parameters

- **gname** (*str*)
- **fieldspecs** (*dict[str, dict[str, Any]]*)

classmethod __index()

Creates a flat list of index keys, separating groups and fields with "/".

Return type
list[str]

__init__(*forcecast=True*, *ignorecase=True*)

Initialises a new DataSchema object.

Parameters

- **forcecast** (*bool*)
- **ignorecase** (*bool*)

classmethod __materialize()

Creates an empty __data store pre-populated with DataGroups/DataFieldLists.

Return type
dict[str, Union[dict[str, Any], Any]]

static __new__(*cls*, **args*, ***kwargs*)

Constructs a new DataSchema instance.

Parameters

- **args** (*Any*)
- **kwargs** (*Any*)

classmethod __schematize(*schema*, *schemaname*)

Return type
Union[*DataGroup*, *DataField*]

Parameters

- **schema** (*Union[DataGroup, dict[str, Union[DataField, dict[str, Any]]], DataField, dict[str, Any]]*)
- **schemaname** (*str*)

__setfieldfactory(*fieldname*, *fieldspec*)

Return type
Callable[[*DataSchema*, Any], None]

Parameters

- **fieldname** (*str*)
- **fieldspec** (*dict[str, Any]*)

__setfieldlistfactory(*fieldname*, *fieldspec*)

Return type
Callable[[*DataSchema*, list[Any]], None]

Parameters

- **fieldname** (*str*)
- **fieldspec** (*dict[str, Any]*)

__setgroupfactory(*gname*, *fieldspecs*)

Return type
Callable[[*DataSchema*, dict[str, Any]], None]

Parameters

- **gname** (*str*)
- **fieldspecs** (*dict[str, dict[str, Any]]*)

_autovalidate(*vr*, *fieldspec*, *value*)

Calls the appropriate validation method for fieldspec and value.

Return type
ValidationResult

Parameters

- **vr** (Validator)
- **fieldspec** (*dict[str, Any]*)
- **value** (*Any*)

_customvalidate(*vr*, *fieldspec*, *value*)

Calls a custom validation method on value and appends result to vr.

Return type
ValidationResult

Parameters

- **vr** (Validator)
- **fieldspec** (*dict[str, Any]*)
- **value** (*Any*)

_getfield(*key*)

Gets the field or group addressed by key.

Return type
Union[*DataField*, *DataGroup*]

Parameters
key (*str*)

_getvalue(*key*)

Gets the value from the data addressed by key.

Return type
Any

Parameters
key (*str*)

static _isna(*value*)

Returns True if value is either None or [], False otherwise.

Return type
bool

Parameters
value (*Any*)

_setvalue(*key*, *value*)

Sets the value for the data addressed by key (without validation).

Return type
None

Parameters

- **key** (*str*)
- **value** (*Any*)

data(*includemissing=False*, *onlyrequired=False*)

Returns the data of the DataSchema as a schematic dictionary.

Return type
dict[str, Union[dict[str, Any], Any]]

Parameters

- **includemissing** (*bool*)
- **onlyrequired** (*bool*)

iscomplete(*onlyrequired=True*)

Checks whether the dataset is complete.

Return type
bool

Parameters
onlyrequired (*bool*)

items(*includemissing=False*, *onlyrequired=False*)

Returns a list of key-value pairs for data in the DataSchema.

Return type
list[tuple[str, Any]]

Parameters

- **includemissing** (*bool*)
- **onlyrequired** (*bool*)

keys(*includemissing=False*, *onlyrequired=False*)

Returns a list of keys for the DataSchema.

Return type
list[str]

Parameters

- **includemissing** (*bool*)
- **onlyrequired** (*bool*)

missing(*onlyrequired=True*)

Return a list of keys for missing fields.

Return type
list[str]

Parameters
onlyrequired (*bool*)

values(*includemissing=False*, *onlyrequired=False*)

Returns a list of values for the data in the DataSchema.

Return type
list[Any]

Parameters

- **includemissing** (*bool*)
- **onlyrequired** (*bool*)

__data

Type: dict[str, Union[dict[str, Any], Any]]

__keys

Type: list[str]

__schema

Type: dict[str, Union[*DataGroup*, dict[str, Union[*DataField*, dict[str, Any]]], *DataField*, dict[str, Any]]]

__schematized

Type: bool

forcecast

Type: bool

ignorecase

Type: bool

class VField

Bases: *DataField*

Defines an auto-validated data field for DataSchema classes.

__init__(*name*, *type_*, *typedesc*, *constraint*, *forcecast=None*, *ignorecase=None*, *flags=0*, *required=True*)

Instantiates a new VField.

Parameters

- **name** (*str*)
- **type_** (*str*)
- **typedesc** (*str*)
- **constraint** (*Any*)
- **forcecast** (*Optional[bool]*)
- **ignorecase** (*Optional[bool]*)
- **flags** (*Union[RegexFlag, int]*)
- **required** (*bool*)

fieldparams = [('name', <class 'str'>, 'The name of the DataField'), ('type', typing.Any, 'The data type for the DataField'), ('typedesc', <class 'str'>, 'A user-intelligible description of the data type'), ('constraint', typing.Any, 'A DataValidator constraint approprite for type_'), ('forcecast', typing.Optional[bool], 'Whether to force casting of data during validation', None), ('ignorecase', typing.Optional[bool], 'Whether to ignore case for string-type data validation', None), ('flags', typing.Union[re.RegexFlag, int], 'Flags to pass to the regular expression engine if type_ is `str`.', 0), ('required', <class 'bool'>, 'Whether the field is required', True)]

Type: list[Union[tuple[str, Any, str], tuple[str, Any, str, Any]]]

constraint

Type: Any

flags

Type: Union[RegexFlag, int]

forcecast

Type: Optional[bool]

ignorecase

Type: Optional[bool]

class VFieldList

Bases: *VField*, *DataFieldList*

Defines a field containing an arbitrary number of VField data.

research_client.datavalidator.types

Type definitions for the datavalidator package.

Module Attributes

RangeT	Type for inclusive numeric ranges (int or float).
SetT	Type for checking whether whether a value is contained in a set.
PolarT	Type for checking whether value falls within a set of polar items.
PatternT	Type for regular expression patterns for string matching.
EnumT	Type for enumerable checking and valuation.

EnumT

Type for enumerable checking and valuation.

Note: The dictionary keys and values can be any type. If a value repeats for more than one individual key it will be treated as an alias.

Examples

- {"A": 1, "B": 2} will match inputs "A", 1, "B", and 2, and cast to either 1 or 2.

PatternT

Type for regular expression patterns for string matching.

Note: In use, a PatternT string will always be annotated with a preceding r"A" and a succeeding r"Z" to match the start and end of a string exhaustively.

Examples

- r"w*" will match anything matched by r"Aw*Z".

PolarT

Type for checking whether value falls within a set of polar items.

Examples

- ({"yes", "on", "true"}, {"no", "off", "false"}) checks whether a given value is in the set {"yes", "on", "true"} or the set {"no", "off", "false"}, and depending on the function might return True for the former set and False for the latter.

Alias of `Union[tuple[Iterable[Any], Iterable[Any]], list[Iterable[Any]]]`

RangeT

Type for inclusive numeric ranges (int or float).

Examples

- (1, 10) validates integers and floats 1 through 10 inclusive.
- (1.0, 10.0) same as (1, 10).
- (0.5, 9.5) validates integers 1 through 10 inclusive, but floats only from 0.50 to 9.50 inclusive.
- [1, 10] same as (1, 10).
- [0.5, 9.5] same as (0.5, 9.5).

Alias of `Union[tuple[int, int], tuple[float, float], list[int], list[float]]`

SetT

Type for checking whether whether a value is contained in a set.

Examples

- (1, 2, 3) checks whether a given value is equal to the integer 1, 2, or 3.
- (True, False) checks whether a given value evaluates to True or False.
- {"A", "b", 3} checks whether a given value equals "A", "b", or the int 3.

Alias of `Iterable[Any]`

XT = TypeVar(XT)

Type: `TypeVar`

Invariant `TypeVar`.

YT = TypeVar(YT)

Type: `TypeVar`

Invariant `TypeVar`.

research_client.datavalidator.validation

Validation tools for the datavalidator package.

Classes

ValidationResult	Data validation result.
Validator	Data validation interface.

class ValidationResult

Bases: object

Data validation result.

