

07 Fakultät für Informations-, Medien- und
Elektrotechnik, Studiengang Technische Informatik
Institut für Nachrichtentechnik

Research Project

Arnau Vázquez Giner

Scale, a Matlab Open Source Software Tool for Listening
Experiments

Contents

1	Introduction	1
1.1	What is Scale?	1
1.2	Tests implemented	1
1.2.1	Adaptive	1
1.2.2	Double-blind Triple-stimulus with Hidden Reference (ABC-HR)	2
1.2.3	ABX	3
1.2.4	MUSHRA	3
1.2.5	SAQI	5
1.3	Type of stimuli available	5
1.3.1	.wav stimuli (WAV)	5
1.3.2	.wav combined with the SSR (BINAURAL_SSR)	5
2	Structure of the program	7
2.1	General	7
2.2	Start	9
2.3	Config	11
2.4	Testing	15
2.5	Analysing	18
3	Procedure Implementation	20
3.1	Configuration process file changes	20
3.1.1	TestTypeEnum	20
3.1.2	Common_create_test	21
3.1.3	GULConfig_3_TestName	23
3.2	Testing process file changes	25
3.2.1	Common_create_results	25
3.2.2	GULTester_TestName	26
3.3	Analyzing process file changes	33
3.3.1	analysing_generate_struct_TestName	33
3.3.2	GULAnalyse_TestName	34
	Bibliography	37
.1	Appendix A	38

List of Figures

1.1	3-AFC procedure trial window	2
1.2	Double-blind triple-stimulus with hidden reference procedure trial window [3]	3
1.3	<i>Scale</i> 's interface during an ABX test	4
1.4	<i>Scale</i> 's interface during a MUSHRA test	4
1.5	<i>Scale</i> 's interface during a SAQI test	5
1.6	System architecture	6
2.1	Structure of the program	8
2.2	Structure of the start part	9
2.3	GULStart_Intro window	9
2.4	GULStart_main window	10
2.5	Subject_register window	10
2.6	Structure of the config part	11
2.7	GULConfig_1 window	12
2.8	GULConfig_2 window	12
2.9	GULConfig_3_TestName window	13
2.10	GULConfig_4 window	13
2.11	GULConfig_5_StimuliName window	14
2.12	GULConfig_6 window	14
2.13	Structure of the testing part	15
2.14	GULTester_prepare_StimuliName window	16
2.15	GULTester_login window	16
2.16	GULTester_Scenario_selection	17
2.17	GULTester_TestName	17
2.18	Structure of the analysis part	18
2.19	GULAnalyse_TestName	19
3.1	Code snippet of the enumeration file	21
3.2	Code snippet of the common test creation code	22
3.3	Code snippet of the specific test creation code	22
3.4	Code snippet of the stimuli specific test creation code	23
3.5	New files created	24
3.6	Special configuration window of MUSHRAMOD test	24
3.7	Code snippet of the common part of the results creation file	25
3.8	Code snippet of the test specific part of the results creation file	26
3.9	Newly created test files	27
3.10	Code snippet of the default functions in the Tester window	27

3.11	GUIDE edition mode of the default test window	28
3.12	GUIDE edition mode of the test window with sliders	29
3.13	Code added to the GULTester_MUSHRAMOD_OpeningFcn function .	29
3.14	Code added to the GULTester_MUSHRAMOD_OutputFcn function . .	30
3.15	Code corresponding to the slider functions	31
3.16	Code corresponding to the stimuli button functions	31
3.17	Code corresponding to the next button function	32
3.18	Code corresponding to the subject information part of the results struct creation	33
3.19	Code corresponding to the ratings information part of the results struct creation	34
3.20	Code corresponding to the completion of the results struct creation . .	34
3.21	Newly created analysis files	35
3.22	Analyse window in edition mode	36
3.23	Code corresponding to the analysis window modification	36

*

1 Introduction

1.1 What is Scale?

Scale is a software tool that covers the full chain of setup, conduction and analysis of psychoacoustic experiments. It offers several testing procedures, the interaction between researcher or subjects and the software is done via a graphical user interface (GUI) and does not require any programming skills. Test setups or results can be easily ported from one instance of the program to another. Thus everything is portable and exchangeable between different computers and researchers. The first version of *Scale* was presented at the DAGA Conference in 2013 in Merano, Italy, and the second version was presented at the DAGA Conference in 2015 in Nürnberg.

1.2 Tests implemented

The initial version of *Scale* included a selection of frequently used test procedures like simple or transformed staircase adaptive procedures and double blind triple-stimulus with hidden reference (ABC-HR). In the second version three additional procedures, ABX, MUSHRA and SAQI, are implemented. In the present version, the third version, no more tests have been included; however, the whole code has been modified in order to give researchers the possibility of adding new test procedures themselves.

1.2.1 Adaptive

Adaptive procedures aim to find a threshold of detection in the psychometric function of a determined dimension of a sound. Stimuli are presented and varied in one dimension. The amount of variation is increased or reduced depending on the preceding subject's responses and on the respective adaptation method. In *Scale* the adaptation is made using different staircase methods like simple staircase (1up/1down) on the one hand, and some transformed staircase methods (1up/2down, 2up/1down) as described in [1] on the other hand. The threshold estimation can either vary depending on the ending conditions of the trial (limited number of runs or reversals) or on how the average is calculated.

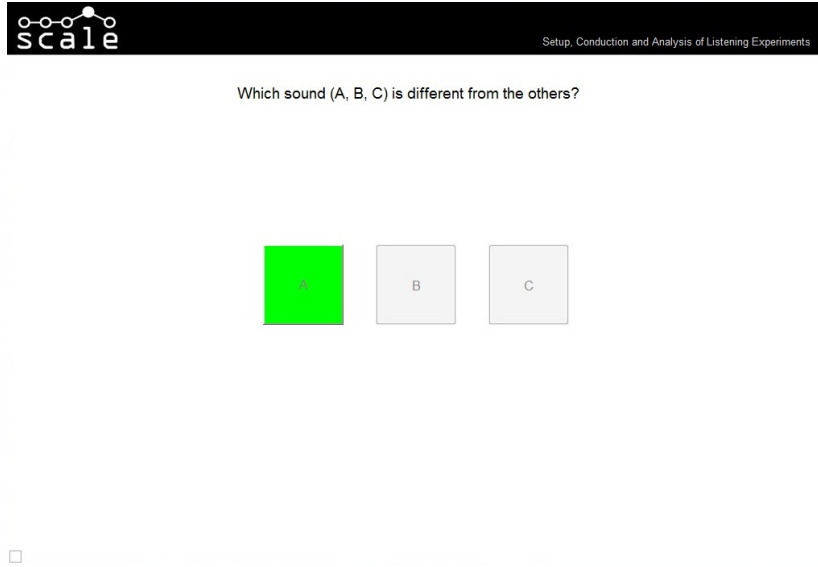


Figure 1.1: 3-AFC procedure trial window

The tests have to be combined with a paradigm. *Scale* provides different paradigms which can be divided into two groups: n-AFC (n alternative forced choice) paradigms and Yes/No paradigms. In the n-AFC paradigms n intervals (as used in [2]) are presented to the subject. When n is equal to 2 the subject has to decide in which of the two intervals a designated signal is present. When n is greater than 2 the subject has to decide in which of the n intervals the presented stimulus is different. The sample assignment to the intervals of the n-AFC paradigms is always automatically randomised. When using a Yes/No paradigm only one interval that includes one or more sounds is presented to the subject. The subject has to decide whether the signal occurs within the presented interval or not.

1.2.2 Double-blind Triple-stimulus with Hidden Reference (ABC-HR)

This procedure has become a standard in psychoacoustics and is used to assess small impairments between sound samples. In every trial, three stimuli are presented in three intervals ("A", "B" and "C"). The stimulus in "A" is presented as the known reference, the stimuli in "B" or "C" are randomly assigned, whereas one of them is a hidden reference and the other one is a sample which is varied in one determined dimension. After listening to the stimuli the subject is asked to assess the impairments between "A" and "B" and "A" and "C" using a rating scale. The rating is performed with a slider along a continuous scale with anchors. The number of grades in the scale as well as the text in the labels of every mark can be set. Thus all requirements to perform the test *"Subjective Assessment of Small Impairments in Audio Systems"*

Figure 1.2: Double-blind triple-stimulus with hidden reference procedure trial window [3]

Including Multichannel Sound Systems” described in the recommendations of the ITU [3] are met, see Figure 1.2.

1.2.3 ABX

The ABX test is a simplification of the *ABC-HR* test. In a trial of an *ABX* test, a subject is presented with two stimuli which are A and B, followed by a third one called X. After hearing A, B and X, the subject must select which of the stimuli in the intervals A or B is the same as in interval X. In this case, as opposite to the *ABC-HR*, no rating is done. In Scale, each scenario can contain many trials and for each trial, the position of the reference stimuli in A and B is randomized. Figure 1.3 shows a trial window of the test.

1.2.4 MUSHRA

The aim of a multi-stimulus test with hidden reference and anchor (MUSHRA) test [4] is to rate global differences between several audio stimuli. All stimuli are presented in a single trial and have to be compared to a given reference. Each stimulus has a continuous scale (continuous quality scale) which goes from 0 (bad) to 100 (excellent), as shown in Figure 1.4. Reference and stimuli can be switched over instantly. The order of the stimuli is randomized and every trial has to include a hidden reference and an anchor.

1.2 Tests implemented

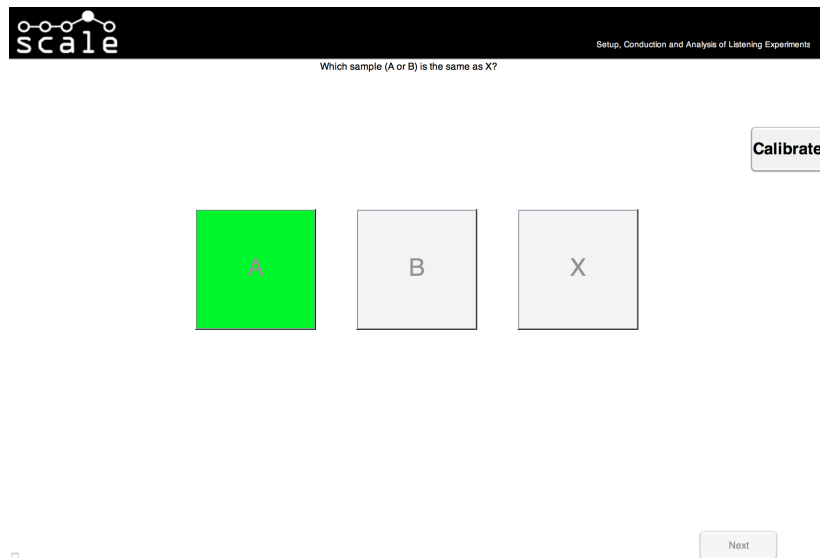


Figure 1.3: *Scale*'s interface during an ABX test

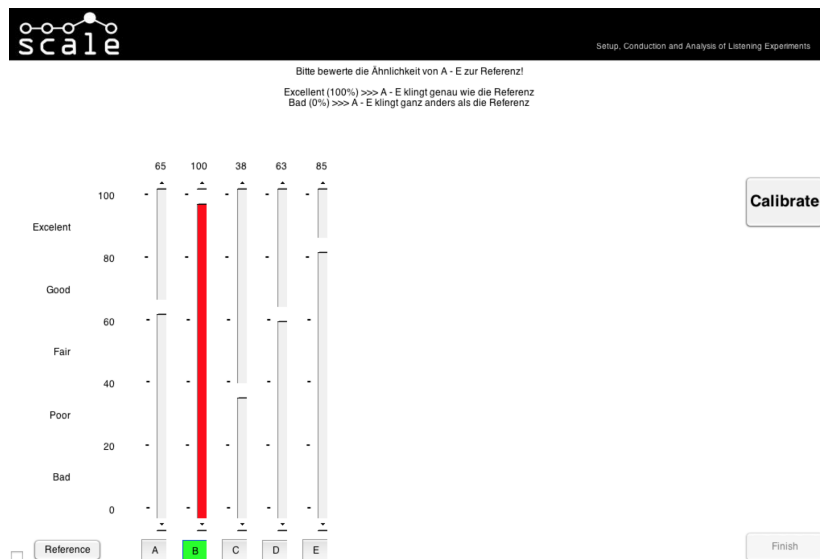


Figure 1.4: *Scale*'s interface during a MUSHRA test

1.2.5 SAQI

The spatial audio quality inventory (SAQI) test [5] has been specifically designed for the perceptual evaluation of virtual acoustic environments. In every trial a reference and a stimulus are presented together with 48 verbal descriptors of perceptual qualities that are assumed to be of practical relevance when comparing virtual auditory environments. Each descriptor comprises a rating scale with a pair of opposed adjectives in its scale ends. The subject's task is to compare the stimulus to a given or inner reference and give a rating for each perceptual quality.

