LMEDS 2.4 Instruction Manual

Tim Mahrt

November 2, 2016

It is my hope that you will find this document useful in working with LMEDS. This document contains information for all aspects of LMEDS, from installation and setup, to creating experiments, running experiments, and obtaining and analyzing test results. If you have any questions, comments, or other feedback, please contact me at timmahrt@gmail.com

Special thanks goes to those who used early versions of LMEDS and provided critical feedback, feature requests, and bug reports (in alphabetical order):

Stefan Baumann
Jennifer Cole
Núria Esteve Gibert
Martine Grice
Jonathan Howell
José I. Hualde
Suyeon Im
Preethi Jyothi
Janina Kalbertodt
Amelia Kimball
Caroline Smith

and the more than 1000 people on Mechanical Turk who have participated in experiments using LMEDS!

Development of LMEDS was possible thanks to

- NSF grant BCS 12-51343 to Jennifer Cole, José I. Hualde, and Caroline Smith
- A*MIDEX project (n° ANR-11-IDEX-0001-02) to James Sneed German funded by the Investissements d'Avenir French Government program, managed by the French National Research Agency (ANR).

Contents

1 Orientation

LMEDS comes packaged with the following folders:

- cgi-bin

  Small code snippets (with the extension .cgi) go here–one for each experiment
- html

  Contains static HTML, Javascript, and CSS files
- imgs

  Image files used by LMEDS
- lmeds

  The main repository for code.
- lmeds/user_scripts

  Within the lmeds code directory are some scripts that users can use in setting up their experiments and in post-processing their data.
- tests

  Holds the resource files for each experiment
- user_manual

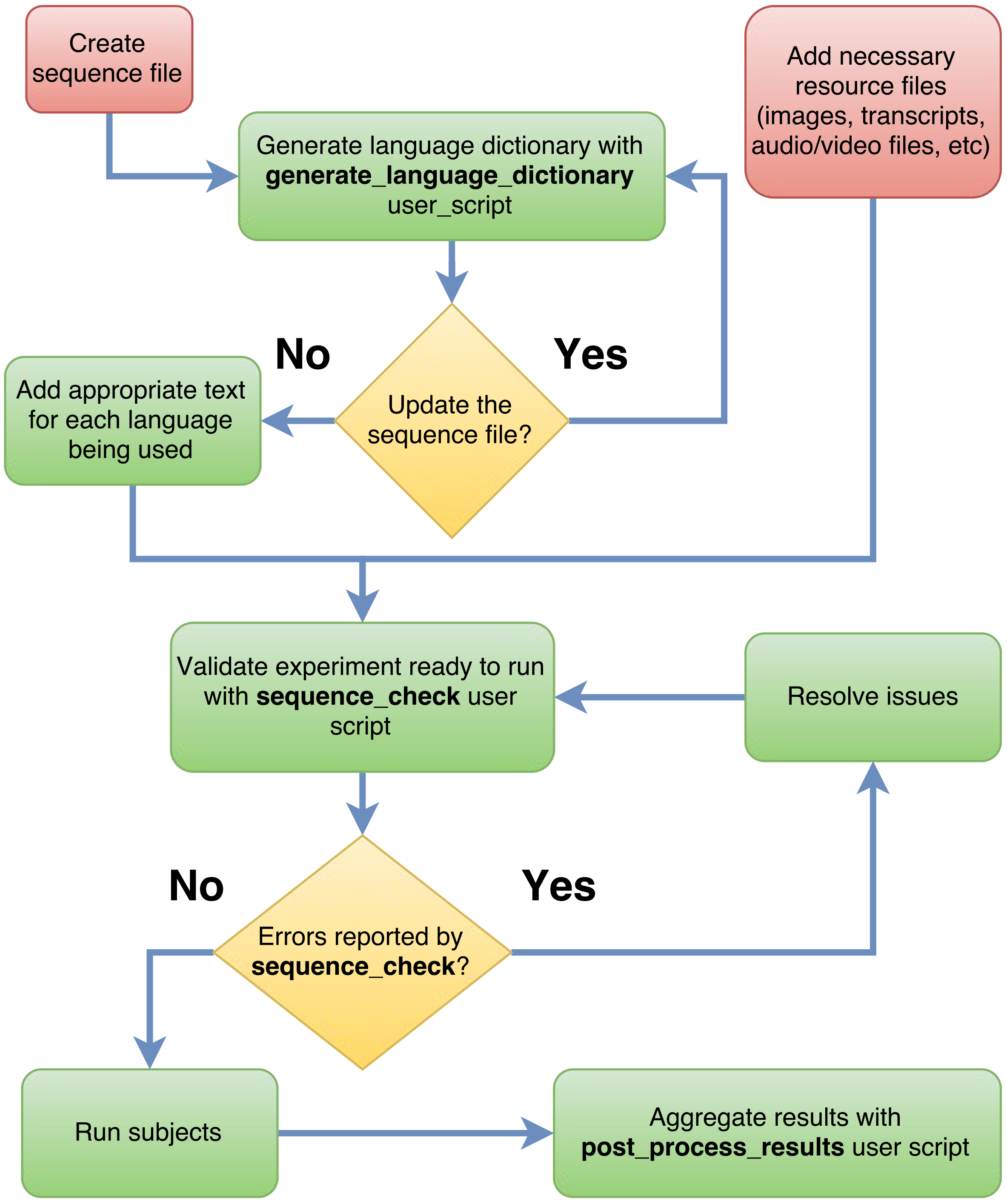
  Contains the user manual

If you are creating an experiment, you'll need to add one file to cgi-bin and you'll need to add a folder to tests, containing all of the resource files used in the experiment–the other folders can all be ignored. The next section goes into the details of how to create the

1.1 Getting started with LMEDS

This manual describes how to create files used by LMEDS from scratch. You might find it useful to reference the experiment files that come with LMEDS, located in /tests/lmeds_demo. In it, you'll find an example dictionary (english.txt), sequence file (sequence.txt), input files contained in the (audio/ and txt/ folders), survey files (presurvey.txt and postsurvey.txt), and some sample output files (stored in the folder output/). **When you are starting a new experiment from scratch, you might find it useful to start with the demo files provided.**

1.2 LMEDS Workflow Schematic


Create sequence file
Generate language dictionary with **generate_language_dictionary** user_script
Add necessary resource files (images, transcripts, audio/video files, etc)
No
Yes
Update the sequence file?
Add appropriate text for each language being used
Validate experiment ready to run with **sequence_check** user script
Resolve issues
No
Yes
Errors reported by **sequence_check**?
Run subjects
Aggregate results with **post_process_results** user script

To begin an LMEDS experiment, two things are needed–resource files and a sequence file (the two boxes in red). Once the sequence file is sufficient for your purposes, a language dictionary is generated from it with the user script generate_language_dictionary. If any changes are made to the sequence file, the language dictionary will need to be regenerated. Once the sequence file and language dictionary are syncronized, enter the appropriate text for each language being used.