Each ValidationResult represents the result of a data validation attempt in a Validator. The ValidationResult can be queried for details on the data that was validated, whether it passed/failed validation, what the requirements for validation were, etc. A ValidationResult will evaluate to True if the validation has succeeded and to False if it has failed.

__init__(*success*, *type_*, *typedesc*, *constraint*, *data*, *rawdata*, *casting*)

Constructs a new ValidationResult.

Parameters

- **success** (bool) – Whether the validaton has succeeded or failed.
- **type** – The built-in data type against which validation was carried out.
- **typedesc** (str) – A short user-intelligible description of the data's type.
- **constraint** (Union[str, tuple[int, int], tuple[float, float], list[int], list[float], Iterable[Any], tuple[Iterable[Any], Iterable[Any]], list[Iterable[Any]], dict[Any, Any], bool, None]) – A constraint data type appropriate to the type of the data, see also the *types* submodule.
- **data** (Any) – The data that was evaluated (after casting if the forcecasting and/or softcasting options were active.)
- **rawdata** (Any) – The raw, uncast data as it was passed to the validation method.
- **casting** (bool) – Whether casting was applied to *rawdata* to yield *data*.
- **type_** (*Any*)

tohtml()

Returns HTML formatted explanation of the data validation result.

Return type
str

tojson()

Returns a JSON representation of the data validation result.

Return type
str

tostring()

Returns string explanation of the data validation result.

Return type
str

`casting`

Type: bool

Whether the data in *data* was cast or not.

Note that this does not necessarily mean that casting was necessary, e.g. an integer that was passed to a Validator's *vint()* method will still have been cast to int() and set the casting attribute to True despite being of type int before.

`constraint`

Type: Union[str, tuple[int, int], tuple[float, float], list[int], list[float], Iterable[Any], tuple[Iterable[Any], Iterable[Any]], list[Iterable[Any]], dict[Any, Any], bool, None]

The constraint, if any, which was used to validate the data.

`data`

Type: Any

The data itself, possibly cast.

This is the data post-casting if force- or softcasting were used, and can be used to automatically ensure typecasting or type-narrowing for data storage. For the raw data pre-casting use the *raw_data* attribute.

`rawdata`

Type: Any

The raw data, as it was passed to the validator.

This is always the data as it was passed to the Validator, irrespective of whether forcecasting or softcasting were applied.

`success`

Type: bool

Whether the data has been succeessfully validated or not.

`type_`

Type: Any

The internal data type indicated for validation.

`typedesc`

Type: str

A user-directed description of the type of data being validated.

`class Validator`

Bases: object

Data validation interface.

A Validator offers a convenient interface for validating a set of data points, of the same or different types. The Validator will store any failed validation results, can optionally force casting of the data to a specific type, can be evaluated for success in a boolean expression, and allows for the conditional raising of a DataValidationError exception if any validation attempts have failed.

A single Validator should only be used once for a closed set of data, as reuse will add the results to the existing Validator and always evaluate False if it has previously had unsuccessful validation attempts (though under some circumstances, e.g. the successive building of datasets with late repairs, this may be desirable).

__casefolddict(*dict_*)

Return type
dict[TypeVar(XT), TypeVar(YT)]

Parameters
dict_ (*dict[~XT, ~YT]*)

__casefoldifstr(*x*)

Return type
TypeVar(XT)

Parameters
x (*XT*)

__condcast(*type_*, *data*, *overwrite=None*)

Conditionally casts data to a type.

Attempts to cast *data* to *type_* if, taking into account *overwrite*, forcecasting applies. Returns *data* itself if forcecasting doesn't apply, and None if forcecasting applies but *data* cannot be cast to *type_*.

Return type
Union[TypeVar(XT), TypeVar(YT), None]

Parameters

- **type_** (*Callable[[...], XT]*)
- **data** (*YT*)
- **overwrite** (*Optional[bool]*)

__forcecast(*overwrite=None*)

Return type
bool

Parameters
overwrite (*Optional[bool]*)

__ignorecase(*overwrite=None*)

Parameters
overwrite (*Optional[bool]*)

__init__(*forcecast=False*, *ignorecase=False*)

Constructs a new Validator.

Parameters

- **forcecast** (bool, default: False) – Whether to force casting of the data arguments to the validation methods to the indicated type (e.g. str for .validatestring()). This will set the default behaviour for validation calls, but can be overwritten by passing the named argument forcecast=True or forcecast=False on individual method calls. For some methods, e.g. polars and enums, casting is done by passing the matched value rather than typecasting.
- **ignorecase** (bool, default: False) – Whether to ignore case in string comparisons. If true, strings will be compared in all uppercase, and regular expression matches will be passed the IGNORECASE flag. Can be overwritten on each validation call by passing ignorecase=True/False. Default value: False.

__storeresult(*result*)

Parameters
result (ValidationResult)

__trycall(*func*, **args*, ***kwargs*)

Returns result of func() if possible, None if an Exception is raised.

Return type
Optional[TypeVar(XT)]

Parameters

- **func** (*Callable[[...], XT]*)
- **args** (*Any*)
- **kwargs** (*dict[Any, Any]*)

raiseif()

Raises a DataValidationError iff at least one validation has failed.

tohtml(*errorsonly=False*)

Returns a paragraph-by-paragraph HTML representation of validation attempts.

Parameters
errorsonly (bool, default: False) – Whether to include only the errors or all validation attempts.

tostring(*errorsonly=False*)

Returns a line-by-line string representation of validation attempts.

Parameters
errorsonly (bool, default: False) – Whether to include only the errors or all validation attempts.

vbool(*typedesc*, *constraint*, *data*, *forcecast=None*)

Validates a bool.

Parameters

- **typedesc** (str) – An end-user intelligible description of the desired data type, e.g. "User ID" or "postcode".
- **constraint** (bool) – A boolean that must be matched, or None to just validate any bool.
- **data** (Any) – The data to be validated.
- **forcecast** (Optional[bool], default: None) – Optional argument to overwrite the objects default setting for forced casting of data arguments.

venum(*typedesc*, *constraint*, *data*, *forcecast=None*, *ignorecase=None*)

Validates a string against a key:value enumerable.

Checks whether *data* is either contained in the keys or the values of the enumerable. If forcecasting is used, it casts to the *value* that was matched (not the key), or to None if no match was found.

Return type
ValidationResult

Parameters

- **typedesc** (*str*)

- **constraint** (*dict[Any, Any]*)
- **data** (*Any*)
- **forcecast** (*Optional[bool]*)
- **ignorecase** (*Optional[bool]*)

vfloat(*typedesc*, *constraint*, *data*, *forcecast=None*)

Validates a float against an inclusive range of integers or floats.

Parameters

- **typedesc** (str) – An end-user intelligible description of the desired data type, e.g. "User ID" or "postcode".
- **constraint** (Union[tuple[int, int], tuple[float, float], list[int], list[float]]) – A two member tuple or list of integers where the first element represents the inclusive lower bound and the second member the inclusive upper bound of the permissible range of integer values. For example, (3, 5) would successfully validate the data inputs 3, 4, 5, but fail validation for 2 or 6.
- **data** (Any) – The data to be validated.
- **forcecast** (Optional[bool], default: None) – Optional argument to overwrite the objects default setting for forced casting of data arguments.

vint(*typedesc*, *constraint*, *data*, *forcecast=None*)

Validates an integer against an inclusive range of integers or floats.

Parameters

- **typedesc** (str) – An end-user intelligible description of the desired data type, e.g. "User ID" or "postcode".
- **constraint** (Union[tuple[int, int], tuple[float, float], list[int], list[float]]) – A two member tuple or list of integers where the first element represents the inclusive lower bound and the second member the inclusive upper bound of the permissible range of integer values. For example, (3, 5) would successfully validate the data inputs 3, 4, 5, but fail validation for 2 or 6.
- **data** (Any) – The data to be validated.
- **forcecast** (Optional[bool], default: None) – Optional argument to overwrite the validator's default setting for forced casting of data arguments.

Return type
ValidationResult

vpolar(*typedesc*, *constraint*, *data*, *forcecast=None*, *ignorecase=None*)

Validates a string against two sets of polar terms.

Checkes whether data is in either of two sets of polar opposition terms. If the forcecast option is active, membership in the first of the two sets results in casting to True, membership in the second set to False, and memebership in neither set to None. Validation is successful if data is contained in either of the two sets, and unsuccessful otherwise.