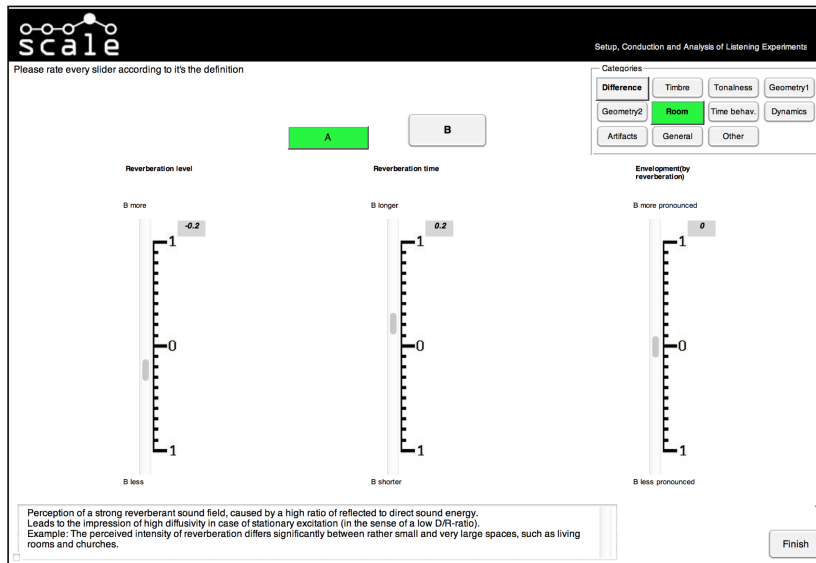


Figure 1.5: *Scale*'s interface during a SAQI test

1.3 Type of stimuli available

In this version of Scale two different types of stimuli are available. Those are *.wav* stimuli and *.wav combined with SSR*. Other types of stimuli can be added by modification of the existing code files.

1.3.1 .wav stimuli (WAV)

The *.wav* stimuli are just audio files in the mentioned format which will be played by *Matlab* using the function `play` of the `audioplayer` objects.

1.3.2 .wav combined with the SSR (BINAURAL_SSR)

This type of stimuli are obtained by a combination of the rendering software Sound Scape Renderer (SSR) [6] and the *.wav* playback function of *Matlab*. As shown in

Figure 1.6, *Scale* processes subject's inputs and operates the *SSR* using its network interface via TCP/IP protocol while the test is performed. The *SSR* runs in the background generating stimuli with the combination of the incoming audio signal, the tracker data and the corresponding head related impulse response (HRIR) or binaural room impulse response (BRIR) set.

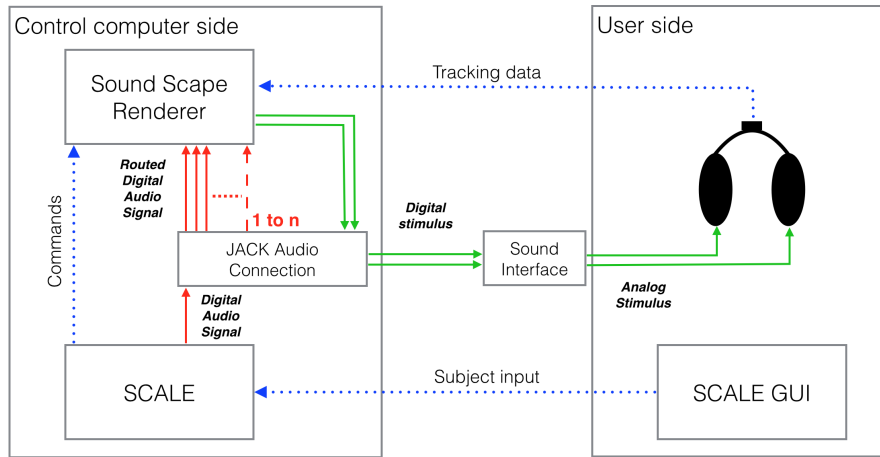


Figure 1.6: System architecture

2 Structure of the program

This version of *Scale* has been implemented in order to give the user the possibility to implement new test procedures. The implementation of a new procedure, however, requires some skills in the Matlab programming language and its graphical user interface design environment (GUIDE).

Since *Matlab* is not an object oriented language, writing a software which can be expanded while maintaining a clean code is a difficult task. In object oriented languages, the programmer tries to keep the code as short and efficient as possible and this can be done by using classes. In the case of *Scale*, another approach has been taken. Instead of keeping the code as short and efficient as possible, the code has been expanded with the idea that a user can make his own changes by adding new code files and modifying as little as possible the existing source code.

In this section an explanation of *Scale*'s structure and the files involved in each part of the program (configuration, testing and analyzing) is given.

2.1 General

Scale can be divided in three main modules which are, from a coding point of view, independent from each other; those are the configuration module, the testing module and the analyzing module. The functions needed by each module are placed inside three different folders in the program folder. A fourth module, which is the start module is also included, this module is only in charge of starting the program and the user do not need to modify it.

Figure 2.1 shows the general structure of the program. Each square is named after its file name and they correspond to interface windows or to functions (the ones in cursive). The squares marked in yellow indicate that the files need to be created (if a new type of test is being implemented) or green (if a new type of stimuli is being added). The squares in blue indicate that the files need to be modified (some code added to them).

Following a detailed explanation of the windows and functions related to each part of the program is presented.

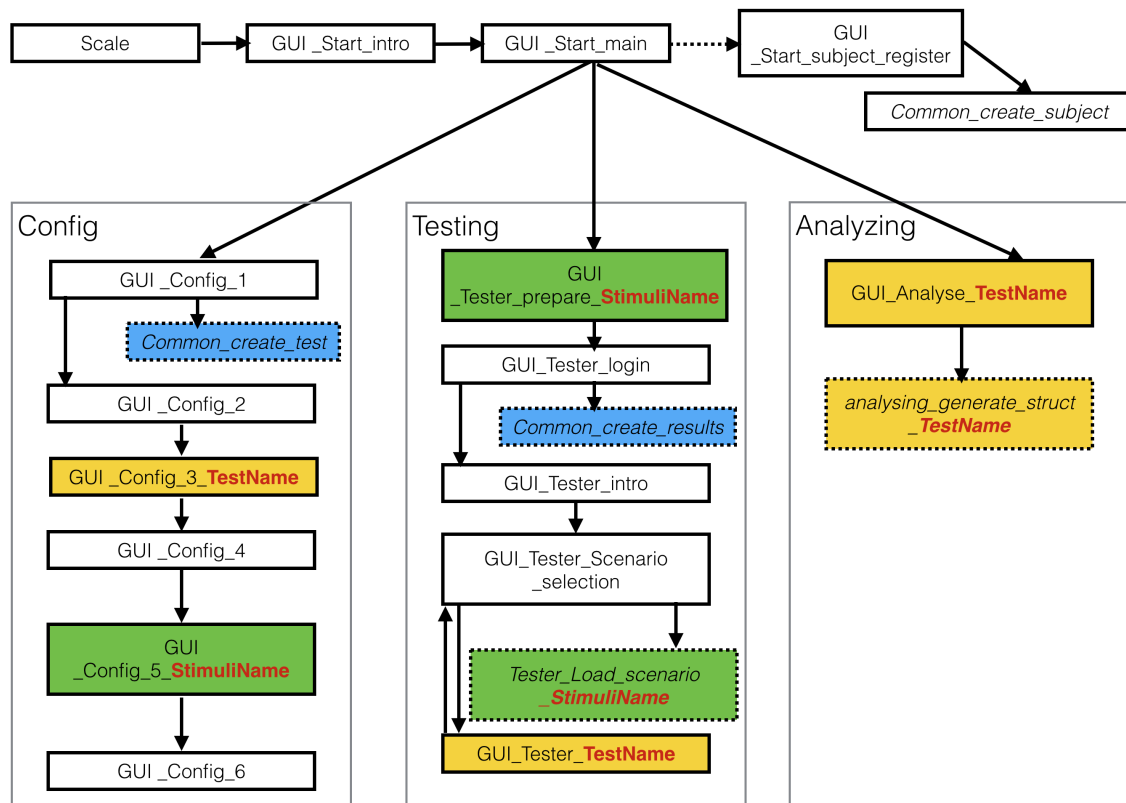


Figure 2.1: Structure of the program

2.2 Start

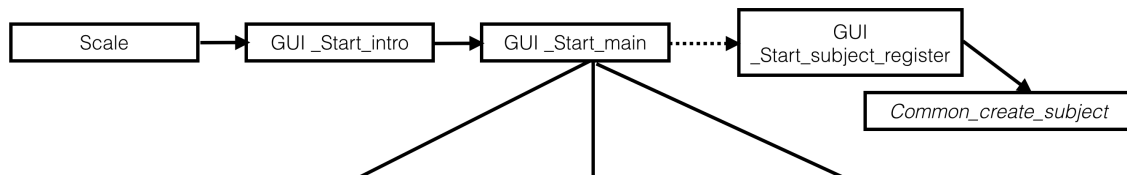


Figure 2.2: Structure of the start part

As depicted in Figure 2.2 the start part contains three windows and a function. After the program is launched, the **GULStart_Intro** window (Figure 2.3) is shown.

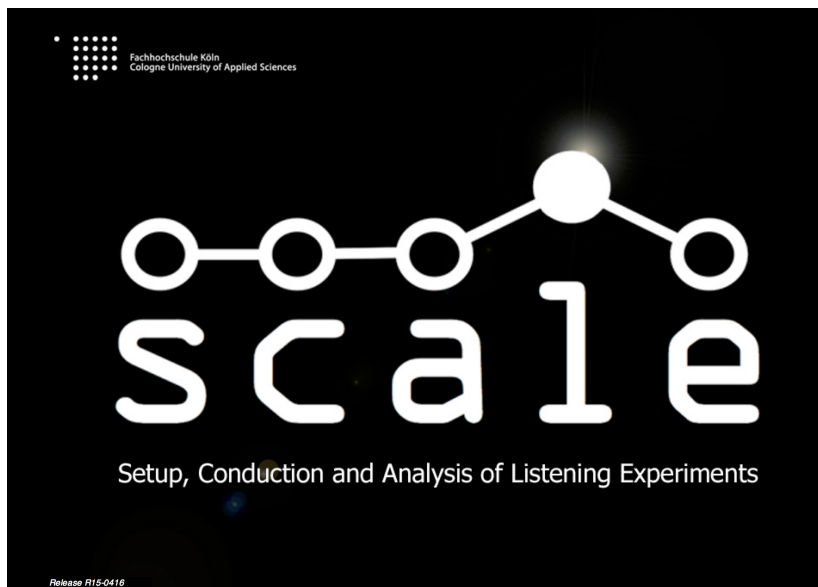


Figure 2.3: GULStart_Intro window

This window just presents the animation and calls the next window **GULStart_main** (Figure 2.4) which presents the three options (configuration, testing and analyzing) plus the registration of the subject button.

In order to register a subject, the user must click the button and the window **GULStart_subject_reg** (Figure 2.5) is shown. This window calls the function **Common_create_subject** which create the subject information file. From the files presented in this section, only the file **GULStart_main** needs to be slightly modified; this will be explained in the test implementation (Section 3).

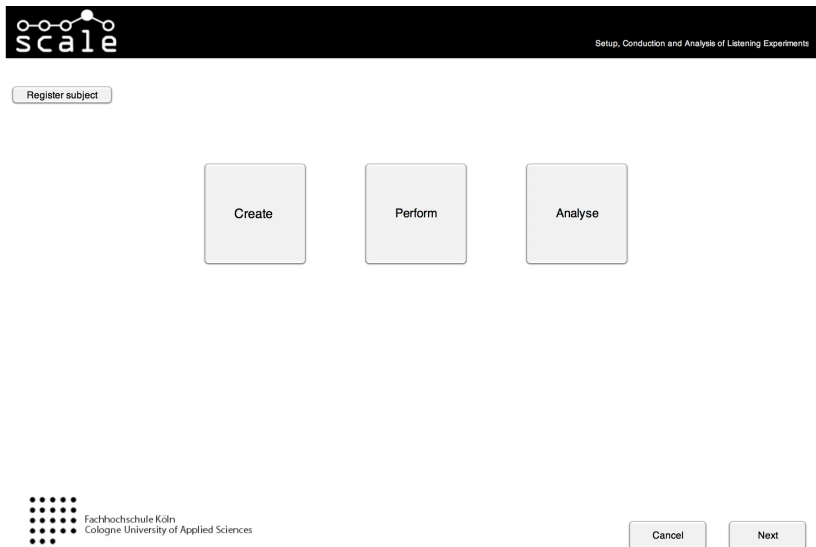


Figure 2.4: GUL_Start_main window

The screenshot shows the Subject_register window. It features a black header bar with the 'scale' logo on the left and the text 'Setup, Conduction and Analysis of Listening Experiments' on the right. Below the header, there are input fields for 'Name (Angela)', 'Surname (Merkel)', 'Birthday' (with dropdowns for day, month, and year), 'Gender' (with a dropdown for Male), and 'Country of birth' (with a dropdown for Germany). Below these fields, there is a list of questions with corresponding dropdown menus: 1) Experience in Psychoacoustic tests (yes), 2) Knowledges about audio processing? (no), 3) Do yo play any musical instrument (never), 4) Weekly exposure to loud music (club, concert...) (0 h...), 5) Weekly use headphones (hours) (0), Are they in ear headphones (yes), 7) Do you have known hearing problems (no). Below these questions, there are three text input fields for 'Other aspects', 'If answer to 7 was yes, make a short description', and 'If answer to 2 was yes, make a short description'. At the bottom right, there are 'Cancel' and 'Next' buttons.