With the sequence files, resource files, and generated language dictionary, validate that the experiment can be run using sequence_check. sequence_check will ensure that all resources and text specified in the sequence file are available and that each page in the sequence file can be created. As long as there are issues, resolve them and run the script again. Once there are no reported errors, test it for yourself and then run subjects. After collecting all subject data, use post_process_results to aggregate the user data.

2 Notices and warnings

2.1 Creating and editing text files

The input and outputs to LMEDS shouldn't be edited in a word processor. A plain text editor should be used. The text editors that come with most operating systems (notepad on Windows or Text Edit on mac) should be suitable however, you may want a more developed plain text editor. I have personal experience with NotePad++, Gedit, and TextWrangler, which are all freely available tools. If you search online, you can find reviews for editors that might suit you better.

Specifically, a plain text editor should be used for the following data types:

.txt .csv .cgi .py .rst

The output **.csv** files can also be opened in spreadsheet viewers like Excel or statistical analysis tools like R.

2.2 No spaces in names

Please do not include spaces in folder or file names. If you have a file composed of multiple words, either do not use spaces or use underscores. E.g. for a folder that you want to call "lmeds demo", reasonable names include "lmeds_demo" or "lmedsDemo" or "lmedsdemo".

2.3 Audio and video file formats

Users can specify the audio and video formats they wish LMEDS to use (see Section **??**). If none are specified, LMEDS will default to certain formats. Regardless, for any resources in a sequence file, experiment administrators **must** provide copies of each resource file for all the specified formats.

Suppose the user specifies in the .cgi file that they want to use .wav, .ogg, and .mp3 files. If there is an audio file named sound1.wav, there must also be a corresponding sound1.wav and sound1.mp3. If these files do not exist, LMEDS will ungracefully crash when it is time to load sound1.wav, even if the experiment participant can load sound1.wav and doesn't need the other two files.

2.4 Errors in LMEDS

LMEDS was written in python. If something goes wrong (an unexpected or anticipated error) python will print out a formatted error report called a **stack trace**. Unintuitively, the direct cause of the error is printed at the **bottom** of this report.

While the stack trace can be difficult to understand for non-programmers, usually the direct cause of the error is printed as reasonably understandable English. For example the last line in the stack trace might be *ERROR: Folder, ../tests/lmeds_demo, does not exist* or it might be *Text key axb_instructions not in dictionary file english.txt. Please add text key to dictionary and try again.'*

Errors are common when designing a new experiment (although the included script /lmeds/user_scripts/sequence_check.py can be used to automatically find common errors like missing resource files or incorrectly specified page definitions). If an error occurs, don't panic. Consult the error log as specified above. If you are unsure about the error, please send it to me.

3 Creating your own experiments

3.1 User-defined test components

An LMEDS test has the following components

3.1.1 Audio and video files

An audio file can be any common type (.mp3, .wav, etc.) and a video file can be one of (.mp4, .ogg, or .webm). Some web browsers are unable to play audio in certain file types. Personally, I encode every audio file as both .mp3 and .ogg and every video file as .mp4 and .ogg. I recommend encoding each file in two different formats. If a user is unable to load one file format, their browser will automatically load the other file format.

See the following pages for more information on browser compatibility:
http://www.w3schools.com/tags/tag_audio.asp
http://www.w3schools.com/html/html5_video.asp

For audio files, audacity makes it very easy to convert entire directories of audio files into multiple audio formats with it's 'chain' functionality.
http://audacityteam.org/

To save server space and speed up loading times for users with slower internet, I reencode my wav files to 16,000 HZ before I convert them to .mp3 and .ogg. This step is not required. Sox is a freely available command line tool that can do this task easily:

```
sox input_file_name -r 16000 output_file_name rate -v 96k
```

http://sox.sourceforge.net/

3.1.2 Transcription files

Only raw txt files with the extension .txt are accepted. Each transcription file should contain the transcript for one stimulus.

For long excerpts, you should specify the line breaks explicitly by chunking the text into pieces over the span of several lines. For a long excerpt, if all of the text lies on one line, the text will run off the page in LMEDS.

3.1.3 Sequence file (see section ??)

The file that specifies the control flow of a test. It specifies what instructions and stimuli users will be presented with in a test and in what order.

Multiple sequence files can exist for a set of audio and transcription files (e.g. one could have a sequence with the stimuli in a certain order and another sequence with the stimuli in a different order or with different instructions, etc.).

3.1.4 Dictionary file(s) (see section ??)

The file that contains all text that will be seen by users in your experiment. This file is independent of a sequence file so one could write a sequence and present it in both English and Russian, for example.

3.1.5 Survey file(s) (see section ??)

A survey file can be used to get various kinds of feedback, such as might be needed for a longform survey or a short-answer question. Supports text boxes, checkboxes, and radio buttons.

Each survey requires its own survey file (e.g. presurvey.txt, postsurvey.txt).

Most pages have a static layout (defined in the python code) that is filled in with the same type of content. A survey is different because the amount of content and inputs vary with each survey page, whereas all AXB pages, for example, share the same number of inputs and outputs. For this reason, each survey requires its own specification file while other types of pages do not.

3.1.6 .CGI file

The file that defines a single experiment–via a sequence file and a language file. The name given to this file (e.g. lmeds_demo.cgi) is the name that participants would use to access the experiment in the URL.

A template of a typical .cgi file can be found in section **??**. An example file is also distributed in the /cgi folder included with LMEDS.