Parameters

- **typedesc** (str) – An end-user intelligible description of the desired data type, e.g. "User ID" or "postcode".

- **constraint** (Union[tuple[Iterable[Any], Iterable[Any]], list[Iterable[Any]]]) – A two member tuple or list of containers of any type (must support membership testing with *in*). The first member is a container of acceptable truthy values, the second is a container of acceptable falsy values. Note that python built-in boolean types True and False are always validated as correct.
- **data** (Any) – The data to be validated.
- **forcecast** (Optional[bool], default: None) – Optional argument to overwrite the objects default setting for forced casting of data arguments.
- **ignorecase** (Optional[bool], default: None) – Optional argument to overwrite the validator's default setting for case sensitivity.

Return type
ValidationResult

vstr(*typedesc*, *constraint*, *data*, *forcecast=None*, *ignorecase=None*, *flags=0*)

Validates a string against a regular expression pattern.

Parameters

- **typedesc** (str) – An end-user intelligible description of the desired data type, e.g. "User ID" or "postcode".
- **constraint** (str) – A regular expression to match the string against. Important: Note that the regular expression will implicitly be enclosed by A and Z to match the beginning and end of the string. These thus need not be specified in the pattern provided.
- **data** (Any) – The data to be validated.
- **forcecast** (Optional[bool], default: None) – Optional argument to overwrite the validator's default setting for forced casting of data arguments.
- **ignorecase** (Optional[bool], default: None) – Optional argument to overwrite the validator's default setting for case sensitivity.
- **flags** (Union[RegexFlag, int], default: 0) – Additional regex flags to be passed to re.match().

Return type
ValidationResult

failed

Type: list[*ValidationResult*]

forcecast

Type: bool

ignorecase

Type: bool

results

Type: list[*ValidationResult*]

successful

Type: list[*ValidationResult*]

5.1.9 research_client.lsbq

Package implementing the Language and Social Background Questionnaire.

Functions

expose_to_eel()	Expose the LSBQe API to Python Eel.

expose_to_eel()

Expose the LSBQe API to Python Eel.

Modules

research_client.lsbq.dataschema	Data schema implementing the LSBQe questionnaire.
research_client.lsbq.eel	Exposes the LSBQe to Python Eel.
research_client.lsbq.patterns	Additional validation patterns for LSBQe.
research_client.lsbq.versions	Version implementations and translations for the LSBQe.

research_client.lsbq.dataschema

Data schema implementing the LSBQe questionnaire.

Classes

Response	Class for representing the data of an LSBQe questionnaire response.

class Response

Bases: *DataSchema*

Class for representing the data of an LSBQe questionnaire response.

__init__(*id_=None*)

Instantiates a new LSBQe response object.

Parameters
id_ (*Optional[str]*)

```
__schema = {'club': {'childhood_friends': {'constraint': (-9223372036854775808,
9223372036854775807), 'required': False, 'type_': <class 'float'>, 'typedesc':
'proportion of language use with friends in childhood'}, 'childhood_grandparents':
{'constraint': (-9223372036854775808, 9223372036854775807), 'required': False,
'type_': <class 'float'>, 'typedesc': 'proportion of language use with
grandparents in childhood'}, 'childhood_neighbours': {'constraint':
(-9223372036854775808, 9223372036854775807), 'required': False, 'type_': <class
'float'>, 'typedesc': 'proportion of language use with neighbours in childhood'},
'childhood_other_relatives': {'constraint': (-9223372036854775808,
9223372036854775807), 'required': False, 'type_': <class 'float'>, 'typedesc':
'proportion of language use with other relatives in childhood'},
'childhood_parents': {'constraint': (-9223372036854775808, 9223372036854775807),
'required': False, 'type_': <class 'float'>, 'typedesc': 'proportion of language
use with parents in childhood'}, 'childhood_siblings': {'constraint':
(-9223372036854775808, 9223372036854775807), 'required': False, 'type_': <class
'float'>, 'typedesc': 'proportion of language use with siblings in childhood'},
'children': {'constraint': (-9223372036854775808, 9223372036854775807),
'required': False, 'type_': <class 'float'>, 'typedesc': 'proportion of language
use with children'}, 'commercial': {'constraint': (-9223372036854775808,
9223372036854775807), 'required': False, 'type_': <class 'float'>, 'typedesc':
'proportion of language use for commercial activities'}, 'emailing': {'constraint':
(-9223372036854775808, 9223372036854775807), 'required': False, 'type_': <class
'float'>, 'typedesc': 'proportion of language use for emailing'}, 'flatmates':
{'constraint': (-9223372036854775808, 9223372036854775807), 'required': False,
'type_': <class 'float'>, 'typedesc': 'proportion of language use with
flat/housemates'}, 'friends': {'constraint': (-9223372036854775808,
9223372036854775807), 'required': False, 'type_': <class 'float'>, 'typedesc':
'proportion of language use with friends'}, 'grandparents': {'constraint':
(-9223372036854775808, 9223372036854775807), 'required': False, 'type_': <class
'float'>, 'typedesc': 'proportion of language use with grandparents'}, 'home':
{'constraint': (-9223372036854775808, 9223372036854775807), 'required': False,
'type_': <class 'float'>, 'typedesc': 'proportion of language use at home'},
'infancy_age': {'constraint': (-9223372036854775808, 9223372036854775807),
'type_': <class 'float'>, 'typedesc': 'proportion of language use in infancy
age'}, 'internet': {'constraint': (-9223372036854775808, 9223372036854775807),
'required': False, 'type_': <class 'float'>, 'typedesc': 'proportion of language
use for internet'}, 'leisure': {'constraint': (-9223372036854775808,
9223372036854775807), 'required': False, 'type_': <class 'float'>, 'typedesc':
'proportion of language use for leisure activities'}, 'neighbours': {'constraint':
(-9223372036854775808, 9223372036854775807), 'required': False, 'type_': <class
'float'>, 'typedesc': 'proportion of language use with neighbours'}, 'notes':
{'constraint': (-9223372036854775808, 9223372036854775807), 'required': False,
'type_': <class 'float'>, 'typedesc': 'proportion of language use for notes'},
'nursery_age': {'constraint': (-9223372036854775808, 9223372036854775807),
'type_': <class 'float'>, 'typedesc': 'proportion of language use in nursery
age'}, 'other_relatives': {'constraint': (-9223372036854775808,
9223372036854775807), 'required': False, 'type_': <class 'float'>, 'typedesc':
'proportion of language use with other relatives'}, 'parents': {'constraint':
(-9223372036854775808, 9223372036854775807), 'required': False, 'type_': <class
'float'>, 'typedesc': 'proportion of language use with parents'}, 'partner':
{'constraint': (-9223372036854775808, 9223372036854775807), 'required': False,
'type_': <class 'float'>, 'typedesc': 'proportion of language use with partner'},
'praying': {'constraint': (-9223372036854775808, 9223372036854775807), 'required':
False, 'type_': <class 'float'>, 'typedesc': 'proportion of language use for
praying'}, 'primary_age': {'constraint': (-9223372036854775808,
9223372036854775807), 'type_': <class 'float'>, 'typedesc': 'proportion of
language use in primary age'}, 'public': {'constraint': (-9223372036854775808,
9223372036854775807), 'required': False, 'type_': <class 'float'>, 'typedesc':
'proportion of language use for public affairs'}, 'reading': {'constraint':
(-9223372036854775808, 9223372036854775807), 'required': False, 'type_': <class
```

Type: dict

research_client.lsbq.eel

Exposes the LSBQe to Python Eel.

Functions

discard(instid)	Discards a Response.
getmissing(instid)	Gets a list of missing fields.
getversions()	Retrieves the available versions of the LSBQe.
init(data)	Initialises a new LSBQe Response.
iscomplete(instid)	Checks whether a Response is complete.
load_version(instid, sections)	Load specified sections of an LSBQe version implementation.
setclub(instid, data)	Adds Community Language Use Behaviour Data to a Response.
setldb(instid, data)	Adds Language and Dialect Background Data to a Response.
setlsb(instid, data)	Adds Language and Social Background Data to a Response.
setnotes(instid, data)	Adds Participant and Experimenter Comments Data to a Response.
store(instid)	Submits a (complete) Response for long-term storage.

_expose(*func*)

Wraps, renames and exposes a function to eel.