Figure 2.5: Subject_register window

2.3 Config

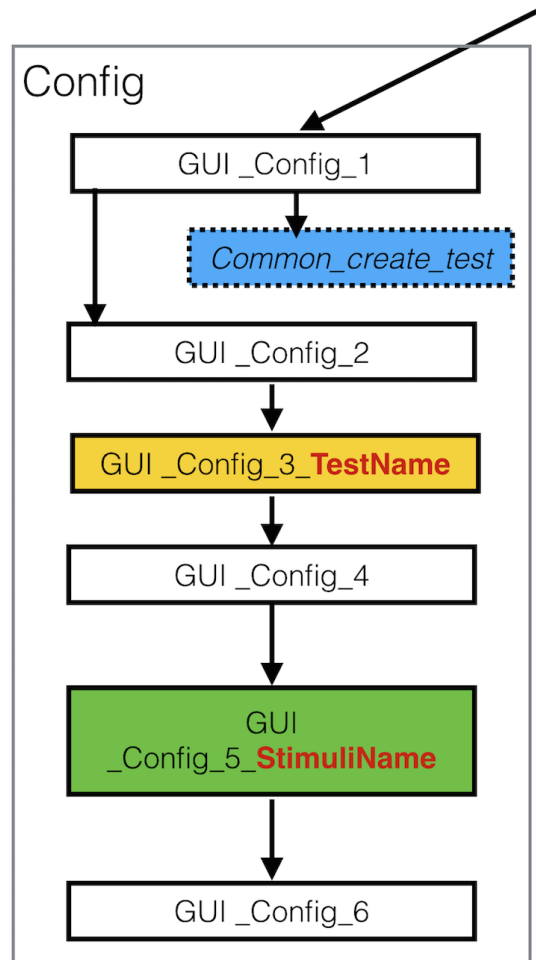
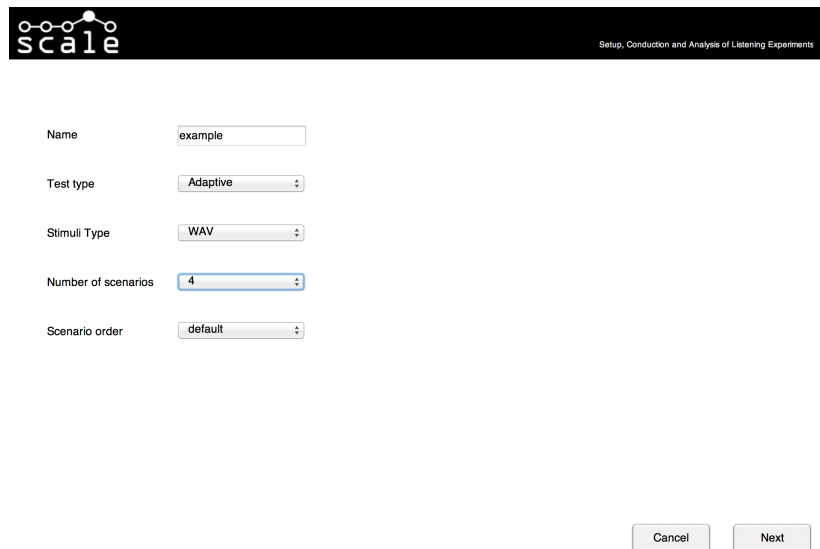


Figure 2.6: Structure of the config part

The config part is related to the configuration of the test. As shown in Figure 2.6 six windows and one function are involved in the configuration process. The first window **GUI_Config_1** (Figure 2.7) just requests the user to input the name of the procedure, the type of procedure, the scenarios and the stimuli type. After confirming the desired options, the *Common_create_test* function which creates the test information file is called. Please note that the *Common_create_test* function will have to be modified in case of implementing a new type of test.

On a next step, the window **GUI_Config_2** (Figure 2.8) appears. In this window, the instructions appearing at test start and in each scenario are given; no modification has to be done here.

The third window to appear is the **GUI_Config_3_TestName** window where the special configuration options for each type of test are set. The file containing the code



scale Setup, Conduction and Analysis of Listening Experiments

Name example

Test type Adaptive

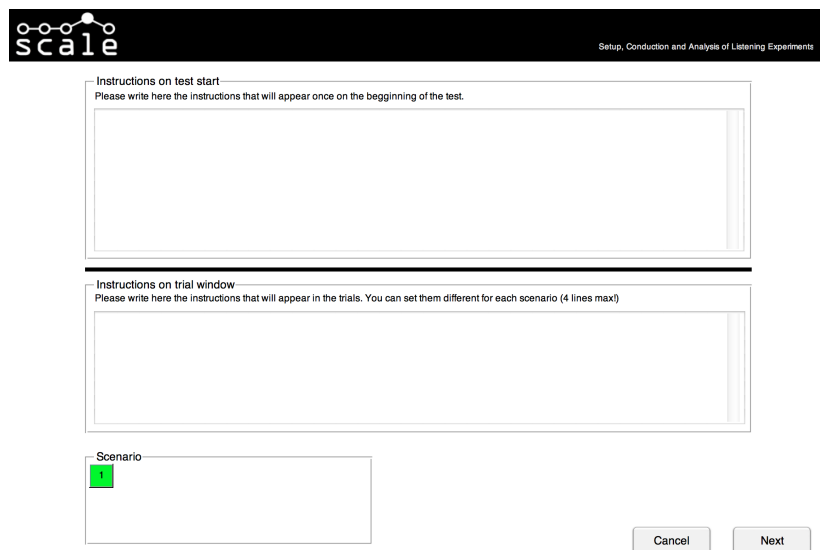
Stimuli Type WAV

Number of scenarios 4

Scenario order default

Cancel Next

Figure 2.7: GUL_Config_1 window



scale Setup, Conduction and Analysis of Listening Experiments

Instructions on test start
Please write here the instructions that will appear once on the beginning of the test.

Instructions on trial window
Please write here the instructions that will appear in the trials. You can set them different for each scenario (4 lines max!)

Scenario
1

Cancel Next

Figure 2.8: GUL_Config_2 window

for this window will have to be newly implemented in case a new test type is being added (this is explained in Section 3).

Figure 2.9: GUI_Config_3_TestName window

GUI_Config_4 (Figure 2.10) window acts as a bridge between the test configuration options and the Stimuli addition window.

Figure 2.10: GUI_Config_4 window

The stimuli addition window **GUI_Config_5_StimuliName** (Figure 2.11) will have to be newly implemented in case a new type of Stimuli is added. For the moment it is implemented for using *.wav* stimuli and *.wav + SSR binaural renderer stimuli*.

The last window of the test configuration part, window **GUI_Config_6** (Figure 2.12) which just asks the user to click the confirmation button to finish the test setting.

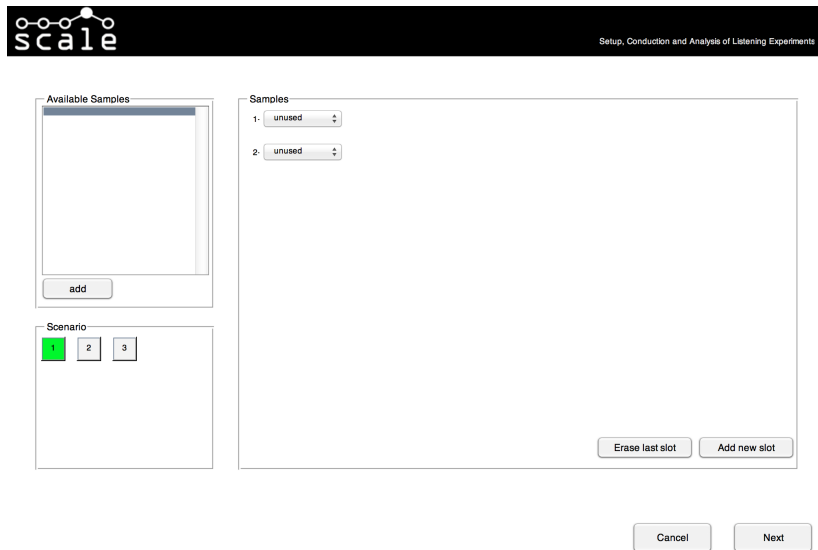


Figure 2.11: GUL_Config_5_StimuliName window

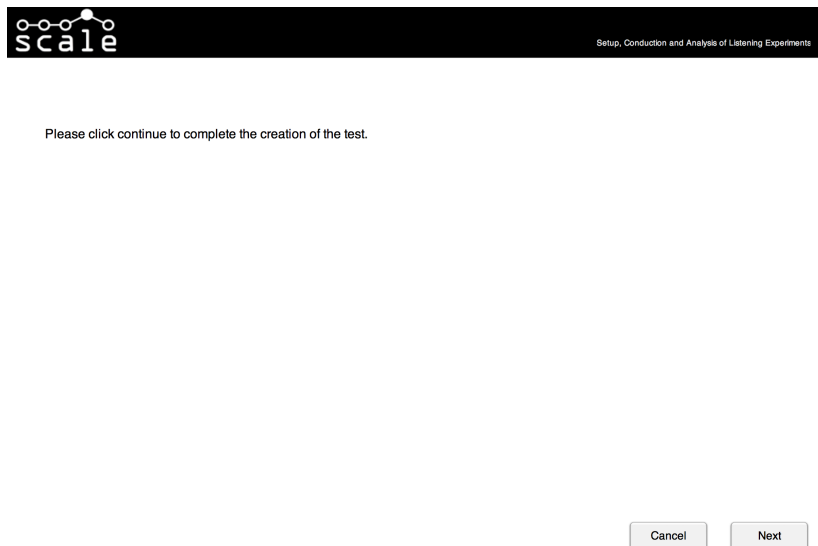


Figure 2.12: GUL_Config_6 window

2.4 Testing

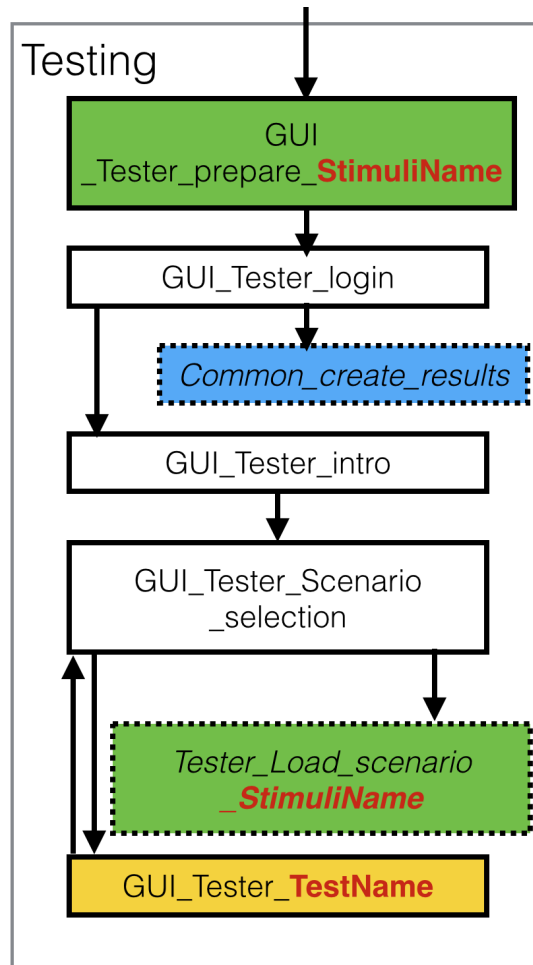


Figure 2.13: Structure of the testing part

The testing part is in charge of the actions happening during test performance. The first window to appear is the **GUI_Tester_prepare_StimuliName** (Figure 2.14). Basically this window acts as a bridge after the launch of *Scale* and the start of a test in case something needs to be loaded or started, for example the *SSR*.

If nothing needs to be done, the user will just press continue (this file needs to be implemented in case a new type of stimuli is added). The next window to appear is the login window called **GUI_Tester_login** (Figure 2.15) where the subject logs in.

After the login is done, the scenario selection window, called **GUI_Tester_Scenario_selection** (Figure 2.16) opens. This window is where the subject sees the next scenario that he is going to perform.