3.2 Dictionary file specification

LMEDS uses **dictionaries** for multi-lingual support. These dictionaries contain a set of keys where each key has text associated with it. One dictionary might have text only in English, while another only in French, but the keys are the same in both. Thus, when a key is used in a page, LMEDS looks in the dictionary (that is appropriate for the current experiment) for the corresponding text for that key. E.g. the key “dictionary” will fetch the text “Dictionary” for English and “Dictionnaire” for French. Each dictionary file is created by the user.

Some keys are defined within the python code. Other keys are user-defined. The user_script **generate_language_dictionary.py** can be used to generate a template dictionary file (all keys are present but there are no text values) or update an existing dictionary file with new keys.

You might find it useful to create a new dictionary by using another one–either if you’re starting a new experiment or translating your current experiment into another language.

A dictionary files has the following components (a very small, but complete dictionary file follows these, alternatively, you can look at the included dictionary file for a real example: **/tests/lmeds_demo/english.txt**):

3.2.1 Sections

A section is denoted with “-” characters on the line above and below the section name. The lengths of these lines (the number of “-”) doesn’t matter but should be kept the same length to be visually more coherent.

Sections are actually ignored in the code. Their only purpose is to help structure dictionaries, perhaps most naturally by page type. Keys shared by multiple pages are kept in their own sections.

3.2.2 Keys

A key is denoted in the same way with “=” characters. Keys must be the same, regardless of the language.

3.2.3 Texts

A “text” appears after a key and before another key or section. Unlike sections and keys, text is undecorated text.

Text is rendered in HTML. This means HTML markup is allowed. It also means that whitespace other than a single space is ignored, although you may use extra whitespace to make it easier to read the text in the dictionary. If you want a line break, for example, you will need to insert one used `<br />`

3.2.4 Dictionary example file

Here is a short, concrete example of a dictionary file

```
_______________
text_page
_______________


===============
title
===============

Perception of Spoken Discourse

===============
first_block_finished
===============

You have completed the first block.<br /><br />

Please feel free to rest for a minute before continuing.
```

The section denotes that these keys are used in the text pages, which tend to be used for giving instructions. Whenever LMEDS needs to display the title, it will look at the text for the key "title" in this case "Perception of Spoken Discourse".

3.3 Survey file specification

The survey format allows for the simple creation of short or long questionnaires. The format is simple–the first line in a survey question contains the text prompt, subsequent lines specify one or more data entry fields that users can use to answer the question, and, finally, a blank line signals that the current item is complete. An small example survey file can be found is at the bottom of this section, two surveys are also bundled in lmeds **/tests/lmeds_demo/presurvey.txt** and **/tests/lmeds_demo/postsurvey.txt**.

A note on arguments: For data entry fields that have arguments, arguments should be separated from the data entry field name by a space and from each other by a comma

e.g.

```
Choice English, Arabic, Other
```

Here is the list of data entry fields available:

3.3.1 None

There are no inputs the user can choose from. This is used to include instructions on the page.

e.g.

```
Please answer the questions below.
None
```

3.3.2 Choice

Users can select exactly one item out of many.

e.g.

```
Sex:
Choice Male, Female
```

3.3.3 Item_List

Users can select as many items as they want out of many.

e.g.

```
Indicate the language(s) that you are familiar with.
Item_List English, French, Spanish, Italian, German
```

3.3.4 Choicebox

A dropdown box. Users can select one item from many. Similar to a **Choice** but is more efficient in space for large lists of items.

e.g.

```
Level of education completed:
Choicebox High School, Some College, Bachelor's Degree
```

3.3.5 Sliding_Scale

A sliding scale. Specify the numerical scale (which controls the granularity of the user's choice) and the labels for the two extremes of the scales.

e.g.

```
How satisfied were you with this experiment?
Sliding_Scale 0, 100, Very Unsatisfied, Very Satisfied
```

3.3.6 Textbox

A single line for users to enter a small amount of information. A textbox takes no arguments.

e.g.

```
Occupation
Textbox
```

3.3.7 Multiline_Textbox

A longer-form textbox that can span several lines. It takes exactly two arguments: the number of characters across and the number of lines.

e.g.

```
What did you think of this test?  Please provide any feedback.
Multiline_Textbox 50, 7
```

3.3.8 Sublists

It is possible to designate a group of items as subquestions of the previous question. These items will be displayed tabbed. To do this, encapsulate the relevant items in the sublist tag:

```
<sublist>
Occupation
Textbox

City of birth
Textbox
</sublist>
```

3.3.9 Survey file example

Here is a short concrete example demonstrating the key features discussed above

```
Please answer the questions below.
None

Sex:
Choice Male, Female

Age:
Textbox

Country of birth:
Choice United States, Other
Textbox

<sublist>
If United States, list city/state:
Textbox

If other, how old were you when you moved to the USA?
Textbox
</sublist>
```

Thus we have a survey with (ignoring the first item, which doesn't take user input) 5 items in total (2 being subitems). The third item shows how multiple data entry fields can be placed on a single question (if the user selects “other” for the first question they are expected to specify what they meant in the Textbox).

3.4 Sequence file specification

The sequence file contains the items that will be presented in a test. The first line in a test is the test name which should be prefixed with a “*” (e.g. *My_Test_Sequence) (**data from the experiment will be output to a folder with the test name. Giving multiple sequence files the same name, means that they will dump their output to the same folder**). The second item must be “login” where users create a name to associate their data with. If login is not the item on the second line, unexpected behavior can result. Subsequent items are presented in a linear fashion as users progress through the sequence. The last item in the sequence must be “end”.

Most pages take arguments. These arguments specify the behavior the page should take (which audio files to play, instructions to present, etc.) Arguments are separated from one another and from the page type by a space. Audio files and textfiles included in an argument should not include their file extension (.txt, .wav, etc.)–LMEDS will determine the appropriate extension to be used.

```
Prominence water water 1 3 true
```

The above example is the entry for the prominence page. It comes with 5 arguments. Sections **??** and **??** include information on how to understand the individual arguments to a page. A full example sequence file can be found in section **??**.

3.4.1 A Note on Default Values

This is a completely optional feature. If you find it confusing you can skip it.

Some page types, such as **prominence** have default values for some arguments. An argument might have a default value if alternative values are uncommon or to support new functionality without requiring changes to pre-existing code.