Return type
TypeVar(F, bound= Callable[..., Any])

Parameters
func (*F*)

_getinstance(*instid*)

Return type
Response

Parameters
instid (*str*)

_handleexception(*exc*)

Passes exception to exceptionhandler if defined, otherwise continues raising.

Return type
None

Parameters
exc (*Exception*)

discard(*instid*)

Discards a Response.

Return type
bool

Parameters
instid (*str*)

getmissing(*instid*)

Gets a list of missing fields.

Return type
list[str]

Parameters
instid (*str*)

getversions()

Retrieves the available versions of the LSBQe.

Return type
dict[str, str]

init(*data*)

Initialises a new LSBQe Response.

Return type
str

Parameters
data (*dict[str, Any]*)

iscomplete(*instid*)

Checks whether a Response is complete.

Return type
bool

Parameters
instid (*str*)

load_version(*instid*, *sections*)

Load specified sections of an LSBQe version implementation.

Return type
dict[str, dict[str, Any]]

Parameters

- **instid** (*str*)
- **sections** (*list[str]*)

setclub(*instid*, *data*)

Adds Community Language Use Behaviour Data to a Response.

Return type
str

Parameters

- **instid** (*str*)
- **data** (*dict[str, Any]*)

setldb(*instid*, *data*)

Adds Language and Dialect Background Data to a Response.

Return type
str

Parameters

- **instid** (*str*)
- **data** (*dict[str, Any]*)

setlsb(*instid*, *data*)

Adds Language and Social Background Data to a Response.

Return type
str

Parameters

- **instid** (*str*)
- **data** (*dict[str, str]*)

setnotes(*instid*, *data*)

Adds Participant and Experimenter Comments Data to a Response.

Return type
str

Parameters

- **instid** (*str*)
- **data** (*dict[str, Any]*)

store(*instid*)

Submits a (complete) Response for long-term storage.

Return type
bool

Parameters
instid (*str*)

F = TypeVar(F, bound=Callable)

Type: TypeVar

Invariant TypeVar bound to typing.Callable[..., typing.Any].

research_client.lsbq.patterns

Additional validation patterns for LSBQe.

research_client.lsbq.versions

Version implementations and translations for the LSBQe.

_get_versions()

Loads all available LSBQe versions into memory.

Return type
dict[str, dict[str, Any]]

5.1.10 research_client.memorygame

Package implementing the Memory Game for the LART Research Client.

Functions

expose_to_eel()	Expose the Memory Game API to Python Eel.

expose_to_eel()

Expose the Memory Game API to Python Eel.

Modules

research_client.memorygame.dataschema	Data structures for the Memory Game.
research_client.memorygame.eel	Exposes the Memory Game to Python Eel.
research_client.memorygame.patterns	Additional validation patterns for Memory Game.
research_client.memorygame.versions	Version implementations and translations for the Memory Game.

research_client.memorygame.dataschema

Data structures for the Memory Game.

Classes

Response	Class for representing the data of a Memory Game.

class Response

Bases: *DataSchema*

Class for representing the data of a Memory Game.

__init__(*id_=None*)

Instantiates a new LSBQ-RML response object.

Parameters
id_ (*Optional[str]*)

```
__schema = {'id': {'constraint':
'[0-9a-fA-F]{8}-?(?:[0-9a-fA-F]{4}-?){3}[0-9a-fA-F]{12}', 'type_': <class 'str'>,
'typedesc': 'Memory Game Response ID'}, 'meta': {'app_version': {'constraint':
'(?:\\d+.)*\\d+\\w?\\w?\\d*', 'type_': <class 'str'>, 'typedesc': 'Version of app
that last modified the Response'}, 'consent': {'constraint': ({'true', True, 'on',
'1', 'yes'}, {False, 'off', '0', 'false', 'no'}), 'type_':
typing.Union[tuple[typing.Iterable[typing.Any], typing.Iterable[typing.Any]],
list[typing.Iterable[typing.Any]]], 'typedesc': 'consent confirmation'}, 'date':
{'constraint': '[0-9]{1,4}\\-(0?[1-9]|1[0-2])\\-(0?[1-9]|[12][0-9]|3[01])',
'type_': <class 'str'>, 'typedesc': 'current date'}, 'participant_id':
{'constraint': '[A-Za-z0-9]{3,10}', 'type_': <class 'str'>, 'typedesc':
'Participant ID'}, 'research_location': {'constraint': "[\\w,'
\\(\\)\\.\\-]{1,50}", 'type_': <class 'str'>, 'typedesc': 'location name'},
'researcher_id': {'constraint': '[A-Za-z0-9]{3,10}', 'type_': <class 'str'>,
'typedesc': 'Researcher ID'}, 'version_id': {'constraint': '\\w{13,17}', 'type_':
<class 'str'>, 'typedesc': 'Memory Game version identifier'}, 'version_no':
{'constraint': '(?:\\d+.)*\\d+\\w?\\w?\\d*', 'type_': <class 'str'>, 'typedesc':
'Memory Game version number'}}, 'scores': {'score': {'constraint': (0,
9223372036854775807), 'multiple': True, 'type_': <class 'int'>, 'typedesc':
'score'}, 'time': {'constraint': (0, 9223372036854775807), 'multiple': True,
'type_': <class 'int'>, 'typedesc': 'time'}}}
```

Type: dict

research_client.memorygame.eel

Exposes the Memory Game to Python Eel.

Functions

discard(instid)	Discards a Response.
end(instid[, data])	Redirect participant in right sequence after Memory Game end screen.
getmissing(instid)	Gets a list of missing fields.
getversions()	Retrieves the available versions of the Memory Game.
init(data)	Initialises a new Memory Game Response.
iscomplete(instid)	Checks whether a Response is complete.
load_version(instid, sections)	Load specified sections of a Memory Game version implementation.
setscores(instid, data)	Adds Memory Game Scores to a Response.
store(instid)	Submits a (complete) Response for long-term storage.

_expose(*func*)

Wraps, renames and exposes a function to eel.

Return type
TypeVar(F, bound= Callable[..., Any])

Parameters
func (*F*)

_getinstance(*instid*)

Return type
Response

Parameters
instid (*str*)

_handleexception(*exc*)

Passes exception to exceptionhandler if defined, otherwise continues raising.

Return type
None

Parameters
exc (*Exception*)

discard(*instid*)

Discards a Response.

Return type
bool

Parameters
instid (*str*)

end(*instid*, *data=None*)

Redirect participant in right sequence after Memory Game end screen.

Return type
str

Parameters

- **instid** (*str*)
- **data** (*Optional[dict[str, str]]*)

getmissing(*instid*)

Gets a list of missing fields.

Return type
list[str]

Parameters
instid (*str*)

getversions()

Retrieves the available versions of the Memory Game.

Return type
dict[str, str]

init(*data*)

Initialises a new Memory Game Response.

Return type
str

Parameters
data (*dict[str, Any]*)

iscomplete(*instid*)

Checks whether a Response is complete.

Return type
bool

Parameters
instid (*str*)

load_version(*instid*, *sections*)

Load specified sections of a Memory Game version implementation.

Return type
dict[str, dict[str, Any]]

Parameters

- **instid** (*str*)
- **sections** (*list[str]*)

setscores(*instid*, *data*)

Adds Memory Game Scores to a Response.

Return type
str

Parameters

- **instid** (*str*)
- **data** (*dict[str, str]*)

store(*instid*)

Submits a (complete) Response for long-term storage.

Return type
bool

Parameters
instid (*str*)

F = TypeVar(F, bound=Callable)

Type: TypeVar

Invariant TypeVar bound to typing.Callable[..., typing.Any].

research_client.memorygame.patterns

Additional validation patterns for Memory Game.

research_client.memorygame.versions

Version implementations and translations for the Memory Game.

_get_versions()

Loads all available Memory Game versions into memory.

Return type
dict[str, dict[str, Any]]

5.1.11 research_client.settings

Package implementing the Settings UI for the LART Research Client.

Functions

expose_to_eel()	Expose the Settings API to Python Eel.

expose_to_eel()

Expose the Settings API to Python Eel.

Modules

research_client.settings.eel	Exposes app configuration to Python Eel as Settings.

research_client.settings.eel

Exposes app configuration to Python Eel as Settings.

Functions

load()	Return current settings and config documentation.
store(settings)	Validate settings, store in config, and save..