After clicking the next button, a script with the name **Tester_Load_Scenario_StimuliName** is called. This script needs to be implemented in case a new type of stimuli is used and

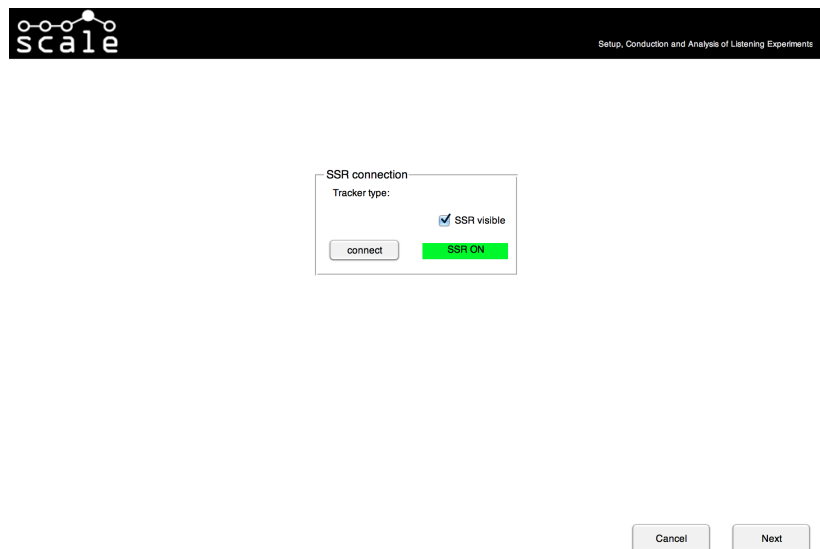


Figure 2.14: GULTester_prepare_StimuliName window

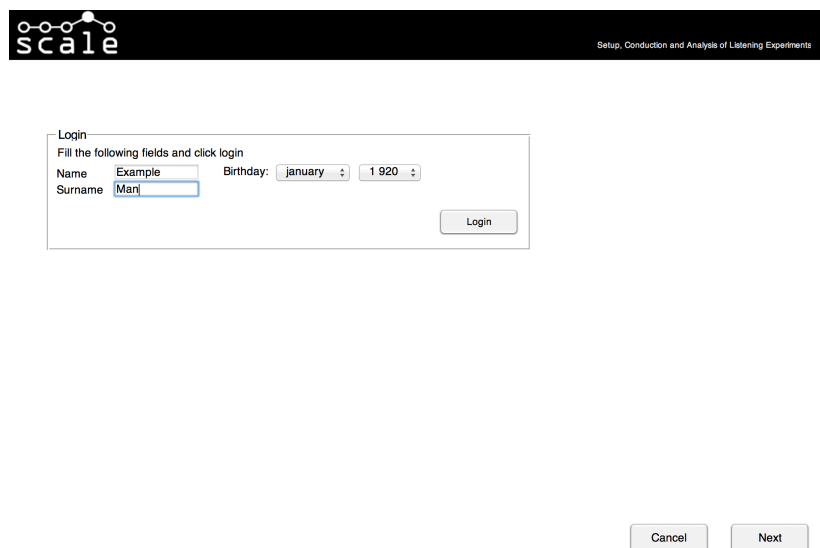


Figure 2.15: GULTester_login window

2.4 Testing

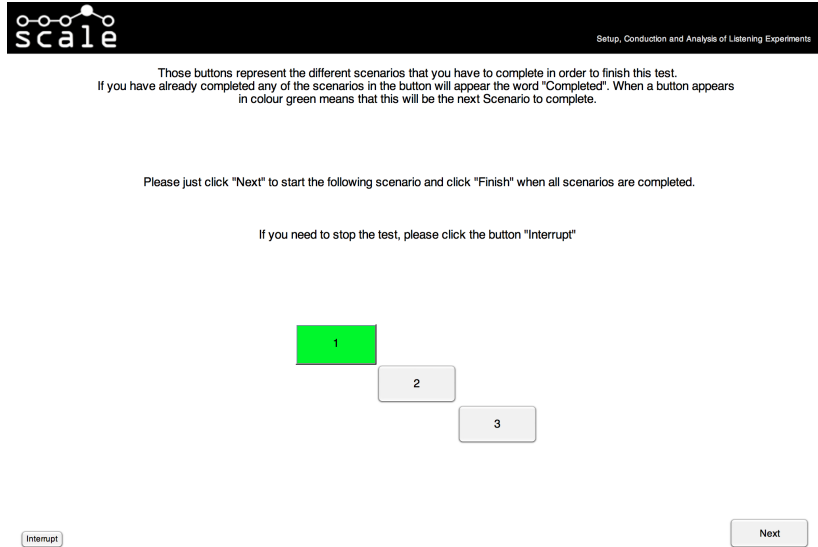


Figure 2.16: GUI_Tester_Scenario_selection

its purpose is to prepare the scenario in case something needs to be done, for example load a scenario of the *SSR*.

Finally, after the selection of the scenario and the preparation of the resources needed, the test window, with the name **GUI_Tester_TestName** (Figure 2.17) opens. This is where the action takes place, here the subject will interact during the test. This window needs to be implemented for each type of test and this is explained in Section 3.

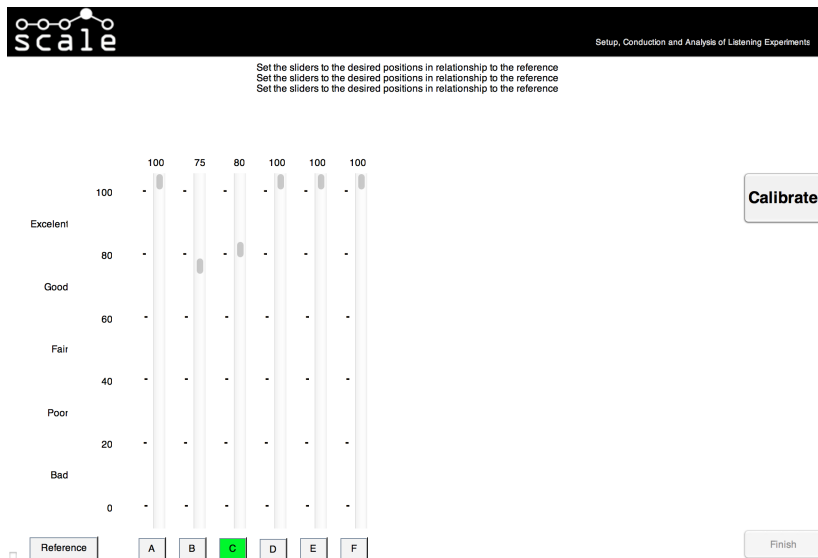


Figure 2.17: GUI_Tester_TestName

2.5 Analysing

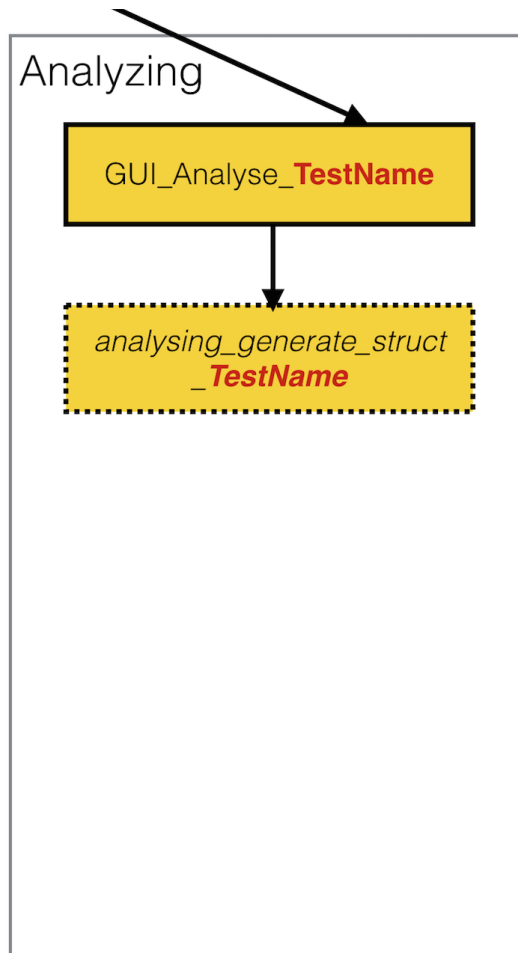


Figure 2.18: Structure of the analysis part

The analysing part is in charge of the analysis of the test results. Results can be visualized graphically on the same *Scale* interface or exported to an external *.mat* file in order to process them as the researcher desires. Only one window and one script are responsible for this process; however, the window and the script must be newly implemented for each kind of test. The reason for that is that each test has different type of results. On the one hand, in a test like *MUSHRA*, several values are obtained in a single trial or scenario and those values are "independent" from each other. On the other hand in a test such as a 2AFC, a value for each trial is obtained, since the responses of the subject are aimed to obtain a single average value, in this case a threshold.

Figure 2.19 is an example of the window *GUI_Analyse_TestName* which varies depending on the type of test. The script *analysing_generate_struct_TestName*

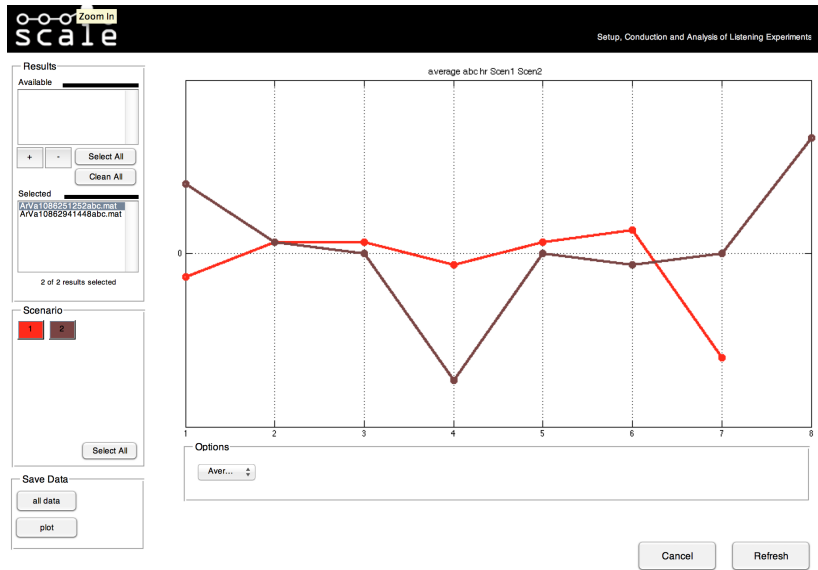


Figure 2.19: GULAnalyse_TestName

is responsible of exporting the test data. This script, has to be newly implemented for each new type of test and this is explained in Section 3

3 Procedure Implementation

This chapter explains the implementation of a new type of test procedure. All the script, window or function files which have to be modified are explained step by step together with code examples. The test type to be implemented will be called MUSHRAMOD. The procedure is similar to a MUSHRA. In each trial some samples will be presented, each of them together with a slider. The user can listen to a sample by clicking on the button linked to it or by moving the slider above. The sliders will be positioned vertically and a scale will be attached to it. The subject will then give their ratings positioning the slider tip in a certain position. The difference between each stimuli will be the set of binaural impulse responses used to filter a noise *.wav* sample which will be played as a loop during all the trial.

3.1 Configuration process file changes

In this part, two files have to be modified and one file need to be created. Those files are:

1. `TestTypeEnum`
2. `Common_create_test`
3. `GUL_Config_3_TestName`

3.1.1 TestTypeEnum

The file *TestTypeEnum* defines an enumeration for the different type of existing tests. This is used during many parts of the program execution since having the test procedures defined in an enumeration reduces the amount of files to be modified when implementing a new procedure. The file is found in *Scale/Scale_code_files/classes/TestTypeEnum*. In this file only the name of the procedure has to be included in the list of the existing tests. Figure 3.1 shows the code snippet where the test procedure name has been placed (highlighted in red).

```

classdef TestTypeEnum
    enumeration
        Adaptive
        ABC_HR
        ABX
        MUSHRA
        SAQI
        MUSHRAMOD
    end
end

```

Figure 3.1: Code snippet of the enumeration file

3.1.2 Common_create_test

The file *Common_create_test* creates the test file. This file contains the information related to the test itself. Some information is common for all type of procedures, for example the number of scenarios, the creation date, the type of procedure, the type of samples used, etc. Other information is specific from the procedure or the sample type. The file is found in *Scale/Scale_code_files/functions/Common_create_test*. The test information is saved in a struct called *test_info*, which is saved inside *Scale/Scale_user_files/tests/nameOfYourTest/test_info*. The fields of the test information struct are the following:

- *test_info.testName*: name of the test.
- *test_info.testType*: type of procedure, value of the enumeration *testType* (MUSHRA, ABX...).
- *test_info.stimuliType*: type of stimuli, value of the enumeration *stimulitype* (WAV, BINAURAL_SSR...).
- *test_info.totalScenarios*: number of scenarios.
- *test_info.scenarioOrder*: order of the scenarios (default or random).
- *test_info.instructStartWindow*: instructions of the test.
- *test_info.instructTrialWindow*: instructions of the trial.
- *test_info.creationDate*: creation date.
- *test_info.creationCompleted*: boolean, indicates if the creation of the test is completed.
- *test_info.TestName*: this field is a struct which contains the specific information of the test procedure. This is explained after this list.

- `test_info.stimuli`: this struct contains info related to the stimuli and its composition depends on the stimuli type used.
- `test_info.SSR_Config`: this struct contains information related to the use of the *SSR* like the connection id, the tracker type, etc.