Arguments that have default values do not have to be specified in a sequence file if you are ok with the defaults–this can make your sequence file cleaner and easier to read and maintain.

There are two ways that you can specify an optional value. One is to list it as normal e.g.

```
prominence water water 1 3 p_ac_instr true
```

A second way is to refer to it explicitly by its name as described in sections **??** and **??**. e.g.

```
prominence water 1 3 instructions=p_ac_instr presentAudio=true
```

This second way is useful, for example, if you wanted to specify the value for a later variable such as “presentAudio” but not “instructions”. The only way to do this is by naming the variable

```
Prominence water 1 3 presentAudio=true
```

If one attempted the same list of arguments but without referring to the variable name, this would set “instructions=true” which would cause an error unless instructions named “true” existed in the dictionary file.

```
Prominence water 1 3 true
```

3.4.2 Valid Keyboard Keys

Some pages allow interaction with the keyboard (such pages will direct you to this section). For those pages, experimenters can specify any letter or number on their keyboard, like so

```
somePage playAudioKey=p
```

If we suppose that ‘playAudioKey’ specifies the button that plays an audio recording, pressing the ‘p’ button on the keyboard would play audio.

What about something like the ‘enter’ key? The following codes can be entered to have their associated button registered as the key: **alt, backspace, capslock, ctrl, enter, escape, shift, space, and tab**. And they can be used like

```
somePage playAudioKey=enter
```

3.4.3 Sequence file example

Unlike the below example, in a sequence file each page description should fit on one line. In this example, for the purposes of space, some pages span several lines. A proper example sequence file is also distributed with the lmeds source:
/LMEDS/tests/lmeds_demo/sequence.txt

```
*LMEDS_Demo
login

text_page demo_instructions
consent demo_consent
survey presurvey

media_list audio 1 1 1 [water apples water]
media_choice same_different_instr audio 0.5 1 -1
        [[water apples]] [same different]
boundary_and_prominence water water 0 -1
        boundary_instr prominence_instr true
boboundary_and_prominence apples apples 2 2
        boundary_instr prominence_instr true
end
```

3.5 Sequence file specification - randomizing stimulus order

A short but important section. The sequence file presents the order that all participants will encounter the stimuli. For many experiments however, it is necessary that the stimuli be presented in a different order for each experiment participant–to account for affects of stimulus order for example.

Technical details Here is a brief explanation of how randomization is performed by LMEDS. LMEDS presents materials based on the order presented in a sequence file. In order to present each user with the stimuli presented in a random order, each user must get their own sequence file. When instructed to do randomization, LMEDS reads in the sequence file created by the experimenter, randomizes the order of the stimuli, and then creates a new sequence file. It then reloads that file and begins the experiment.

The following instructions show how sequence randomization can be implemented in a sequence file.

1. Indiciate to LMEDS that you want to randomize a sequence:

   In the .cgi file, in the function 'runExperiment' include the line **individualSequences=True**

2. Specify which portions should be randomized

   In the sequence file, surround the section to be randomized with the following two lines
   <randomize>
   </randomize>

3. Create a folder where the user-specific sequence files will be placed by LMEDS. /LMEDS/tests/<test root folder name>/individual_sequences/<first line in sequence file>/

So, for a short concrete example, here is how the above three would look in practice

```
experiment_runner.runExperiment('lmeds_demo',
                                'sequence_randomized.txt',
                                'english.txt',
                                individualSequences=True)
```

and

```
*LMEDS_Simple_Randomized_Demo
login
text_page same_different_randomized_instructions
<randomize>
media_choice choice_instr audio 0.0 -1 -1 [[water]] [a b]
media_choice choice_instr audio 0.0 -1 -1 [[books]] [a b]
media_choice choice_instr audio 0.0 -1 -1 [[apples]] [a b]
</randomize>
```

and
/LMEDS/tests/lmeds_demo/individual_sequences/LMEDS_Simple_Randomized_Demo

In the above example, although the sequence file specifies the stimuli in a particular order, these three items would be presented in a randomized order for each participant. Each participant would get their own customized sequence file that would be placed in /LMEDS/tests/lmeds_demo/individual_sequences/LMEDS_Simple_Randomized_Demo.

For a fuller example, please see in the example demo files provided with the lmeds distribution:
/LMEDS/cgi/lmeds_randomized_demo.py
/LMEDS/tests/lmeds_demo/sequence_randomized.txt

For more details on post-processing the results of a randomized sequence, please see the post-processing section, Section **??**.

3.6 Sequence file specification - common pages

The following page types may be useful, regardless of the kind of experiment you're running.

3.6.1 login

This is the first page that users see. They enter their name here. Names must be unique. If someone has already attempted to start a test under that user name the user will not be able to proceed from the login page.

"login" takes no arguments.

3.6.2 media_test

Some users have reported an inability to hear audio in LMEDS–perhaps there is an issue with the browser they are using and it won't load audio or their internet is too slow. For this reason there is the media_test page. This page can be presented right after a participant logs in to save them time in the event that their browser is not correctly playing audio or video.

"media_test" takes the following arguments

- `audio` if the page presents an audio file. `video` if it presents a video file.
- the name of the audio or video file

e.g.

```
audio_test audio example_audio_file
```

I recommend using a file that is similar to the stimuli they will experience in the experiment. If the stimuli are long, a short test file might not represent the kind of load the user's computer might encounter while doing the experiment.

3.6.3 consent

Presents the consent form. If users opt not to consent, the test ends immediately. If they consent, they proceed to the next page.

"consent" takes one argument

- the name of the consent form to display to users (the name is a key contained in the language dictionary – so consent forms can be of the appropriate language)

e.g.

```
consent main_consent
```

3.6.4 text_page

A page that displays nothing but the text from a single text key. This page could be used to provide task instructions to users, indicate that they should take a break, etc.

"text_page" takes one argument

- the name of the dictionary key

e.g.

```
text_page first_block_instructions
```

3.6.5 survey

Specifies a survey page (see Section **??** for info on surveys).

"survey" takes a single argument

- the name of the survey file that should be loaded (without the .txt extension)

e.g.

```
survey presurvey
```

This would load the survey stored in the file called "presurvey.txt"

3.6.6 end

The final page of the test.

"end" takes no arguments.