_expose(*func*)

Wraps, renames and exposes a function to eel.

Return type
TypeVar(F, bound= Callable[..., Any])

Parameters
func (*F*)

_handleexception(*exc*)

Passes exception to exceptionhandler if defined, otherwise continues raising.

Return type
None

Parameters
exc (*Exception*)

load()

Return current settings and config documentation.

Return type
dict[str, dict[str, Any]]

store(*settings*)

Validate settings, store in config, and save..

Return type
bool

Parameters
settings (*dict[str, dict[str, Any]]*)

F = TypeVar(F, bound=Callable)

Type: TypeVar

Invariant TypeVar bound to typing.Callable[..., typing.Any].

5.1.12 research_client.utils

Utility functions for the LART Research Client app.

Functions

export_backup([filename])	Export app data as a ZIP archive.
manage_settings(command)	Manage app settings file.
show_error_dialog([title, message])	Display a graphical error message box even if eel is not active.

_recursively_overwrite_attr(*obj*, *attr*, *value*)

Return type
bool

Parameters

- **obj** (*object*)
- **attr** (*str*)
- **value** (*Any*)

export_backup(*filename=None*)

Export app data as a ZIP archive. Prompt for filename if needed.

Return type
bool

Parameters
filename (*Optional[Union[Path, str]]*)

manage_settings(*command*)

Manage app settings file.

Parameters
command (Union[Literal['update', 'reset', 'clear'], str]) – One of the operations to be carried out on the settings file, or a JSON string with key-value pairs to be merged into the current settings for the app.

The following commands are available by keyword:

- ***update*: Load app settings from current file and save them**
  again. This is useful if a settings file may not include all the key-value pairs that a user may want to control, for instance after an app update.
- ***reset*: Reset the settings file to the hard-coded app**
  defaults. This is useful in cases where a settings file may have become corrupted and where the user wants to start afresh with manually edit the local settings. Effectively, this is the same as using *clear* followed by *update*.
- ***clear*: Remove the settings file. On the next start-up, the**
  app will then use the hard-coded app defaults. This is useful in cases where the user wants to revert to the apps default settings, without the intent to make manual changes.

Return type
`bool`

`show_error_dialog`(*title=None*, *message=None*)

Display a graphical error message box even if eel is not active.

Parameters

- **`title`** (*`Optional[str]`*)
- **`message`** (*`Optional[str]`*)

5.2 Frontend API (JavaScript)

Caution: The included JavaScript API documentation is generated semi-automatically with sphinx-js, which unfortunely doesn't support the full set of jsdoc directives. It can therefore lack detail and/or be out of sync. If in doubt we recommend running `jsdoc` on the individual JavaScript files yourself to generate more detailed API documentation.

5.2.1 lart.js

L'ART Research Client JavaScript Library.

This library implements interfaces and utility functions designed to simplify common tasks for the L'ART Research Client frontend. It's designed to work together with Python eel and booteel.js.

Namespaces

lart.appLock

App locking state management.

The `lart.appLock` namespace provides functionality for the management of the app's global lock state. The lock state is used to optionally enable or disable certain functionality in the UI, such as the user's ability to open the right-click context menu. Generally, the functionality that is made dependent on the lock state should only include that functionality which may be inadvertently used by a user during a task that could corrupt the responses collected for that task (e.g. by right clicking they could reload, resubmit, inspect the source logic, etc.).

Attributes

lart.appLock.**state**

type: string

The app's current global lock state.

This will be either the string 'locked' or the string 'unlocked'.

You should never set the app's lock state manually by manipulating this variable. Instead use the {@link lart.appLock.lock} and {@link lart.appLock.unlock} functions to set the app's lock state.

lart.appLock.**switches**

type: Set

Set holding references to HTMLElements that should reflect the app's global lock state.

Use {@link lart.appLock.registerSwitch} to register HTMLElements that should be switched along with the app's global lock state.

Functions

static lart.appLock.**_contextMenuHandler**(*event*)

Simple event handler to prevent default behaviour when the context menu is triggered.

Arguments

- **event** (*Event*) – The context menu triggering event.

static lart.appLock.**_setSwitchState**(*element*)

Sets the innerHTML of an HTMLElement according to the current switch state.

Arguments

- **element** (*HTMLElement*)

static lart.appLock.**lock**()

Set app's lock state to *locked*.

Returns
null –

static lart.appLock.**registerSwitch**(*switchElementOrId*, *eventType*)

Register an HTMLElement to be switched over on changes to the app's global lock state.

Arguments

- **switchElementOrId** (*HTMLElement*|*string*)
- **eventType** (*string*)

Returns
null –

static lart.appLock.**toggleState**()

Toggle the app's lock status, irrespective of its current state.

Calling this function will set the app's global lock state to *unlocked* if it is currently *locked*, and it will set it to *locked* if it is currently *unlocked*.

Returns
null –

static lart.appLock.**unlock**()

Set app's global state to *unlocked*.

Returns
null –

lart.forms

Form management utilities for the L'ART Research Client.

This namespace implements an extensive set of utility and helper functions which facilitate the implementation of forms for the L'ART Research Client.

Types

lart.forms.**HTMLFormControlElement**

type: HTMLInputElement|HTMLSelectElement|HTMLTextAreaElement|RadioNodeList|HTMLMeterElement|HTMLProgressE

A HTML form control element.

See also:

- {@link lart.forms.HTMLFormControlElementTypes}
- {@link lart.forms.isHTMLFormControlElement}

Attributes

lart.forms.**HTMLFormControlElementTypes**

type: Set

Set containing all the types recognised as {@link lart.forms.HTMLFormControlElement}.

See also:

- {@link lart.forms.HTMLFormControlElement}
- {@link lart.forms.isHTMLFormControlElement}

lart.forms.**_repeatBlockCounter**

type: object

Counter to keep track of the number of repeats of a block.

This counter is used internally by {@link lart.forms.repeatBlock} to keep track of the number of repetitions for a repeated block.

lart.forms.**conditionMatcherCondition**

type: string

Enum of conditions for {@link lart.forms.conditionMatcherFactory}.

Five *condition* values are supported:

- *EQUAL* is equivaluent to *actualValue* == *comparisonValue*.
- *NOT_EQUAL* is equivalent to *actualValue* != *comparisonValue*.
- *SMALLER* is equivalent to *actualValue* < *comparisonValue*.
- *GREATER* is equivalent to *actualValue* > *comparisonValue*.
- *MATCH* is equivalent to *actualValue.match(comparisonValue)*, where *comparisonValue* is a *RegExp*.

Functions

static lart.forms.**autoFill**(*formOrId*, *delay=500*)

Auto-fill a form with data from the URL's query string/search params.

This function will wait for *delay* (in ms) before attempting to fill all HTMLInputElement, HTMLSelectElement, and HTMLTextAreaElement elements attached to the HTMLForm specified by *formOrId*.

If the query string/search params include a field called *${formId}.submit* (where *formId* is the HTMLFormElement's *id* attribute) and its value is either the string *true* or *1*, then following another wait for *delay* ms an Event of type *click* is triggered on the first HTMLElement with a *type="submit"* attribute attached to the HTMLForm.

Arguments

- **formOrId** (*HTMLFormElement|string*) – A HTMLFormElement or a string which identifies a HTMLFormElement by its *id* attribute.
- **delay** (*number*) – The delay in milliseconds before autoFilling the form, and if applicable, again before submitting it.

Returns

null –

static lart.forms.**conditionMatcherFactory**(*controls*, *comparisonValue*, *condition*)

Condition matching function factory.

Generates a function which checks whether the value of one or more form fields (specified by the *nodes* argument) evaluates to *true* or *false* compared to *comparisonValue* under *condition*.

comparisonValue should be a string or number for all *condition* types *except* {@linkcode lart.forms.conditionMatcherCondition.MATCH MATCH}, for which a RegExp object should be supplied.

If the *condition* is {@linkcode lart.forms.conditionMatcherCondition.MATCH MATCH} and the *comparisonValue* is not a *RegExp* object, it will be implicitly converted (i.e. *comparisonValue = RegExp(comparisonValue);*).

For the supported *condition* types, see {@link lart.forms.conditionMatcherCondition}.

Where *controls* is a HTMLForm or an iterable of more than a single HTMLElement, the functions is true whenever *any* of the HTMLFormControlElement elements in *nodes* satisfy the test condition.