The common part is created as shown in Figure 3.2, this part do not need to be modified and it is automatically filled.

```
function test_info = Common_create_test(testName,testType,stimuliType,totalScenarios,scenario)
    %%those are the fields of the test_info struct, common for all type of
    %%tests
    test_info.testName=testName;
    test_info.testType=testType;
    test_info.stimuliType=stimuliType;
    test_info.totalScenarios=totalScenarios;
    test_info.scenarioOrder=scenarioOrder;
    test_info.instructStartWindow='';
    test_info.instructTrialWindow=cell(1,totalScenarios);
    test_info.creationDate=date;
    test_info.creationCompleted=0;

    %%those are the fields of the test_info struct, special for each type of
    %%test, this info is saved in a substruct like
    %%test_info.<type_of_test>.... (example: test_info.adaptive.paradigm (paradigm parameter fo

    switch test_info.testType
    case TestTypeEnum.Adaptive %%case adaptive test
        test_info.adaptive.paradigm=ParadigmEnum.twoAFC; %%default
    end
```

Figure 3.2: Code snippet of the common test creation code

The test specific part, has to be implemented by the user, as a new **case** in an existing switch clause. In the case of our new test, no modifications have to be done to it, since the only information needed by the test, are the number of stimuli in each trial, and this is given in the `test_info.stimuli` struct. Figure 3.3 shows the code snippet of this part where an empty **case** condition is placed with the name of the test procedure. In case we would like to add something in the struct, we would write `test_info. <name of our test procedure>.(field)`. (See the SAQI **case** clause as an example in the snippet).

```
case TestTypeEnum.ABX %%case ABX test
case TestTypeEnum.MUSHRA %%case MUSHRA

case TestTypeEnum.SAQI %%case SAQUI
    test_info.SAQI.language=LanguageEnum.english; %%by default english
    test_info.SAQI.selectedQualities=zeros(1,48);
    test_info.SAQI.referenceType='external'; %%by default external
    test_info.SAQI.sliderSteps=3; %%by default 3
case TestTypeEnum.MUSHRAMOD %%case MUSHRAMOD
end
```

Figure 3.3: Code snippet of the specific test creation code

The third part of the code file includes the struct inside `test_info` related to the stimuli. This struct will be later filled with the number of stimuli for each scenario, the names of the *.wav* samples and the filter numbers in case of using the *SSR*. For the creation of a new procedure, this part do not need to be modified. The code snippet presented in Figure 3.4 presents this last part of the code.

```

%%those are fields of a substruct inside the test_info struct regarding
%%the type of stimuli used test_info.<name_of_the_stimuli_type>....
switch test_info.stimuliType
case StimuliTypeEnum.WAV
    test_info.stimuli(totalScenarios).wavList=cell(1,1);
    test_info.stimuli(totalScenarios).nStimuli=zeros(1,1);
case StimuliTypeEnum.Binaural_SSR
    test_info.stimuli(totalScenarios).wavList=cell(1,1);
    test_info.stimuli(totalScenarios).BRIRList=zeros(1,1);
    test_info.stimuli(totalScenarios).asdFileName='';
    test_info.stimuli(totalScenarios).nStimuli=zeros(1,1);
    test_info.SSR_config.trackerType='';
    test_info.SSR_config.sampleRate='';
    test_info.SSR_config.renderType='brs';
    test_info.SSR_config.connectionIdx=0;
    test_info.SSR_config.cardId=0;
end

```

Figure 3.4: Code snippet of the stimuli specific test creation code

3.1.3 GUI_Config_3_TestName

The file `GUI_Config_3_TestName` corresponds to the code of a window that appears during the configuration process. This window corresponds to the setting of the special parameters of the test procedure. In the case of our new test, no parameters need to be added so we don't really need it, however the window has to be implemented. The way of doing this is the following:

1. Open Matlab
2. Set the working directory to *your_path_to_scale/Scale/Scale_code_files/config*
3. Type in the command prompt *guide GUI_Config_3_DUMMY* (the call to the file has to be like this, be sure you do not call the file plus *.m* or *.fig*) and press enter. Wait for the figure window to open.
4. Having the window open, click on *file* and then *save* and save the figure as *GUI_Config_3_TestName* (TestName refers to the name of the procedure, in this case it has been saved as *GUI_Config_3_MUSHRAMOD*).
5. Close the figure window.
6. If everything was done right, now two new files should have been created named *GUI_Config_3_TestName.fig* and *GUI_Config_3_TestName.m*. Those files refer to the figure file and the code file respectively. (Figure 3.5)

Now we have already the window where we can configure the options special to the new created procedure. As explained before, no special configuration has to be done in our MUSHRAMOD procedure, so no modification needs to be done in the file. The dummy file, just presents a window which says "Please click next to add the samples" and after the user clicks on next, the sample window will be opened. When the window is opened, the *test_info* struct is loaded and temporally saved inside the handles variable. In case

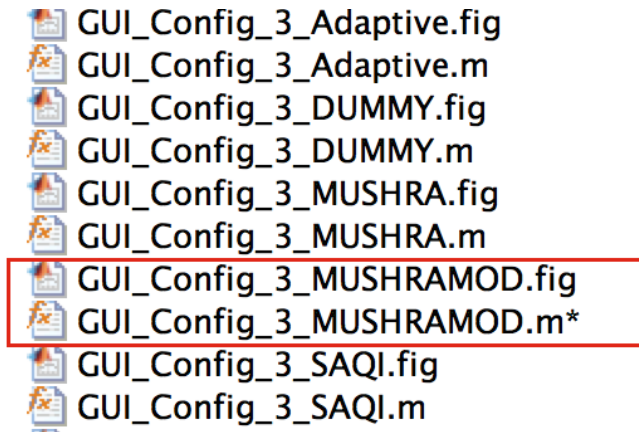


Figure 3.5: New files created

special options for the test are needed, we could add input fields that the user can interact with. The information then would need to be saved during the process as *handles.test_info.TestName.field=xxx*. When the user press next, the *test_info* struct is passed to the next window.

Figure 3.6 shows how the window will look like in the case of the test which is being created.

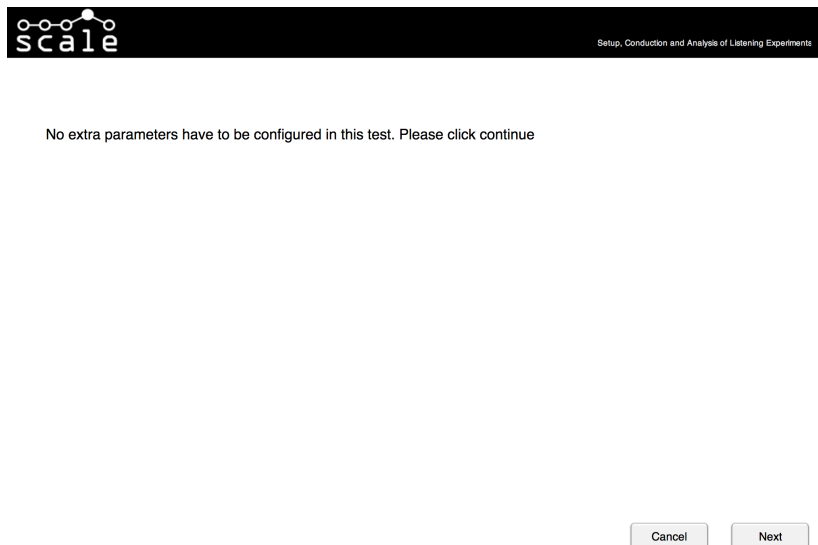


Figure 3.6: Special configuration window of MUSHRAMOD test

*Please note that **handles** is in itself a struct where information can be saved during the execution of an interface window and the information contained in this struct can be accessed by all the functions in the window. It is like a global variable. If we would not load the **test_info** struct inside the **handles** struct, we could not access it and modify it from all functions inside the window.*

3.2 Testing process file changes

For this part one file has to be modified and one file has to be created. The files are the following:

1. Common_create_results
2. GUI_Tester_TestName

3.2.1 Common_create_results

The first file is the *Common_create_results* file found in *your_path_to_scale/Scale_code_files/functions/Common_create_results*. This file implements the function responsible of the creation of the results file for a determined test of a determined subject. The function will be called after the subject performs the login and the results file created will be automatically named as: *first two letters of the subject name/first two letters of the subject surname/birth month of the subject/birth year of the subject/hour of creation/first three letters of the test name/.mat* and will be saved in *your_path_to_scale/Scale_user_folder/name_of_the_test/results*.

The function can be divided in two parts, the first part is common for all tests and is responsible of creating the fields such as the date of the experiment, the vector indicating if scenarios are completed, etc. Figure 3.7 shows the snippet of code where this is happening.

```

% Author: Alvaro Vazquez
function results_info = Common_create_results(test_info,subject_info)
%%those are the fields of the results_info struct, common for all type of
%%tests
time=clock();
month=num2str(time(2));
day=num2str(time(3));
hour=num2str(time(4));
minute=num2str(time(5));

results_info.resultsCode=[subject_info.Code day month hour minute test_info.testName(1:3)];
results_info.testName=test_info.testName;
results_info.subjectCode=subject_info.Code;
results_info.date=date;
results_info.completedScenarios=zeros(1,test_info.totalScenarios);
results_info.completedTest=0;

if(test_info.scenarioOrder==ScenarioOrderEnum.default)
    results_info.scenariosOrder=1:test_info.totalScenarios;
elseif(test_info.scenarioOrder==ScenarioOrderEnum.random)
    results_info.scenariosOrder=randperm(1:test_info.totalScenarios);
end

%%those are the fields of the results struct, special for each type of
%test this info is added in a subsequent file

```

Figure 3.7: Code snippet of the common part of the results creation file

The second part of the function consists of a switch clause where for each test, a new case statement must be added. In our case, we have added a statement called **case TestTypeEnum.MUSHRAMOD**. Inside the case statement, a new struct is created inside the *results_info* struct with the name of the test, in this case MUSHRAMOD.

During the test, *Scale* will save the results inside the struct *MUSHRAMOD*. The struct contains at the same time a vector of structs called *scenario* (there are as many values as scenarios) which will contain the ratings given by the user and the presentation order of the sliders which will be randomized at this point. The ratings will be saved specifically in the *sampleRating* vector (which have a length equal to the number of stimuli in the scenario) while the presentation order will be saved in the *presentationOrder* vector which also have a length equal to the number of stimuli. The function *randperm()* is used here to randomize the values. This might be a little bit confusing to understand by words, so Figure 3.8 shows the snippet of code where all this happens.

```

case TestTypeEnum.MUSHRA %%case MUSHRA
for scenarioIdx=1:test_info.totalScenarios
    results_info.MUSHRA.scenario(scenarioIdx).sampleRating=zeros(1,test_info.stimuli(scenarioIdx).nStimuli);
    results_info.MUSHRA.scenario(scenarioIdx).presentationOrder=randperm(test_info.stimuli(scenarioIdx).nStimuli);
end
case TestTypeEnum.SAQI %%case SAQUI
for scenarioIdx=1:test_info.totalScenarios
    results_info.SAQI.scenario(scenarioIdx).qualitiesRating=zeros(1,48);
end
case TestTypeEnum.MUSHRAMOD %%case MUSHRAMOD
for scenarioIdx=1:test_info.totalScenarios
    results_info.MUSHRA.scenario(scenarioIdx).sampleRating=zeros(1,test_info.stimuli(scenarioIdx).nStimuli);
    results_info.MUSHRA.scenario(scenarioIdx).presentationOrder=randperm(test_info.stimuli(scenarioIdx).nStimuli);
end
end

```

Figure 3.8: Code snippet of the test specific part of the results creation file

3.2.2 GUI_Tester_TestName

This file is responsible for the code of the trial window. Since tests can be very different from each other, a new file has to be implemented for each new test. This part will require medium-high programming skills in *Matlab* and specially *GUIDE*. The first steps to follow are listed below:

1. Open Matlab
2. Set the working directory to *your_path_to_scale/Scale/Scale_code_files/tester*
3. Type in the command prompt *guide GUI_Tester_DUMMY* (the call to the file has to be like this, be sure you do not call the file plus *.m* or *.fig*). Press enter and wait for the figure window to open.
4. Having the window open, click on *file* and then on *save* and save the figure as *GUI_Tester_TestName* (TestName refers to the name of the procedure, in this case has been saved as *GUI_Tester_MUSHRAMOD*).
5. Close the figure window.

6. If everything was done right, now two new files should have been created named *GUI_Tester_TestName.fig* and *GUI_Tester_TestName.m*. Those files refer to the figure file and the code file respectively (see Figure 3.9).

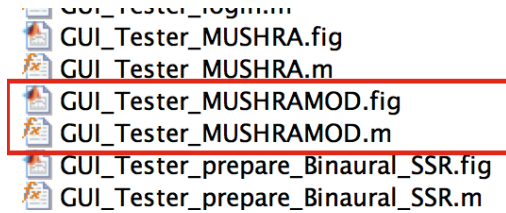


Figure 3.9: Newly created test files

```

function varargout = GUI_Tester_DUMMY(varargin) ...
% End initialization code - DO NOT EDIT

% --- Executes just before GUI_Tester_DUMMY is made visible.
function GUI_Tester_DUMMY_OpeningFcn(hObject, eventdata, handles, varargin) ...

% --- Outputs from this function are returned to the command line.
function varargout = GUI_Tester_DUMMY_OutputFcn(hObject, eventdata, handles) ...

function next_but_Callback(hObject, eventdata, handles) ...

function GUI_Tester_DUMMY_CloseRequestFcn(hObject, eventdata, handles) ...

function calibrate_Callback(hObject, eventdata, handles) ...