3.7 Sequence file specification - experiment-specific pages

Here is a list of page types involving stimuli presentation.

3.7.1 prominence

Users are presented with an audio file and the associated transcript. They can click on a word to indicate that it is prominent. This changes the selected word to **red**. They can click it again to change it back to black.

"prominence" takes the following arguments:

- name - the name of the audio file
- transcriptName - the name of the text file
- minPlays - the minimum number of times the audio file has to be played before the user can continue. A value of -1 indicates no minimum.
- maxPlays - the maximum number of times the audio file can be played before the audio button is disabled. A value of -1 indicates no maximum.
- instructions - on the page users will encounter short (approx. one line) instructions reminding them of the task. With this argument, you can present different stimuli with different short instructions (e.g. meaning, acoustics, vague).

- presentAudio - either “true” or “false”, specifies whether the audio is hidden or presented. If “false” the values for “minPlays” and “maxPlays” are ignored. **Default**: *true*
- bindPlayKeyID - the keyboard button that will activate the audio ’play’ button on the page. **Default**: *None* (i.e. no keyboard key) See section **??** for acceptable values.
- bindSubmitID - the keyboard button that will activate the ’submit’ button on the page. **Default**: *None* (i.e. no keyboard key) See section **??** for acceptable values.
- minNumSelected - the minimum number of words that the user must select before continuing. **Default**: *-1* (i.e. no minimum)
- maxNumSelected - the maximum number of words that the user can select in order to continue. **Default**: *-1* (i.e. no maximum)

e.g.

```
prominence water water 1 3 prominence_acoustics true
prominence apples apples 1 1 prominence_meaning
prominence apples apples -1 -1 choose_two true minNumSelected=2
prominence apples apples 1 1 prominence_meaning bindPlayKeyID=p
```

3.7.2 syllable_marking

“syllable_marking” is the same as **prominence** except that the user can mark individual syllables rather than the word

“syllable_marking” takes the same arguments as **prominence** pages but with the addition of one final obligatory argument:

- syllableDemarcator - specify the symbol for dividing syllables in the text transcript (e.g. ‘.’)

e.g.

```
syllable_marking syllables syllables 1 -1
        nonspecific_syllables true syllableDemarcator=.
```

3.7.3 boundary

Users are presented with an audio file and the associated transcript. They can click on a word to mark the presence of a boundary after it. This places a solid, vertical line after the word. Clicking on the word again makes the vertical line disappear.

"boundary" takes the same arguments as **prominence** pages but with the addition of one final, optional argument:

- boundaryToken - specify the symbol to use for marking boundaries between words. The default symbol used by LMDS is a vertical bar "|").

e.g.

```
boundary water water 1 3 boundary_acoustics true &
boundary apples apples 1 2 boundary_meaning
```

3.7.4 boundary_and_prominence

A combination of prominences and boundaries. Users first mark boundaries. They then can mark prominences. While marking prominences they can see but not change the boundaries that they marked–this is the only difference between this page and splitting the prominence and boundary task across two pages.

The arguments for boundary_and_prominence are the same as for boundary, except that users need to indicate both the instructions for boundaries and the instructions for prominences.
e.g.

```
boundary_and_prominence water water 0 3 b_instr p_instr true &
```

3.7.5 media_list

The user is presented with a single button. On being pressed, a series of audio or video files will be played. On a follow up page, the user could answer questions about the stimuli that were seen or heard.

media_list takes the following arguments:

- `audio` if the page will present audio files. `video` if the page will present video files.
- the length of pause in seconds between each file
- the minimum number of times the media series can be played
- the maximum number of times the media series can be played
- the list of media files to play, enclosed by `[ ]`

e.g.

```
media_list audio 1 1 1 [water apples water]
```

3.7.6 media_slider

The user is presented with a single button, a text transcript, and a slider. On the press of the button the media file is played. The user can then input their response through the slider. The text description and the transcript can be used to help guide their decision.

media_slider takes the following arguments:

- instructionText - the dictionary key to the instructions to be presented.
- audioOrVideo - `audio` if the page will present audio files. `video` if the page will present video files.
- minPlays - the mininum number of times the media can be played.
- maxPlays - the maximum number of times the media can be played.
- mediaName - the name of the media file to play.
- transcriptName - the name of the text file to display below the audio.
- sliderMin - the smallest numerical value on the slider (this is not shown to the user).
- sliderMax - the largest numerical value on the slider (this is not shown to the user).
- sliderLabel - the label to put under the slider. **Default**: *None* (i.e. no label).

- leftRangeLabel - the label to put on the left edge of the slider. **Default**: *None* (i.e. no label).
- rightRangeLabel - the label to put on the right edge of the slider. **Default**: *None* (i.e. no label).

e.g.

```
media_slider prominence_scale_instr audio 1 -1 water water_word
      0 100 leftRangeLabel=nonprominent rightRangeLabel=prominent
```

3.7.7 media_choice

This is used for presenting forced-choice-like tasks to the user with the number of media files (stimuli) and responses set by the experimenter.

This page replaces the various functionality of the pages: same_different_stream, axb, ab, and their many variants. These pages have been removed from LMEDS.

media_choice takes the following arguments:

- the short form instructions to present on the page
- `audio` if the page will present audio files. `video` if the page will present video files.
- duration of pause in seconds for media files
- the minimum number of times the audio series can be played
- the maximum number of times the audio series can be played
- a list of lists of audio files (see discussion below)
- a list of the labels for the response options
- mediaButtonLabelList - a list of names for labels for the audio files (must match the length of the media buttons) **Default**: *None* (No labels are placed above the audio files–see discussion below)
- transcriptList - a list of transcripts for the buttons (must match the number of media buttons). Transcripts are placed below the the buttons. **Default**: *None*

- bindPlayKeyIDList - a list of keyboard buttons that will activate the associated media buttons on the page (must match the length of the media buttons). **Default**: *None* (i.e. no keyboard keys) See section **??** for acceptable values.
- bindResponseKeyIDList - a list of keyboard buttons that will activate the associated media buttons on the page (must match the length of the media buttons). **Default**: *None* (i.e. no keyboard key) See section **??** for acceptable values.
- timeout - the time in seconds after which the page will automatically transition to the next page and the user's inaction will be marked in the output. **Default**: *None* (i.e. no timeout)

e.g.

```
media_choice same_different_instr audio 0.5 1 -1 [[water apples]]
             [same different] [left_choice right_choice]
```

compare that example with the next one:

```
media_choice same_different_instr audio 0.5 1 -1 [[water] [apples]]
             [same different] [left_choice right_choice]
```