Arguments

- **controls** (*HTMLFormElement|HTMLFormControlElement|Array.<HTMLFormControlElement>|Set.<HTMLFormControlElement>|HTMLCollection|NodeList*) – The HTMLElement or NodeList of HTMLElement targets for which the generated function should check the provided condition.
- **comparisonValue** (*String|Number|RegExp*) – The value to which the target should be compared.
- **condition** (lart.forms.conditionMatcherCondition) – The type of condition to be applied in comparing the actual value of the node(s) to *comparisonValue*.

Returns

lart.forms.conditionMatcherFactory~conditionMatcher – Returns a function taking no arguments, which returns *true* when the specified *condition* is met on at least one of the *controls*, and *false* otherwise.

See also:

- {@link lart.forms.conditionMatcherCondition}

static lart.forms.**conditionalDisable**(*observedControlName*, *targetElementOrId*, *value*, *condition*)

Conditionally disable an element depending on the value of a form control.

Observe and conditionally set the disabled attribute on a HTMLElement depending on the value of a HTMLFieldControlElement.

Arguments

- **observedControlName** (*string*) – The name of the {@link lart.forms.HTMLFormControlElement} whose value shall be observed.
- **targetElementOrId** (*string*) – The HTMLElement (or a string with its *id* attribute) which shall be conditionally disabled.
- **value** (*string*) – The value of the observed field at which the target element shall be disabled.
- **condition** (lart.forms.conditionMatcherCondition) – The condition used to compare *value* with the observed field's value.

Returns

null –

static lart.forms.**conditionalDisplay**(*observedControlName*, *targetElementOrId*, *value*, *condition*)

Conditionally display an element depending on the value of a form control.

Observe and conditionally set the display property on an HTMLElement depending on the value of an {@link lart.forms.HTMLFormControlElement}.

Arguments

- **observedControlName** (*string*) – The name of the {@link lart.forms.HTMLFormControlElement} whose value shall be observed.
- **targetElementOrId** (*string*) – The HTMLElement (or a string with its *id* attribute) which shall be conditionally displayed.
- **value** (*string*) – The value of the observed field at which the target element shall be displayed.
- **condition** (lart.forms.conditionMatcherCondition) – The condition used to compare *value* with the observed field's value.

Returns

null –

static lart.forms.**conditionalRequire**(*observedControlName*, *targetControlName*, *value*, *condition*)

Conditionally require a form control depending on the value of a different control.

Observe and conditionally set the *required* attribute on a HTMLFieldControlElement depending on the value of a different HTMLFieldControlElement.

Arguments

- **observedControlName** (*string*) – The name of the {@link lart.forms.HTMLFormControlElement} whose value shall be observed.
- **targetControlName** (*string*) – The name of the {@link lart.forms.HTMLFormControlElement} which shall be conditionally required.
- **value** (*string*) – The value of the observed field at which the target element shall be marked as *required*.

- **condition** (lart.forms.conditionMatcherCondition) – The condition used to compare *value* with the observed field's value.

Returns
null –

static lart.forms.**getControlValue**(*controlElement*)

Obtain the value of any {@link lart.forms.HTMLFormControlElement}.

Arguments

- **controlElement** (lart.forms.HTMLFormControlElement) – The form control element for which the value should be optained.

Returns
string|Array.<string>|null – The value of the passed form control element, if any. This will be a string for control elements with a single value (e.g. text input), an array of strings for those with multiple values (e.g. multiselect, checkboxes), or *null* if no value is set for the targeted {@link lart.forms.HTMLFormControlElement}.

static lart.forms.**getElementByGreed**(*ref*, *root=document*)

Greedily and flexibly try to get an element from the DOM.

This function allows for the flexible retrieval of an element from the DOM. It is primarily meant to be called inside other functions needing a reference to an element and there facilitates a more flexible calling pattern to those functions, allowing the element to be referenced either directly as a HTMLElement or RadioNodeList, or by its *id* or *name* attribute.

The procedure followed to find an element is as follows:

- If the passed argument is a HTMLElement or RadioNodeList, return it unchanged.
- **If the passed argument is a string:**
  - If *root* implements *.getElementById*, call *root.getElementById(ref)* and return the HTMLElement, if any.
  - If *root* implements *.getElementsByName*, call *root.getElementsByName(ref)*. If the returned NodeList contains only HTMLInputElements of type radio which have both the same name and belong to the same HTMLForm (i.e. that form a radio group), then obtain the relevant RadioNodeList and return it. Otherwise, if the NodeList contains only a single HTMLElement, return it.
  - If *root* implements *.querySelector*, call *root.querySelector(ref)* and return the HTMLElement, if any. Note that this will be the *first* element inside *root* which satisfies the *ref* passed to *querySelector*.
- If no HTMLElement or RadioNodeList could be found following the above procedure, return *null*.

The *root* argument is optional, and can be either a Document root or a HTMLElement node to be used as the root. Note that *HTMLElements* don't implement all the supported query methods and will typically default to the HTMLElement.querySelector method. Where *root* is not specified, it defaults to the global *document* object. The *root* is not used where the *ref* argument already is a *HTMLElement* or *RadioNodeList*.

Arguments

- **ref** (*HTMLElement|RadioNodeList|string*) – A reference to the HTMLElement which should be obtained from the DOM. Can be either a JavaScript object directly representing it, or a string which identifies the element via its *id* or *name* attribute, or a string which identifies the element using the *querySelector* syntax.
- **root** (*Document|HTMLElement*) – A Document or HTMLElement to be used as the root for querying. Ignored if *ref* already is a HTMLElement or RadioNodeList.

Returns

HTMLElement|RadioNodeList|null – Returns a single *HTMLElement*, a *RadioNodeList*, or *null*, depending on the first suitable item found in the DOM according to the algorithm described above.

`static` lart.forms.**getFormControlElements**(*formOrId*)

Get a pre-processed Set of a form's control elements.

Convenience function, which obtains a HTMLFormElement's {@link lart.forms.HTMLFormControlElement}s. In contrast to a HTMLFormControlsCollection, this function returns a *Set* in which each control is only present once, and where HTMLInputElements of type *radio* are represented by a RadioNodeList, rather than the individual HTMLInputElements themselves. Additionallly, HTMLFieldSetElements have been removed.

Arguments

- **formOrId** (*HTMLFormElement|string*) – The HTMLFormElement (or a string with its *id* attribute) for which the HTMLFormControlCollection should be obtained.

Throws

TypeError – Throws a TypeError if *formOrId* does not refer to a HTMLFormElement.

Returns

Set.<lart.forms.HTMLFormControlElement> – Returns a Set of form control elements representing unique instances of form control elements in the targeted form.

`static` lart.forms.**getFormData**(*formElementOrId*)

Assemble all data from a specified form into an object of key-value pairs.

Arguments

- **formElementOrId** (*HTMLFormElement|string*) – A form element to extract data from.

Returns

object – - Returns a dictionary-like object of key-value pairs, where key is the name (or *id* as fallback) and value the value of each data field within the specified form.

`static` lart.forms.**getRadioNodeList**(*radioElement*)

Get the RadioNodeList associated with a radio input control.

Arguments

- **radioElement** (*HTMLInputElement*) – The radio input control for which the RadioNodeList should be obtained.

Throws

TypeError – Throws *TypeError* if the passed element is not a radio element, not attached to a form, or doesn't have a *name* attribute.

Returns

RadioNodeList –

`static` lart.forms.**getSelectValues**(*selectElement*)

Obtain the selected option values of an HTMLSelectElement.

Arguments

- **selectElement** (*HTMLSelectElement|string*) – The HTMLSelectElement whose values shall be optained, or a string with the HTMLSelectElement's id or name.

Throws

TypeError – Throws a TypeError if *elementOrId* does not refer to an HTMLSelectElement.

Returns
Array.<string> – A list of values of all selected options inside the select element.

`static` lart.forms.**isHTMLFormControlElement**(*element*)

Check whether an element is a {@link lart.forms.HTMLFormControlElement}.

Arguments

- **element** (*object*) – The object to be checked for implementation of a relevant prototype.

Returns
boolean – Returns *true* if the object is a {@link lart.forms.HTMLFormControlElement}, *false* otherwise.

See also:

- {@link lart.formsHTMLFormControlElement}
- {@link lart.forms.HTMLFormControlElementTypes}

`static` lart.forms.**isHTMLRadioInputElement**(*element*)

Check whether a HTMLElement is a radio input control.