```

Figure 3.10: Code snippet of the default functions in the Tester window

Once those steps are performed we are ready to start modifying the file. The DUMMY file we start with, has some already some code written in it. Figure 3.10 shows a snippet of code of the functions that are already implemented. Those functions are the opening function, the closing function, the output function, the **calibrate** button click function and the **next** button click function. The code which have been written inside of them should not be deleted, however, if necessary might be expanded.

In GUIDE, when a window is opened, the function *Opening* is automatically called. In the *Opening* function, the actions which take place before the window opens have to be written. Those actions can, be for example, loading the results file in a variable, loading the logo images, write the instructions in the instructions field, etc.... The next function to be automatically called is the *output* function which is called right after the window appears. The *next* button click function is the function that will be called when the user has finished the trial, the code in it orders *Scale* to close the window and set the variable scenario completed on the results file to true. The *Calibrate* button function and the *Close* function will calibrate the tracker (when using the SSR) and close the scenario test respectively. Both of them do not require any modification.

Now we can start the tuning of the test window. For this, we will open the figure with *GUIDE* typing in the command prompt *guide GUI_Tester_MUSHRAMOD* (for our test MUSHRAMOD). The figure that will appear will look like Figure 3.11. It is

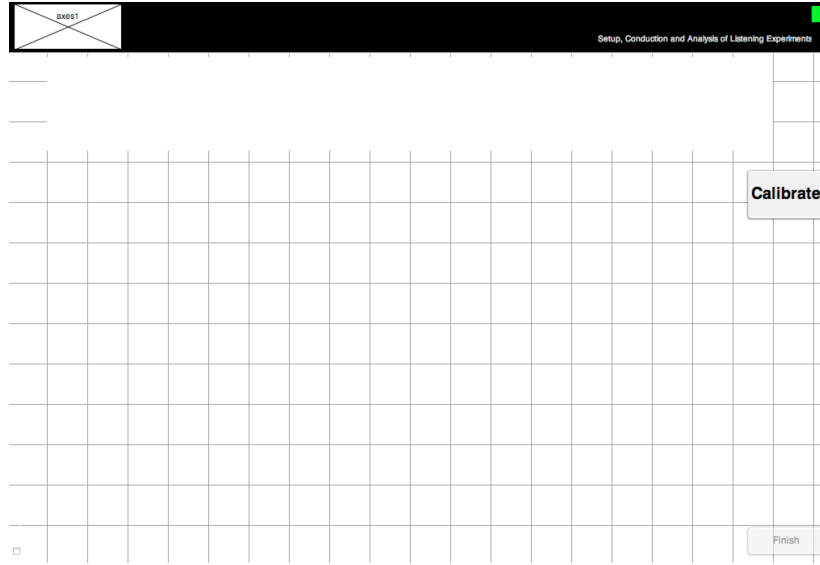


Figure 3.11: GUIDE edition mode of the default test window

nearly an empty window. From here on, the explanation will be directly based on the MUSHRAMOD test, since each test type has different layouts, presentation of stimuli or evaluation methods.

Since the MUSHRAMOD test has for each stimuli, a button, a slider and some y axis labels, we will create panels and inside each panel we will place those items. At this point the number of stimuli is not known, but since it should not be greater than 15, we will create 15 panels, please note that the panels which will be not needed will be just left invisible when the test is loaded. The buttons corresponding to each stimuli will be named alphabetically from A to O. To add the panels, buttons, ticks, etc, we will use the *GUIDE* editor. At this step we will also place the ratings (very far, far, close...) at the left part of the window.

Now the window looks like in Figure 3.12. In *GUIDE* every object or element in a window has a name (tag). The panels has been named as *Slider_pan_(number from 1 to 15)*, the sliders has been named as *slider_(number from 1 to 15)* and the stimuli buttons *play_(number from 1 to 15)*. The ticks on the y axis next to each slider are each a text field with the symbol -. They do not need to have a special name and have been added one by one. Once this is done, we need to click the save button on the guide window. It will take some time to process, and once it is done, *GUIDE* itself will have created for us all the callback functions (functions that are called when the element is clicked) of the buttons and the sliders. Those functions will be in the code file of the window and will be empty so we will have to fill them. Please note that the option *visible* of the slider panels has to be set as *off*. The option is accessed by clicking on the element in the *GUIDE* editor.

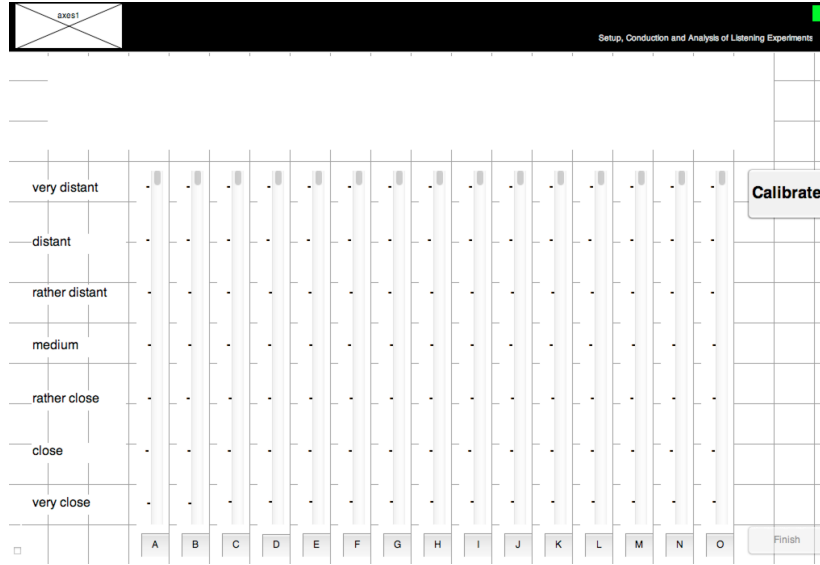


Figure 3.12: GUIDE edition mode of the test window with sliders

Once the modifications of the window are done and the functions corresponding to the buttons and sliders have been created we can start modifying the code. At this point we will go function by function.

GUI_Tester_MUSHRAMOD_OpeningFcn

In this function we will add the code necessary to turn visible the sliders needed. The code has to be added right at the end of the already written code. See Figure 3.13 to see the snippet of code of this part.

```
%%show the scenario number in the upper-right corner
set(handles.scenario_num_txt,'visible','on');
set(handles.scenario_num_txt,'string',num2str(handles.actualScenario));

guidata(hObject,handles);

%%make visible the needed question panels
for sliderIdx=1:handles.test_info.stimuli(handles.actualScenario).nStimuli
    set(eval(['handles.Slider_pan_' num2str(sliderIdx)]),'visible','on');
end
pause(0.1);
```

Figure 3.13: Code added to the GUI_Tester_MUSHRAMOD_OpeningFcn function

GUI_Tester_MUSHRAMOD_OutputFcn

This function will be called right after the test starts, since we want that the first sample starts playing automatically we will add the code necessary to activate stimuli placed in position 1. The type of stimuli used is not decided yet, for this reason, the implementation is done for both cases of existing stimuli types: WAV and BINAURAL_SSR.

In the case of normal WAV, we will just call the button 1 function as if the user clicked on stimuli in A. For the case of the BINAURAL, we have to start playing the *.wav* sample in a loop that will not be stopped until the trial is finished, so we do it here. Once the *.wav* sample is playing, we create a vector which will be used to keep track of the sources that are active and which ones are not. This vector is called *active_sources* and will be saved in the *handles* variable to be accessed from other functions. Each time a button is pressed, the vector is checked and the *SSR* closes all sources that are marked as active inside it. After that, the vector is set to zero, the new source is activated on the *SSR* and the position in the vector of the new source is set to 1. This part of the code can be seen in Figure 3.14.

```
function varargout = GUI_Tester_MUSHRAMOD_OutputFcn(hObject, eventdata, handles)
global Sample;
%%center window and hide loading panel
Common_position_window('defined',handles);
Common_loading_panel('off',handles);

switch handles.test_info.stimuliType
case StimuliTypeEnum.WAV
    play_1_Callback(handles.play_1, eventdata, handles);

case StimuliTypeEnum.Binaural_SSR
    %%start playing the .wav sample looped
    [Y,FS,NBITS]=wavread(fullfile('Scale_user_folder','tests',handles.test_info.testName,handles.test_info.stimuli(handles.actualScenario).wavList{1}));
    Sample=audioplayer(Y,FS,NBITS,handles.test_info.SSR_config.cardid);
    Sample.StopFcn=@(wavSampleLoop,Sample);
    play(Sample);

    %%retrieve scen information from the .asd file
    [scene,finished]=SSR_scene_parser(handles.test_info.stimuli(handles.actualScenario).asdFileName,fullfile('Scale_user_folder','tests',handles.test_info.testName));
    while(finished==0)
        pause(0.01);
    end
    handles.active_sources=zeros(1,scene.total_sources);
    handles.active_sources(1)=1;
    guidata(hObject,handles);

    play_1_Callback(handles.play_1, eventdata, handles)

end
guidata(hObject,handles);
```

Figure 3.14: Code added to the GUI_Tester_MUSHRAMOD_OutputFcn function

slider_(number of slider)_Callback

This function corresponds to the callback of each slider, this means the action that takes place when a slider is clicked. Basically the only think that happens is that the if the stimuli corresponding to the slider was not playing, it should be activated, if it was already playing we do not make nothing. The rating that the user gives, will be saved later in the moment when the scenario is closed.

Please note that in GUIDE when the code to perform is very similar we can call a function inside another function. For example, the code of the slider action can be written only once on the **slider_1_Callback and in the other sliders, we call this function. This is shown in Figure 3.15 highlighted in green.*

play_(number of button)_Callback

This function corresponds to the callback of each stimuli button. When a button is clicked two different things can happen. If the button was active, it has to be turned off and the playback stopped. If the button was not active, it should be activated and

```
function slider_1_Callback(hObject, eventdata, handles)
    name=get(hObject,'tag'); %%get the tag name of the slider element
    sliderClicked=str2num(name(8:end)); %%get the number of the slider extracting it from it's name

    %%If the stimuli corresponding to the slider is not already playing, make
    %%it play, this is checked evaluating the color of the button
    col=get(eval(['handles.play_' num2str(sliderClicked)], 'backgroundColor')); %%get the color of the button under the slider
    green=[0 1 0];
    if col~=green %%if the button is not green (not playing) call the button clicked function
        eval(['play_' num2str(sliderClicked) '_Callback(handles.play_' num2str(sliderClicked) ', eventdata, handles)']);
    end

function slider_2_Callback(hObject, eventdata, handles)
function slider_3_Callback(hObject, eventdata, handles)
function slider_4_Callback(hObject, eventdata, handles)
```

Figure 3.15: Code corresponding to the slider functions

the stimuli played. To know which stimuli corresponds to the button, the presentation order is checked. This first part can be seen in Figure 3.16 inside the red square.

The rest of the code corresponds to the playback of the stimulus. Since there are two type of stimuli implemented, we implement this two times.

In the first case, for a *WAV* type stimulus, we load the *.wav* file, stop the playback of the previous one, create a new audio object and start playing it. The code can be seen in Figure 3.16 inside the blue frame. For the *BINAURAL_SSR* stimuli type, since the *.wav* file is already playing and do not need to be changed we just activate the source in the *SSR* that corresponds to the stimulus and we turn off any other source that was active. This can be seen in Figure 3.16 inside the green frame.

Please note that in guide when the code to perform is very similar we can call a function inside another function. For example, the code of the button action can be written only once on the **play_1_Callback and in the other buttons, we call this function. This is the same as presented in Figure 3.15 highlighted in green.*

```
function play_1_Callback(hObject, eventdata, handles)
global Sample
%%set all buttons to grey
set(handles.play_ref, 'backgroundColor', [0.941 0.941 0.941]);
for butIdx=1:15
    set(eval(['handles.play_' num2str(butIdx)], 'backgroundColor'), [0.941 0.941 0.941]);
end

%%set clicked button to green
name=get(hObject, 'tag');
set(eval(['handles.play_' name(6:end)], 'backgroundColor'), [0 1 0]);

%%check the stimuli to be played looking the presentation vector
stimuliIdx=handles.results_info.MUSHR.scenario(handles.actualScenario).presentationOrder(str2num(name(6:end)));

%%now proceed to play the stimuli corresponding to the type of stimuli
%%selected
switch handles.test_info.stimuliType
case StimuliTypeEnum.WAV
    %%read the sample information
    fileName=fullfile('Scale_user_folder', 'tests', handles.test_info.testName, handles.test_info.stimuli(handles.actualScenario).wavList{stimuliIdx});
    [Y, FS, NBITS]=wavread(fileName);
    %%if there was a sample playing stop the loop and stop the sample
    if ~isempty(Sample)
        Sample.StopFcn=@(wvSampleStop, Sample);
        stop(Sample);
    end
    %%prepare the new sample and play it
    Sample=audioplayer(Y, FS, NBITS);
    play(Sample);
    Sample.StopFcn=@(wvSampleLoop, Sample);

case StimuliTypeEnum.Binaural_SSR
    asdSourceIdx=handles.test_info.stimuli(handles.actualScenario).BRIRList{stimuliIdx}; %%get the asd source idx
    sendCommandSSR('set_source sound', asdSourceIdx, handles.test_info.SSR_config.connectionIdx); %%activate the source
    active=find(handles.active_sources==1);
    for activeSourcesIdx=1:length(active) %%mute all the sources that were active
        if asdSourceIdx~=active(activeSourcesIdx)
            sendCommandSSR('set_source mute', active(activeSourcesIdx), handles.test_info.SSR_config.connectionIdx);
        end
    end
    handles.active_sources(1:end)=0; %%save the information of the active source
    handles.active_sources(asdSourceIdx)=1;

end
guidata(hObject, handles);
```

Figure 3.16: Code corresponding to the stimuli button functions

next_but_Callback

This function is activated when the user clicks the next button indicating that he has finished the trial. Most of the code for this function do not need to be written since it is the same for all type of test procedures. This common part corresponds to the stop of the playback in case a sample was sounding, the saving of the results file, the closing of the window and the opening of the scenario selection window.