In the first example, with [[water apples]], there is one button that plays two audio files with a half second delay between each. In the second example, with [[water] [apples]], there are two audio buttons and each is played only once. Note that [water apples] will produce an error. **The argument must be a list of lists**: [[water apples]] or [[water] [apples]]. Similarly with three arguments: [[water apples candy]] or [[water] [apples] [candy]] for the cases with either 1 button or 3 buttons, respectively. In a minimal case, with only one audio file, you would write: [[audio_name]], as in

```
media_choice same_different_instr audio 0.5 1 -1 [audio_name]
       [p b] [audio_button_text] [p_button_text b_button_text]
```

Here the user would hear a single audio file and select option "p" or option "b" (such as in classical experiments investigating perception of VOT contrasts).

If you feel the response options are obvious–because there are two audio buttons and two corresponding response options–you could remove the last option like so:

```
media_choice same_different_instr audio 0.5 1 -1
              [[water] [apples]] [same different]
```

Here are some examples of using keyboard keys to activate the buttons on the page. The important thing here is to use the same number of keys and buttons. As with other lists used in sequence file, the value still needs to be enclosed in [ ] even if there is only one value

This example specifies one audio button with three response buttons. Pressing ‘p’ will play the two audio files ‘water’ and ‘apples’ in sequence. Pressing ‘z’ will register the *dislike* response, the spacebar *indifferent*, and ‘m’ *like*.

```
media_choice three_point_scale_instr audio 0.5 1 −1 [water apples]
             [dislike indifferent like] bindPlayKeyIDList=[p]
             bindResponseKeyIDList=[z space m]
```

This example specifies two audio buttons with two response buttons. Pressing the ‘w’ key will play the left audio button for the audio file ‘water’ while pressing the ‘p’ key will play the right audio button for the audio file ‘apples’. Pressing ‘z’ will register the *same* response while pressing ‘m’ will register the *different* response.

```
media_choice same_different_instr audio 0.5 1 −1 [[water] [apples]]
             [same different] bindPlayKeyIDList=[w p]
             bindResponseKeyIDList=[z m]
```

Normally, the result is output as a series of 0s separated by commas with a 1 indicating the position of the marked item. The number of digits reflects the number of response buttons. All of the examples above use either two or three. **If the timeout feature is being used** an extra digit is added. The value is 0 if the user selected any output but the value is 1 if the timeout was triggered before the user selected a response. Here is an example entry in the sequence file for a media_choice with timeout

```
media_choice same_different_instr audio 0.5 1 −1
             [[water] [apples]] [same different] timeout=10
```

It is also possible to have **media-less media choice pages**. One might want to have a media task or a memory recall task where the audio was played on a previous page. To do this, one simple needs to make an empty audio list of lists like so: [[ ]] Here is an example used in the demo for a fill-in-the-blank type presentation which may have been part of a memory task.

```
media_choice fill_in_the_blank_instruct audio 0.5 -1 -1 [[ ]]
        [polluted cold not_drinkable] transcriptList=[water_blank]
```

3.8 Parsing the LMEDS output

The output to LMEDS is quite simple. Each line has 4 components:

- the page type
- the argument list
- peripheral information collected on the page (number of times users listened to audio files, time spent on the page, etc.)
- the users input in data-entry fields presented on the page

For most needs, the page type and user response are the most important bits of information. The page type is separated from the rest of the row by the occurrence of the first comma. The user input is separated from the rest of the row by the sequence ";,". The argument list is enclosed in `[ ]` and separated from the peripheral information by a comma.

The user response is a string of comma-separated values. Every individual item in a data-entry field, except for textboxes, is represented by a 0 or 1, where 0 indicates the user did not choose that item and 1 indicates the user did choose that item. So if the user has a choice between A or B and selects A, their output will look like 1, 0. And if they select B: 0, 1. For a boundary_and_prominence page, each word gets two digits in the output (one for boundary and one for prominence).

Let's look at a real example. Here we have the sequence:

```
login
boundary_and_prominence water water 0 3 b_acoustics
        p_acoustics true
media_choice same_different_instr audio 0.5 1 -1
        [[water apples]] [same different]
media_choice axb_instr audio 0.5 1 -1
        [[water] [water2] [water3]] [ax xb]
media_choice ab_instr audio 0.5 1 -1 [[water]] [a b]
end
```

The first page is a typical boundary and prominence page. The three subsequent pages are all media choice pages. The first would be used for a same different task, the second for an axb task (the user listens to three audio files, is one of the files more similar to one of the other two? And in the third one, a labeling task–is the file a member of category ‘a’ or category ‘b’? And here is one possible output for this sequence.

```
boundary_and_prominence ,[ water water 0 3 b_acoustics
        p_acoustics true ], 1 ,1 ,0:8.9 ,1; ,
        0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0
media_choice ,[ same_different_instr audio 0.5 1 −1 [[ water apples ]]
        [same different ]] ,0 ,0 ,0:4.8 ,2; ,1 ,0
media_choice ,[ axb_instr audio 0.5 1 −1 [[ water ] [ water2 ] [ water3 ]]
        [ax xb ]] ,0 ,0 ,0:6.7 ,3; ,0 ,1
media_choice ,[ ab_instr audio 0.5 1 −1 [[ water ]] [a b ]] ,
        0 ,0 ,0:3.3 ,4; ,1 ,0
```

If we separate out just the page name (the content before the first ‘,’) and the user response (everything after the ‘;,’) we can easily see the data that we want to analyze:

```
boundary_and_prominence ; ,0 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,0 ,0 ,0 ,0 ,0
media_choice ; ,1 ,0
media_choice ; ,0 ,1
media_choice ; ,1 ,0
```

LMEDS offers some useful post-processing that can help prep data for analysis (see **??**. Otherwise, with the above information, you should have everything you need to do your own analysis.

3.9 CGI File (experiment-wide configuration)

The cgi file specifies the location of the experiment files, the sequence name, and the dictionary name. Using these pieces of information, LMEDS will correctly locate the inputs and place the outputs. The CGI file itself is quite short. You can copy and paste the text below, changing only the arguments to the function **runExperiment**.