Arguments

- **element** (*HTMLElement*) – The Element to be checked.

Returns
boolean – Returns *true* if the element is a HTMLInputElement of type 'radio', *false* otherwise.

`static` lart.forms.**pipeData**(*event*, *receiver*)

Form submission event callback to validate a form and pipe data to another function.

Implements the functionality of {@link lart.forms.registerPipeline} following the triggering of a *submit* Event on the targeted form. See the documentation there for details of the behaviour.

Can also be called directly rather than as an Event callback to simulate the submission of a form. To do this you should create a new Event (typically of type *submit*) attached to a HTMLFormElement, and pass this along with the *receiver*. Note that doing this without actually triggering the event might bypass other registered event listeners.

Arguments

- **event** (*Event*) – An event (typically 'submit') on a form element, e.g. as issued by .addEventListener().
- **receiver** (*function*) – A callable to which the form data will be passed as a dictionary.

Returns
boolean – Returns *true* if the receiver function returns a truthy value, *false* if validation fails or the receiver function returns a falsy value.

`static` lart.forms.**registerPipeline**(*formElementOrId*, *receiver*)

Register a pipeline to submit a form's data to a JavaScript function instead of an HTTP request.

Registers an event handler on the form specified by *formElementOrId* so that the data inside the form will be piped to the function specified by *receiver* and the further propagation of the submission event will be halted. Thus, no HTTP request will be issued and the page with the form won't advance to the specified target or reload; if this is desired then the receiver of the data should manually direct the user to the new page.

Data will only be piped to the *receiver* if the form passes client-side validation via the JavaScript validation API. If you intend to also register a custom function to the submit event of the form to validate (e.g. *requireValidation*),

then you should register the validation function **before** registering the pipeline, so that a failure to pipe an invalid form won't block the validation function callback.

Normally the function as written here will be used to pipe data to a function exposed via Python's *eel* module (marked *@eel.expose* in Python) via a JavaScript wrapper, but it could of course also be used in other scenarios where it is desirable to simply process the submitted data client-side.

After registering a pipeline callback, roughly the following behaviour ensues:

- A *submit* EvenListener is registered on the form.
- Following a *submit* event on the form, the default submittion/event's default behaviour is prevented and propagation of the event is stopped.
- If the form fails to validate (i.e. if *form.checkValidity() == false*), then nothing happens.
- If the form validates, the form's data is obtained via {@lart.forms.getData}.
- *receiver* is called with this data as the only argument.

Arguments

- **formElementOrId** (*HTMLFormElement|String*) – A HTMLFormElement or its *id* for which data is to be piped to *receiver*.
- **receiver** (*function*) – The callback function that should receive the the form data upon submission and passed validation.

static lart.forms.**repeatBlock**(*elementOrId*, *pattern*)

Note: Deprecated: It's better to use HTMLTemplateElements for this functionality.

Repeat a HTMLElement and use a regular expression to replace strings inside with a running counter.

Note: This function is now deprecated and should not be used in implementing any new functionality. Instead use the HTMLTemplateElement together with a slot and then insert copies of the template dynamically as required. This function will be removed from the library once all currently implemented functionality has been transitioned to the use of HTML templates.

Arguments

- **elementOrId** (*HTMLElement|string*) – A HTMLElement or a string with an id or unique name attribute to identify a HTMLElement which shall be repeated. Usually this is a block-level element.
- **pattern** (*RegExp*) – A regular expression pattern to match which will be appended with a counter for each repetition of the block.

Returns

null –

static lart.forms.**requireValidation**(*novalidate*)

Require all forms marked *.needs-validation* to pass client-side validation before submitting.

This function will register an event on all forms with the CSS class *.needs-validation* which prevents submission if there are any invalid fields according to the JavaScript Validation API. Forms will be marked by adding the class *.was-validated* after the first attempt to submit, which will enable Bootstrap to show custom user feedback messages.

If the option *novalidate* is set to *true* (default *false*) then the function will automatically set the attribute *novalidate* on all the forms it attaches to. This is needed to display Bootstrap form validation feedback instead of the

browser's built-in feedback, so should usually be specified *true* where Bootstrap-type user feedback messages are provided.

Arguments

- **novalidate** (*bool*) – Whether to mark affected forms 'novalidate' to suppress browsers' built-in feedback.

Returns
null –

`static` lart.forms.**validateRangeInputs**(*targetZoneElementOrId*)

Add validation to HTMLInputElements of type *range*.

Attach a validation observer to all HTMLInputElements of type *range* which are children of the HTMLElement indicated by *targetZoneElementOrId* and which have the *required* attribute set.

The observed range inputs will pass (client-side) validation only if their slider has been moved at least once, irrespective of their value.

The method employed to do this involves setting a custom data attribute (*data-lart-range-moved*) on the HTMLInputElement and checking this as part of the elements custom validation. An event listener is attached to the element to monitor for input and adjust the data attribute accordingly. A MutatioObserver is further attached to the target zone to monitor for the insertion of any further range controls.

Arguments

- **targetZoneElementOrId** (*HTMLElement|String*) – A HTMLElement, or a string referring to an *id* or *name* attribute identifying an HTMLElement which is the parent of the range input controls to be validated.

Returns
null –

lart.tr

On-the-fly client-side translation management for the L'ART Research Client.

This namespace implements functionality to facilitate the on-the-fly translation of user interface elements with strings loaded on-demand from the backend.

Attributes

lart.tr.**_activeObservers**

type: object.<string, Set.<string>>

List of Node Id's that are being actively observed for string translation.

lart.tr.**_callbackQueue**

type: object.<string, Array.<function()>>

Queue of callbacks waiting to be called once a certain translation namespace becomes available.

lart.tr.**_observerQueue**

type: object.<string, Set.<string>>

List of Node Id's that should be observed for string translation after loading strings for a namespace.

lart.tr.**missing**

type: object.<string, object.<string, string>>

Dictionary-like object of translation IDs and innerHTML content for missing translation items.

You should not normally access the missing translation objects directly, but rather call {@link lart.tr.getMissing} to obtain the missing translation IDs and associated innerHTML for a specific translation namespace. To add missing translation objects use {@link lart.tr.addMissing}

See also:

- {@link lart.tr.getMissing}
- {@link lart.tr.addMissing}

lart.tr.**strings**

type: object.<string, object.<string, object.<string, string>>>

Dictionary-like object holding translation identifiers and translations.

You should not normally access the translation strings directly, but rather call {@link lart.tr.get} to access the translation strings, as this implements additional logic and will be stable even if the internal structure of the string object should change in the future.

See also:

- {@link lart.tr.get}

Functions

static lart.tr.**_activateObservers**(*ns*)

Activate translation observers currently held in the queue.

Cycles through queued Node Id's for translation observation. If the translation strings have been loaded already cycles through the nodes to translate them and then registers a MutationObserver on the node to monitor for changes. If translation strings are not available yet a timeout is set for 100ms, and translation and observer registration is done once the translation strings have been loaded.

Arguments

- **ns** (*string*) – The namespace for which observers should be activated.

Returns

null –

static lart.tr.**_addStrings**(*ns*, *strings*)

Add translation strings from array to {@link lart.tr.strings}.

Arguments

- **ns** (*string*) – The translation namespace to add the strings to.
- **strings** (*object.<string, string>*) – An associative array with translation strings labelled by section. Each section contains an associative array with a translation-string id, and a list of [1] the original untranslated string, and [2] the version-specific translation/adaptation.

Returns

null –

static lart.tr.**_translateElement**(*ns*, *element*)

Check, and if applicable, substitute a HTMLElement's innerHTML with version-specific string.

Arguments

- **ns** (*string*) – The translation namespace to be used.
- **element** (*HTMLElement*) – the HTMLElement to apply translation to.

Returns
null –

static lart.tr.**_triggerCallbacks**()

Trigger waiting callbacks once a translation space has become available.

Returns
null –

static lart.tr.**addMissing**(*ns*, *trId*, *innerHTML="'???'"*)

Add a missing translation IDs and associated innerHTML to the missing trId cache.

Arguments

- **ns** (*string*) – The translation namespace to add the missing trId to.
- **trId** (*string*) – The translation string identifier.
- **innerHTML** (*string*) – The associated innerHTML, default is *'???'*.