The part that has to be implemented is the saving of the ratings which we will code before the existing code, as shown in snippet 3.17 framed in red.

```
function next_but_Callback(hObject, eventdata, handles)
    global Sample
    for sliderIdx=1:handles.test_info.stimuli(handles.actualScenario).nStimuli(handles.actualScenario)
        stimuliIdx=handles.results_info.MUSHRAMOD.scenario(handles.actualScenario).presentationOrder(sliderIdx);
        handles.results_info.MUSHRAMOD.scenario(handles.actualScenario).sampleRating(stimuliIdx)=round(get(eval(['handles.slider_' num2str(sliderIdx)]), 'value'));
    end
    switch handles.test_info.stimuliType
        case StimuliTypeEnum.WAV
            if ~isempty(Sample) %%stop the wav sample
                Sample.StopFcn=@wavSampleStop,Sample;
                stop(Sample);
            end
        case StimuliTypeEnum.Binaural_SSR
            if ~isempty(Sample) %%stop the wav sample
                Sample.StopFcn=@wavSampleStop,Sample;
                stop(Sample);
            end
    end
    sendCommandSSR('clear scene',0,handles.test_info.SSR_config.connectionIdx); %%clear the loaded scene
end

%%set the scenario completed flag and save the results info
handles.results_info.completedScenarios(handles.actualScenario)=1;
guidata(hObject,handles);
results_info=handles.results_info;
save(fullfile('Scale_user_folder','tests',handles.test_info.testName,'results',handles.results_code),'results_info');

%%close scenario and open scenario selection window
pause(0.1);
delete(eval(handles.fig_name));
pause(0.2);
GUI_Tester_scenario_selection(handles.test_name,handles.results_code);
return;
```

Figure 3.17: Code corresponding to the next button function

As seen in the snippet, to save the rating, we go slider by slider checking the value and using the presentation order vector we match each position with the real stimuli positions. This means that if slider number 1 played stimulus number 4, in position 4 of the sample rating vector the value of slider 1 will be saved.

3.3 Analyzing process file changes

For this part two new files have to be created

1. analysing_generate_struct_TestName
2. GUIAnalyse_TestName

3.3.1 analysing_generate_struct_TestName

This file corresponds to a script which is called during the analysis process in order to export and combine the ratings of the selected subjects and some information about them. The script is found in *your_path_to_scale/Scale_code_files/analysing/analysing_generate_struct_DUMMY*. The first thing to do is open the script code in *Matlab* and save it as *analysing_generate_struct_TestName*. In our case it will be saved as *analysing_generate_struct_MUSHRAMOD*.

Once we have our own file, we can start the code modification. The first part of the script is a *for* loop repeated once for each result file. Inside the loop, two parts can be distinguished.

The first one is dedicated to save the subject information such as age, gender or experience. This is shown in Figure 3.18 and this part does not need any modification.

The second part is dedicated to save the ratings (Figure 3.19 red frame) and this part needs to be implemented for each type of test.

```

%%%check if the results file correspond to someone, if not skip them
existSubject=1;
try load(fullfile('Scale_user_folder','subjects',subjectID{resIdx,1}))
catch errorL
end
if exist('errorL')
    if strcmp(errorL.identifier,'MATLAB:load:couldNotReadFile')
        existSubject=0;
    end
end

%%%if they correspond add age, gender, and others...
if existSubject
    Surname{resIdx,1}=subject_info.Surname;
    Name{resIdx,1}=subject_info.Name;
    Gender{resIdx,1}=subject_info.Gender;
    date_today=date;
    year_now=str2double(date_today(end-3:end));
    Age{resIdx,1}=year_now-subject_info.B_year;
    Country{resIdx,1}=subject_info.Country;
    Experience{resIdx,1}=subject_info.Experience;
    Audio_Knowledge{resIdx,1}=subject_info.Audio_Knowledge;
    Hearing_Problems{resIdx,1}=subject_info.Hearing_problems;
else
    Surname{resIdx,1}='unknown';
    Name{resIdx,1}='unknown';
    Gender{resIdx,1}='unknown';
    date_today=date;
    year_now=str2double(date_today(end-3:end));
    Age{resIdx,1}=NaN;
    Country{resIdx,1}='unknown';
    Experience{resIdx,1}='no';
    Audio_Knowledge{resIdx,1}='no';
    Hearing_Problems{resIdx,1}='no';
end

```

Figure 3.18: Code corresponding to the subject information part of the results struct creation

```

        Audio_Knowledge{resIdx,1}='no';
        Hearing_Problems{resIdx,1}='no';
    end

    %%if subject did the scenario add the data, if not, add NANS
    for scenIdx=1:handles.test_info.totalScenarios
        if results_info.completedScenarios{scenIdx}
            eval(['Scen' num2str(scenIdx) '(' num2str(resIdx) ',:)=results_info.MUSHRA.scenario(' num2str(scenIdx) ').sampleRating;'])
        else
            eval(['Scen' num2str(scenIdx) '(' num2str(resIdx) ',1:handles.test_info.stimuli(' num2str(scenIdx) ').nStimuli)=NaN;'])
        end
    end
end

```

Figure 3.19: Code corresponding to the ratings information part of the results struct creation

Once the *for* loop is completed only two things remain. The first one is to get the maximum length of the results vector. Since different trials can have different number of stimuli, we want to get the one with the maximum length for plotting purposes later in the plotting window. This can be done as represented in Figure 3.20 inside the red frame. This part of code needs to be specifically implemented for each type of test. The remaining code corresponds to the saving of the vectors created into a struct. This struct will be the one used for the plotting and the one that will be exported when the user wants to save the results in an external file. This last part of the code Figure 3.20 (not framed), needs no modification.

```

%%get the max value to prepare the info vectors
max=0;
for scenIdx=1:length(selected)
    num=handles.test_info.stimuli(selected(scenIdx)).nStimuli;
    if num > max
        max=num;
    end
end

%%create the struct object
value={};
field='subjectID';
eval(['data_subs.' field '=' field ';']);
field='resultsCode';
eval(['data_subs.' field '=' field ';']);
field='Surname';
eval(['data_subs.' field '=' field ';']);
field='Name';
eval(['data_subs.' field '=' field ';']);
field='Gender';
eval(['data_subs.' field '=' field ';']);
field='Country';
eval(['data_subs.' field '=' field ';']);
field='Experience';
eval(['data_subs.' field '=' field ';']);
field='Audio_Knowledge';
eval(['data_subs.' field '=' field ';']);
field='Hearing_Problems';
eval(['data_subs.' field '=' field ';']);

for scenIdx=1:handles.test_info.totalScenarios
    field=['Scen' num2str(scenIdx)];
    eval(['data_subs.' field '=' field ';']);
end

handles.data_subs=data_subs;

```

Figure 3.20: Code corresponding to the completion of the results struct creation

3.3.2 GUI_Analyse_TestName

This file is responsible for the analysis of the results window. Since tests can be very different from each other, a new file has to be implemented for each new test. This part will require medium-high programming skills in *Matlab* and specially *GUIDE*. The first steps to follow are listed below:

1. Open Matlab
2. Set the working directory to *your_path_to_scale/Scale/Scale_code_files/analysing*
3. Type in the command prompt *guide GUIAnalyse_DUMMY* (the call to the file has to be like this, be sure you do not call the file plus *.m* or *.fig*). Press enter and wait for the figure window to open.
4. Having the window open, click on *file* and then on *save* and save the figure as *GUIAnalyse_TestName* (TestName refers to the name of the test procedure, in this case has been saved as *GUIAnalyse_MUSHRAMOD*).
5. Close the figure window.
6. If everything was done right, two new files should have been created with the names *GUIAnalyse_TestName.fig* and *GUIAnalyse_TestName.m*. Those files refer to the figure file and the code file respectively. (Figure 3.21)

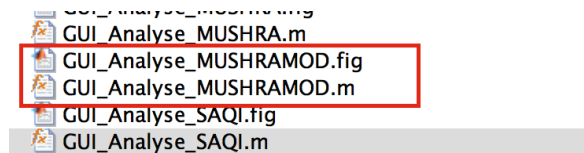


Figure 3.21: Newly created analysis files

The window, which have been created by default, will look like in Figure 3.22 and in most procedures, such as ours, will not require any changes in its layout but only in the code.

The analysis window includes many elements and many functions. For most tests, the rating include only values of one variable. Depending on the test, it may include one or more values for each trial. In the case of the implemented test, *MUSHRAMOD*, we will have to represent several values for each trial.

The snippet of code where this is done is presented in Figure 3.23. Actually only two lines of code need to be added (the ones inside the red frame). In the dummy file, those lines are already written but commented. For some procedures they cpuld even work without needing any modification. Please note that this part of the code is found inside the *refresh_but_Callback* function.

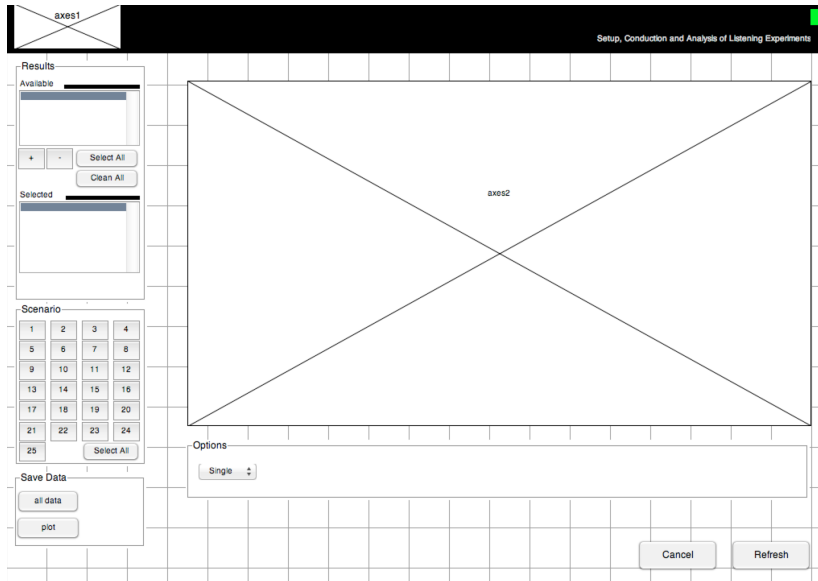


Figure 3.22: Analyse window in edition mode

```

187 - legend_count=1;
188 - if or(strcmp(handles.visualisationOption,'single'),strcmp(handles.visualisationOption,'combined')),length(handles.data_subs
189 - for resIdx=1:length(handles.data_subs.subjectID)
190 - for scenIdx=1:length(selected)
191 - temp=eval(['handles.data_subs.Scen' num2str(selected(scenIdx)) '(' num2str(resIdx) ',:')]);
192 -
193 - Subj_code=eval(['handles.data_subs.resultsCode(' num2str(resIdx) ')']);
194 - vec(legend_count,:)=temp NaN(1,max-length(temp));
195 - leg(legend_count)=Subj_code(1) 'Scen' num2str(selected(scenIdx));
196 - color(legend_count,:)=handles.colors(resIdx,:);
197 - plot_type(legend_count)=1;
198 - plot_width(legend_count)=1;
199 - plot_marker(legend_count)='o';
200 - plot_style(legend_count)='-.';
201 - if length(handles.data_subs.subjectID)==1
202 - plot_style(legend_count)='-';
203 - plot_width(legend_count)=2;
204 - color(legend_count,:)=handles.colors(selected(scenIdx),:);
205 - end
206 - legend_count=legend_count+1;
207 - end
208 - end
209 - end
210 - if and(strcmp(handles.visualisationOption,'average'),strcmp(handles.visualisationOption,'combined')),length(handles.data_sul
211 - for scenIdx=1:length(selected)
212 - temp=eval(['nanmean(handles.data_subs.Scen' num2str(selected(scenIdx)) ')']);
213 -
214 - vec(legend_count,:)=temp NaN(1,max-length(temp));
215 - leg(legend_count)='Scen' num2str(selected(scenIdx));
216 - plot_type(legend_count)=2;
217 -

```

Figure 3.23: Code corresponding to the analysis window modification

Bibliography

- [1] H. Levitt, *The City University of New York*. PhD thesis, University of Wollongong, 1970.
- [2] E. Zwicker and H. Fastl, *Psychoacoustics. Facts and Models*. Springer, 2nd ed., 1999.
- [3] I.T.U., “Methods for the subjective assessment of small impairments in audio systems including multichannel sound systems,” *ITU-R BS*, no. 1116-1, 1997.
- [4] I.T.U., “Method for the subjective assessment of intermediate quality level of coding systems,” *ITU-R*, vol. BS, pp. 1534–2, 2014.
- [5] A. Lindau, V. Lepa, *et al.*, “A spatial audio quality inventory (saqi),” *Acta Acustica united with Acustica*, vol. 100, pp. 984–994, 2014.
- [6] M. Geier, J. Ahrens, and J. Spors, “The soundscape renderer: A unified spatial audio reproduction framework for arbitrary rendering methods,” *Proceedings of the 124th Convention of the AES*, no. 7330, 2008.