Through the CGI file, one can indicate experiment-wide parameters.

runExperiment() has the following arguments

- name of the folder in /lmeds/tests
- name of the sequence file
- name of the dictionary file
- disableRefresh: if 'True' and users try to refresh a page or click back, they will be unable to progress. Set to 'True' if you suspect your subjects might try to revisit completed pages. Otherwise, in my experience, let them use back and refresh–they're only going to use it if their browser freezes up.
- audioExtList: the list of audio extentions to use in an experiment. The default value is ['.ogg', '.mp3']
- videoExtList: the list of video extensions to use in an experiment. The default value is ['.ogg', '.mp4']
- allowUtilityScripts: if 'True', users can run utility scripts from the browser. Could be 'True' for a local LMEDS install used for testing but should be 'False' for the online version. See section **??** for more information on the utility user scripts.
- allowUsersToRelogin: if 'True', users can use their login name to resume their progress if the are stopped midway through an experiment (loss of internet, audio fails to load, etc). Otherwise, if 'False', users will have to restart from the beginning.

```
\#!/usr/bin/env python
\# -*- coding: utf-8 -*-

import experiment_runner as exp_runner
exp_runner.runExperiment('lmeds_demo',
                         'sequence.txt',
                         'english.txt',
                         disableRefresh=False,
                         audioExtList=['.ogg', '.wav'],
                         videoExtList=['.webm', '.mp4'],
                         allowUtilityScripts=True,
                         allowUsersToRelogin=True)
```

4 Running your LMEDS experiments

Before you run your experiment, it is recommended that you run the user script sequence_check.py to ensure verify that your sequence file, dictionary file, and survey files are all well formed and that your resource files are all available.

Regardless of whether you will run your experiments remotely or locally, before you run your experiment, you must create the output directory. Suppose your experiment folder is called **my_experiment**. Then you should have the directory **/tests/my_experiment**. The first line of your sequence file is the name of the specific experiment, such as **experiment_b**. You will then need to create the folder **/tests/my_experiment/output/experiment_b**. This is where LMEDS will store the user responses.

If you are using the randomization function, you will also need to create the directory **/tests/my_experiment/individual_sequences/experiment_b**

4.1 Running experiments remotely

To run LMEDS experiments online, you're going to need a server. This server could be one that you own or maintain, one managed by the IT department at your place of work, or one that you rent online.

Whichever route you go, the server you use will need to have cgi enabled and also be use python (apache's mod_python). If your server has these two things, drop the unzipped contents of the LMEDS distribution on the server

Not all commercial servers have python or cgi enabled. One web service that I have used for running LMEDS experiments is `https://www.nearlyfreespeech.net/`. They are a pay-as-you-go website, unlike many web hosts online. Their pricing model is a little confusing but to run just under 1000 subjects over the last 2 years has cost me less than $50. Other similar services can be found online if you search around.

If you are installing a server from scratch, you'll need to make sure it comes with hooks to python–which is fortunately fairly typical these days. In your httpd.conf file you'll want to enable .cgi files. That is all! LMEDS only uses standard python libraries and was developed under python 2.7 (please check the readme.rst file distributed with LMEDS for information on what versions of python LMEDS has been tested with–several versions in the 2.x and 3.x series have been tested and worked fine).

Before you run experiments you must change the public permissions of the cgi file to execute (665 for the permission octals). The output directory (/tests/your-test-name/output/first-line-in-sequence-file/) then needs to have full write and execute permissions (777 for the permission octals). If you are using the randomization feature then the folder /tests/your-test-name/individual_sequences/first-line-in-sequence-file/) also needs to have full write and execute permissions.

4.2 Running experiments locally

LMEDS comes with it's own local server. Unlike a regular server, experiments can only be run on the computer that is running the local server. However, the local server is useful when:

- no remote server exists
- running experiments in lab-like setting
- the internet is very slow or there is no internet
- self-piloting an experiment or portion of an experiment

The only requirement for the local server is python (https://www.python.org/downloads/). The most recent version of 2.7 and 3.x should work with LMEDS. Please check README.rst for more information on what versions of python LMEDS works with.

4.2.1 Running the server

There are two ways to run the server. One is from the command line:

```
python lmeds_local_server.py
```

Alternatively, python comes with a tool for writing and running code called IDLE. You can open the script lmeds_local_server.py in IDLE via `file >> open` and select `run >> run module`.

Once the local server is running, it will direct you to enter in a URL in your web browser. At this point, the server is working and you'll be able to access your experiments from that machine only.

4.2.2 Running your experiments

Once the local server is running, there are a few final things you have to do before you can run your experiments.

On Windows you must rename your .cgi files to .py. If extensions are hidden, you might need to make them visible in order to change them easily.

On Linux and OS X you must change the public permissions of the cgi file to execute (665 for the permission octals). The output directory then needs to have full write and execute permissions (777 for the permission octals).

5 User scripts

Everything in LMEDS can be done by hand, but the provided scripts located in `/lmeds/user_scripts` enable you to easily perform certain routine tasks.

All scripts can be run in one of three ways.

The first, and most user-friendly option, is from the web browser. Begin by entering the URL of the relevant experiment. Add a '?' and then the script name. For example,

```
http://127.0.0.1:8123/cgi-bin/lmeds_demo.py?sequence_check
```

Each script requires certain information. Conveniently, all of this information is already specified in the .cgi/.py file, so if you choose this option, you don't need to enter any further information–unlike the other two options. The output is displayed directly in the browser–except for get_duration and post_process_results which may display confidential information (e.g. if the subjects write their full name as their user id). Instead, for those, a log file is generated that the experimenter can access later.

In order to run scripts via the web browser, the cgi file must be configured properly. See section **??**.

Second, the user scripts can be run from the command line (*cmd* on Windows or *terminal* on OS X or Linux). `python <<script_name>>.py -h` will print out the options for that script. For example,

```
python sequence_check.py -h
```

Third, user scripts can be run from within a python development environment, such as IDLE, which is bundled with every version of python. If you open IDLE, choose `File >> Open` and open the desired script. The script will open in a new window. Select `Run >> Run Module`. The application will launch. It will ask if you want to go into interactive mode. Type "yes". The script will then ask question-by-question for all of the information it needs to run.