Returns
null –

static lart.tr.**get**(*ns*, *trId*)

Get a translation string by its translation ID.

Arguments

- **ns** (*string*) – The translation namespace to get the translation string from.
- **trId** (*string*) – The translation identifier string.

Returns
string – - The translated string for the identifier, or *null* if no string for the *trId* could be found in *ns*.

static lart.tr.**getMissing**(*ns*)

Get the missing translation IDs and associated innerHTML for missing translation items.

Results are returned as a JSON template, so that they can be copied easily into existing JSON task version translation files (or indeed used as the basis for a new one).

Arguments

- **ns** (*string*) – The translation namespace to get missing strings for.

Returns
string – Returns a JSON template of the missing translation IDs with their innerHTML.

static lart.tr.**loadFromEel**(*ns*, *eelLoader*, *loaderParams=[]*)

Load translation/adaptation strings into a translation namespace.

Arguments

- **ns** (*string*) – The translation namespace to be used.

- **eelLoader** (*function*) – The Python eel function (from eel.js) implementing the translation loader on the backend.
- **loaderParams** (*Array.<any>*) – The parameters that the *eelLoader* should be called with.

Returns
null –

Examples:

```
<caption>Loads the LSBQe sections 'meta', 'base' and 'lsb' into the 'lsbq'␣
↪namespace:</caption>
const instanceId = lart.utils.searchParams.get('instance');
lart.tr.loadFromEel('lsbq', eel._lsbq_load_version, [instanceId, ['meta', 'base',
↪'lsb']]);
```

static lart.tr.**registerCallback**(*ns*, *callback*, *callbackParams*)

Register a callback to be triggered when translations for a namespace become available.

This can be useful if certain UI functionality should be delayed until the translations for a specific namespace (specified by *ns*) have been loaded from the backend.

If the information depending on the translation is encodable as regular HTML content, it is often preferable to insert the HTML with a *data-*ns*-tr* attribute and use the regular {@lart.tr.registerObserver} method instead, as this will add less overhead where the targeted element or its parent might already be observed for changes.

Arguments

- **ns** (*string*) – The translation namespace to monitor.
- **callback** (*function*) – The callback function to call once the specified translation namespace is available.
- **callbackParams** (*any*) – The parameters to pass back to *callback* when calling it.

Returns
null –

static lart.tr.**registerObserver**(*ns*, *nodeId*)

Register a Node for translation observation for a specific namespace by a Node's Id.

This will add an observer running translation on any *HTMLElement* or *Node* that is a child of the Node specified by *nodeId*. Any child node with a *data-*ns*-tr* attribute specifying the *trId* will be subject to translation from its namespace where a string with a matching *trId* is available.

Observers will automatically delay observation until the relevant namespace has been loaded. There is no need to (or point in) registering them separately as a callback with {@link lart.tr.registerCallback}.

Arguments

- **ns** (*string*) – The namespace that should be observed for translation.
- **nodeId** (*string*) – The Id of the Node (and its children) to be observed.

Returns
null –

static lart.tr.**translateAttrs**(*ns*, *attrs*)

Translate an attribute on one or more elements specified by their id.

If namespace *ns* has not been loaded yet, then translation will automatically be postponed until it is loaded. There is no need to manually register a callback.

Arguments

- **ns** (*string*) – The translation namespace to be used.
- **attrs** (*object.<string, Array.<string, string>>*) – An associative array with Element Ids as key, and a two-member list as values, where the first is the attribute name and the second the translation ID.

Returns
null –

`static` lart.tr.**translateNode**(*ns*, *node*)

Traverse through a DOM Node and translate all applicable HTMLElements.

Normally this will be called automatically after an observer has been registered for a Node or HTMLElement with {@link lart.tr.registerObserver}. However, you can call this manually passing a Node or HTMLElement to trigger a single pass of the string replacement algorithm for the namespace *ns* on that *node* and its children.

Arguments

- **ns** (*string*) – The translation namespace to be translated on the nodes.
- **node** (*Node|HTMLElement*) – A DOM Node to be traversed and translated.

Returns
null –

lart.utils

On-the-fly client-side translation management for the L'ART Research Client.

This namespace implements functionality to facilitate the on-the-fly translation of user interface elements with strings loaded on-demand from the backend.

Namespaces

lart.utils.UUID	Utilities for working with UUIDs.

Attributes

lart.utils.**searchParams**

type: URLSearchParams

Shortcut to the URLSearchParams for the current window location.

See also:

- {@link https://developer.mozilla.org/en-US/docs/Web/API/URLSearchParams MDN Documentation for URLSearchParams}

lart.utils.UUID.**nilUUID**

type: string

Nil UUID as hex-string with separators.

lart.utils.UUID.**pattern**

type: RegExp

RegEx pattern for identifying valid hex-format UUIDs.

lart.utils.UUID.**plainNilUUID**

type: string

Nil UUID as hex-string without separators.

Functions

static lart.utils.**extractLanguageFromVersion**(*version*)

Extract ISO 639-2/3 alpha-3 language code from a version string.

Given an input of the form "XxxYyy_Zzz_CC" where Xxx, Yyy, and Zzz are three-letter ISO 639-2 or ISO 639-3 alpha-3 language codes, and CC is a two-letter country code, this function will return the string Zzz, which is used in the L'ART Research Client to identify the primary display langauage of a test version.

Arguments

- **version** (*string*) – L'ART test version string of the form "XxxYyy_Zzz_CC".

Returns

string|null – Returns three character alpha-3 language code representing the primary display languageof a L'ART test version, or *null* if the supplied string doesn't validly encode one.

static lart.utils.UUID.**isNilUUID**(*identifier*)

Check whether a UUID in hex format is the Nil UUID.

Arguments

- **identifier** (*string*)

Returns

boolean – Returns true if the provided identifier is the *Nil UUID* (~UUID equivalent of *null*), with or without separators, *false* otherwise..

static lart.utils.UUID.**isUUID**(*identifier*)

Check whether a string is a valid UUID in hex format.

Arguments

- **identifier** (*string*)

Returns

boolean – Returns *true* if the provided identifier is a valid hex-formated UUID string (with or without separators), *false* otherwise.

lart	Root namespace for the lart.js library
lart.appLock	App locking state management
lart.forms	Form management
lart.tr	On-the-fly UI translation management
lart.utils	General utility functions

5.2.2 booteel.js

The Booteel Frontend Library.

The Booteel Frontend Library provides functionality to make Python's eel module work and interact nicely with a Bootstrap-based frontend. This JavaScript library implements the frontend side of the Booteel functionality, while the Python module *research_client.booteel* implements the backend functionality.

Note: Unfortunately, booteel.js has thus far been insufficiently annotated to provide API documentation here. We will aim to get this ready in due course - in the meantime please refer to the backend booteel API and inspection of the `booteel.js` file.

---

lart.js	The L'ART JavaScript Library
booteel.js	The Booteel Frontend Library

CHAPTER

SIX

REFERENCES

BIBLIOGRAPHY

[Anderson-Mak-EtAl-2018] Anderson, J.A.E., Mak, L., Keyvani Chahi, A. & Bialystok, E. (2018). The language and social background questionnaire: Assessing degree of bilingualism in a diverse population. *Behaviour Research Methods* 50, 250–263. https://doi.org/10.3758/s13428-017-0867-9

[Lambert-Hodsgon-EtAl-1960] Lambert, W. E., Hodgson, R. C., Gardner, R. C. & Fillenbaum, S. (1960). Evaluational reactions to spoken languages. *The Journal of Abnormal and Social Psychology* 60(4), 44.

[Markel-EtAl-1967] Markel, N. N., Eisler, R. M. & Reese, H. W. (1967). Judging personality from dialect. *Journal of Verbal Learning and Verbal Behavior* 6(1), 33–35.

[Schoel-Roessel-EtAl-2013] Schoel, C., Roessel, J., Eck, J., Janssen, J., Petrovic, B., Rothe, A., Rudert, S.C. and Stahlberg, D. (2013). Attitudes To wards Languages" (AToL) Scale: A Global Instrument. *Journal of Language and Social Psychology*, 32(1), 21–45.

[Breit-Tamburelli-EtAl-2023] Breit, F., Tamburelli, M., Gruffydd, I., Brasca, L. (2023). The L'ART Research Client: A digital toolkit for bilingualism and language attitude research. Bangor University.

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

M

N

P

R

S

T

V

X

Y