5.1 Generating the language dictionary

Given a sequence file, python can generate an empty dictionary that contains all of the keys needed by the pages used in the sequence file. The script can also update existing dictionaries used in other experiments or in cases where pages have been added or removed from a sequence file.

```
python generate_language_dictionary.py -m update lmeds_demo
                                    sequence.txt english.txt
```

There are three versions of this script that can be run from the web browser.

To create a dictionary for a new project:

```
http://127.0.0.1:8123/cgi-bin/lmeds_demo.py?create_dictionary
```

To update an existing dictionary (preserves existing keys and text):

```
http://127.0.0.1:8123/cgi-bin/lmeds_demo.py?update_dictionary
```

To remove keys that are no longer relevant to a sequence file. Note that if there is a language dictionary that is common to two different sequence files, running crop in this manner on one sequence file may make the dictionary no longer work with the other sequence.

```
http://127.0.0.1:8123/cgi-bin/lmeds_demo.py?crop_dictionary
```

5.2 Verifying experiment integrity

This user script ensures that an experiment is ready to be run. It makes sure that all text keys are in the dictionary, and that LMEDS can access all the wav and text resources that are included in the sequence file. **Even if you use this script, you'll still want to run through the experiment at least once before you start collecting data.** However, this script will save you some headaches if you've misspelled resource names in the sequence file, for example.

```
python sequence_check.py lmeds_demo sequence.txt english.txt true
```

To run this from a web browser:

```
http://127.0.0.1:8123/cgi-bin/lmeds_demo.py?sequence_check
```

5.3 Getting test duration

This simple script will output the length of time each user spent on your experiment, along with the average time and the standard deviation.

```
python get_test_duration.py lmeds_demo sequence.txt
```

To run this from a web browser:

```
http://127.0.0.1:8123/cgi-bin/lmeds_demo.py?get_test_duration
```

5.4 Post-processing the results

Each user will have their data stored in a separate file. For doing many kinds of analysis, this is not convenient. Furthermore, the questions are removed from the responses and the responses are mixed with different kinds of questions.

This script was made to remedy some of these issues. First, items are separated by page type. Then, for survey items and rpt experiments, items are transposed, paired with the inputs (survey questions and transcripts, respectively) and then all of the data is combined into one spreadsheet for each page type. This makes it much more convenient to explore the data and to do statistical analysis.

Currently, LMEDS may experience duplicate data entries–in particular when the **lmeds_main.py**'s *disableRefreshFlag* is set to False and the user hits back or refresh. The code will warn when it detects multiple items and prevent the user from continuing but there is also the option to remove duplicate entries when they appear (printing the found instances). This happens before anything else in the script.

The code to remove duplicate entries is a little naive. If there is a situation where two identical lines could appear adjacent in your sequence file, LMEDS will assume in the output that the two lines are duplicates of each other and automatically remove the second one if *removeDuplicatesFlag* is set to True. If set to False, it makes no assumptions about the relationships between lines. True is the convenient option, False is the safe option–if you are concerned.

The script can also remove any items used in the experiment that the user does not want to include in the output–for example test items or items that are removed from the study before data collection concludes. By default the post processing user script doesn't remove any data.

An important note on randomized sequences, if a sequence is flagged as 'randomized' in it's cgi file, the order each participant did the stimuli in will be randomized. The post-processing code will recognize this automatically arrange all user responses in the order specified in the main sequence file. However, the post processing script will also output the order that each item was presented in. This can be seen in the example output files included in the following folder with LMEDS
/LMEDS/tests/lmeds_demo/output/LMEDS_Randomized_Demo/duplicates_removed_results/

Output data can be found in subfolders created in the output folder of the experiment.

```
python post_process_results.py testName sequenceFN
        removeDuplicatesFlag removeItemList=[]
```

```
python post_process_results.py lmeds_demo sequence.txt false
```

To run this from a web browser (assumes *removeDuplicatesFlag* is True):

```
http://127.0.0.1:8123/cgi-bin/lmeds_demo.py?post_process_results
```

6 LMEDS change history

November 2, 2016 - LMEDS 2.4

- Stimulus presentation can now be automatically randomized for each user
  - Test post-processing scripts will unscramble the output while marking the order users did the task in.
- Task timeout and media-less support added to media_choice pages
- A limit can be set on the minimum and maximum number of prominences and boundaries that can be made on a text in the boundary- and prominence-marking pages.
- Keyboard keys can be bound to buttons on a page (e.g. the ‘p’ button can be used to play audio or ‘enter’ can be pressed to submit a response)
- Numerous bugfixes and code cleanup.

May 17, 2016 - LMEDS 2.3

- Video now supported with the same controls and functionality as audio
- User script utilities can now be run from a web browser for greater ease of use
- Pages can now have sliding scales
- New syllable-marking page (branch of prominence-marking pages)
- Experiment participants can now relogin and resume from where they left off (if enabled by the experimenter)
- Integration tests added for greater reliability. Also, numerous bugfixes.

December 15, 2015 - LMEDS 2.2

- Added support for python 3.x (tested on 3.5)

September 2015 - LMEDS 2.1

- Substituted the many specific forced choice pages (same_different, ab, axb, etc) with one new page: audio_choice

August 2015 - LMEDS 2.0

- First public release.
- Inclusion of user script utilities.
- Numerous bugfixes and stability improvements (audio is significantly less error prone).
- Offline server for running experiments locally.
- Companion website.

May 2014 - LMEDS 1.5

- Object oriented refactor.
- Numerous bugfixes and stability improvements.

November 2013 - LMEDS 1.0

- First official release.
- First official experiments run using LMEDS.

May 2013 - LMEDS 0.0

- Work begins on LMEDS

7 LMEDS manual change history

November 2, 2016

Updated the manual to be in line with changes from LMEDS 2.4.

May 17, 2016

Updated the manual to be in line with changes from LMEDS 2.2 and 2.3. Added the workflow schematic. Clarified other parts of text.

November 11, 2015

Updated the manual to be in line with changes from LMEDS 2.1.

August 5, 2015

Second release. Move from Word to LaTeX for easier tracking of changes.

December 17, 2013

First release