.1 Manual

Scale

Setup, Conduction and Analysis of Listening experiments

User Documentation

Release: R16-0726 (third release)

July 3, 2016

Contents

1	Introduction	4
1.1	What is Scale?	4
1.2	Tests implemented	4
1.2.1	Adaptive	4
1.2.2	Double-blind Triple-stimulus with Hidden Reference (ABC-HR)	5
1.2.3	ABX	6
1.2.4	MUSHRA	6
1.2.5	SAQI	6
2	Type of stimuli available	7
2.1	.wav stimuli (WAV)	7
2.2	.wav combined with the SSR (BINAURAL_SSR)	8
2.3	Contact	8
3	Installing and starting the program	9
3.1	Running the program for the first time	9
3.2	Running an already installed version	10
3.3	Running an installed version from another computer	10
4	Running the program - Overview	11
5	Running the program - Create	12
5.1	General configuration window	12
5.2	Set Instructions Window	12
5.3	Procedure specific options	13
5.3.1	Adaptive	13
5.3.2	SAQI	14
5.3.3	ABC-HR	15
5.3.4	ABX and MUSHRA	16
5.4	Stimuli configuration	17
5.4.1	WAV	17
5.4.2	BINAURAL SSR	18
5.4.3	Order and number of stimuli	19
5.4.4	Finish	20
5.4.5	Files created	20
6	Running the program - Perform	21
6.1	Test preparation	21
6.2	Subject login	22
6.3	Test Introduction Window	22
6.4	Scenario Selection Window	23
6.5	Trial Window	24
6.5.1	Adaptive Methods	24
6.5.2	ABC-HR method	25
6.5.3	ABX method	26

6.5.4	MUSHRA Method	27
6.5.5	SAQI method	27
7	Running the program - Analyse	29
7.1	Analyse Window - Overview	29
7.2	Analyse Window - Adaptive, ABC-HR, ABX and MUHSRA	30
7.3	Analyse Window - SAQI	30
7.4	Exported Data	31
7.4.1	Common fields	31
7.4.2	Adaptive	31
7.4.3	ABC-HR	33
7.4.4	ABX	33
7.4.5	MUSHRA	33
7.4.6	SAQI	33
8	Closing Scale	34
9	The Sound Scape Renderer	35
9.1	Introduction	35
9.2	Sound Scape renderer and Scale	36
9.3	Installation	36
10	Jack Audio Connection Kit	36

1 Introduction

1.1 What is Scale?

Scale is a software tool that covers the full chain of setup, conduction and analysis of psychoacoustic experiments. It offers several testing procedures and the interaction between researcher or subjects and the software is done via a graphical user interface (GUI) and does not require any programming skills. Test setups or results can be easily ported from one instance of the program to another. Thus everything is portable and exchangeable between different computers and researchers. The first version of *Scale* was presented at the DAGA Conference in 2013 in Merano [1], Italy and the second version was presented at the DAGA Conference in 2015 in Nürnberg [2].

1.2 Tests implemented

The initial version of *Scale* included a selection of frequently used test procedures like simple or transformed staircase adaptive procedures and double blind triple-stimulus with hidden reference. A description of the latter can be found in [1]. In the second version three additional procedures, ABX, MUSHRA and SAQI, are implemented. In this third version, no more tests have been included however, the possibility of adding new procedures haven been made possible.

1.2.1 Adaptive

Adaptive procedures aim to find a threshold of detection in the psychometric function of a determined dimension of a sound. Stimuli are presented and varied in one dimension. The amount of variation is increased or reduced depending on the preceding subject's responses and on the respective adaptation method. In *Scale* the adaptation is made using different staircase methods like simple staircase (1up/1down) on the one hand, and some transformed staircase methods such as 1up/2down or 2up/1down as described in [3] on the other hand. The threshold estimation can either vary depending on the ending conditions of the trial (limited number of runs or reversals) or on how the average is calculated. In this version the threshold estimation using *QUEST* [4] has been also added.

The tests have to be combined with a paradigm. *Scale* provides different paradigms which can be divided into two groups: n-AFC (n alternative forced choice) paradigms and Yes/No paradigms. In the n-AFC paradigms n intervals (as used in [5]) are presented to the subject. When n is equal to 2 the subject has to decide in which of the two intervals a designated signal is present. When n is greater than 2 the subject has to decide in which of the n intervals the presented stimulus is different. The sample assignment to the intervals of the n-AFC paradigms is always automatically randomised. When using a Yes/No paradigm only one interval that includes one or more sounds is presented to the subject. The subject has to decide whether the signal occurs within the presented interval or not.

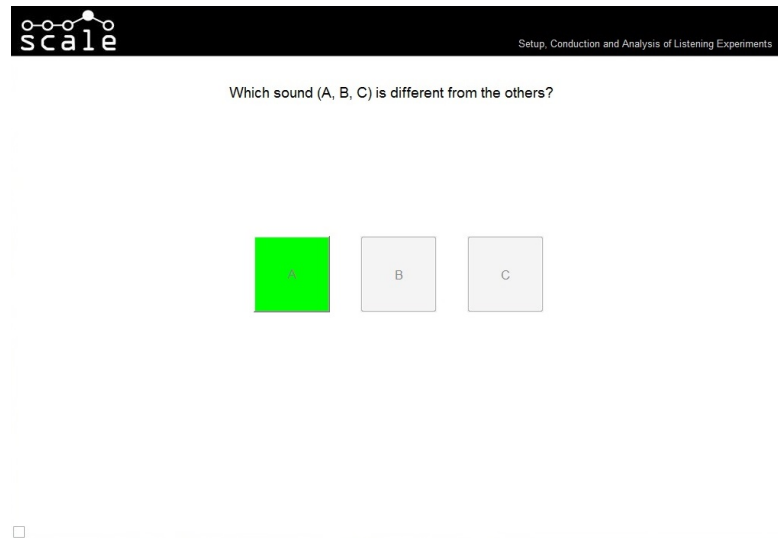


Figure 1: 3-AFC procedure trial window

1.2.2 Double-blind Triple-stimulus with Hidden Reference (ABC-HR)

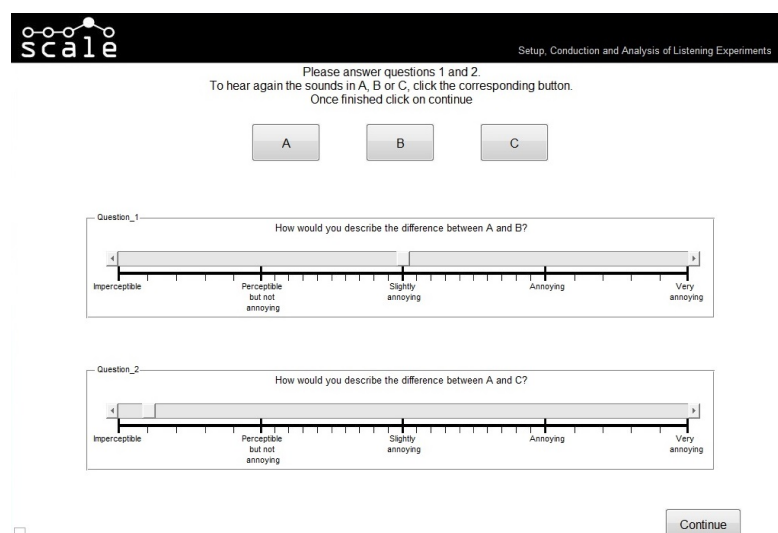


Figure 2: Double-blind triple-stimulus with hidden reference procedure trial window [6]

This procedure has become a standard in psychoacoustics and is used to assess small impairments between sound samples. In every trial, three stimuli are presented in three intervals ("A", "B" and "C"). The stimulus in "A" is presented as the known reference, the stimuli in "B" or "C" are randomly assigned, whereas one of them is a hidden reference and the other one is a sample which is varied in one determined dimension. After listening to the stimuli the subject is asked to assess the impairments between "A" and "B" and "A" and "C" using a rating scale. The rating is performed with a slider along a continuous scale with anchors. The number of grades in the scale as well as the text in the labels of every mark can be set. Thus all requirements to perform the test "*Subjective Assessment of Small Impairments in Audio Systems Including Multichannel Sound Systems*" described in the recommendations of the ITU [6] are met (see Figure 2).

1.2.3 ABX

The ABX test is a simplification of the *ABC-HR* test. In a trial of an *ABX* test, a subject is presented with two stimuli which are A and B, followed by a third one called X. After hearing A, B and X, the subject must select which of the stimuli in the intervals A or B is the same as in interval X. In this case, as opposite to the *ABC-HR*, no rating is done. In Scale, each scenario can contain many trials and for each trial, the position of the reference stimuli in A and B is randomized. Figure 3 shows a trial window of the test.

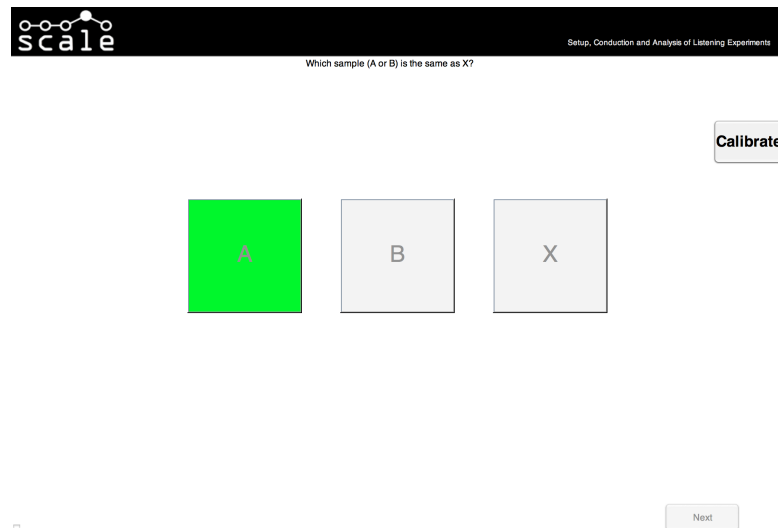


Figure 3: *Scale*'s interface during an ABX test

1.2.4 MUSHRA

The aim of a multi-stimulus test with hidden reference and anchor (MUSHRA) test [7] is to rate global differences between several audio stimuli. All stimuli are presented in a single trial and have to be compared to a given reference. Each stimulus has a continuous scale (continuous quality scale) which goes from 0 (bad) to 100 (excellent), as shown in Figure 4. Reference and stimuli can be switched over instantly. The order of the stimuli is randomized and every trial has to include a hidden reference and an anchor.

1.2.5 SAQI

The spatial audio quality inventory (SAQI) test [8] has been specifically designed for the perceptual evaluation of virtual acoustic environments. In every trial a reference and a stimulus are presented together with 48 verbal descriptors of perceptual qualities that are assumed to be of practical relevance when comparing virtual auditory environments. Each descriptor comprises a rating scale with a pair of opposed adjectives in its scale ends. The subject's task is to compare the stimulus to a given or inner reference and give a rating for each perceptual quality.

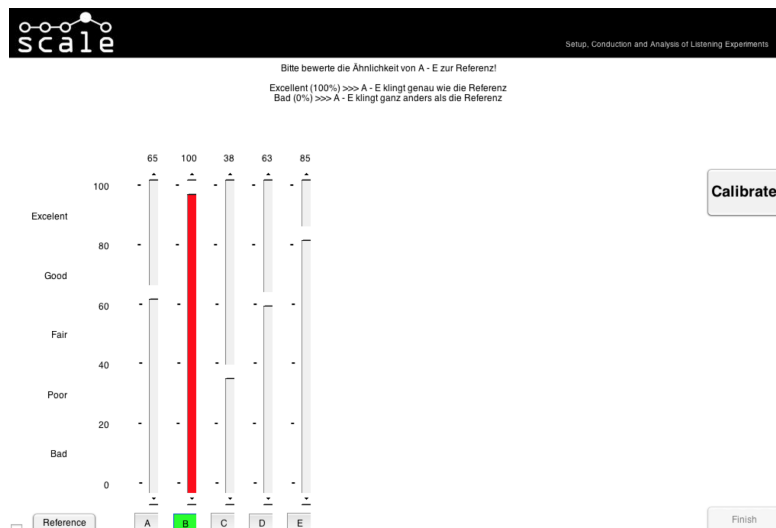


Figure 4: Scale's interface during a MUSHRA test

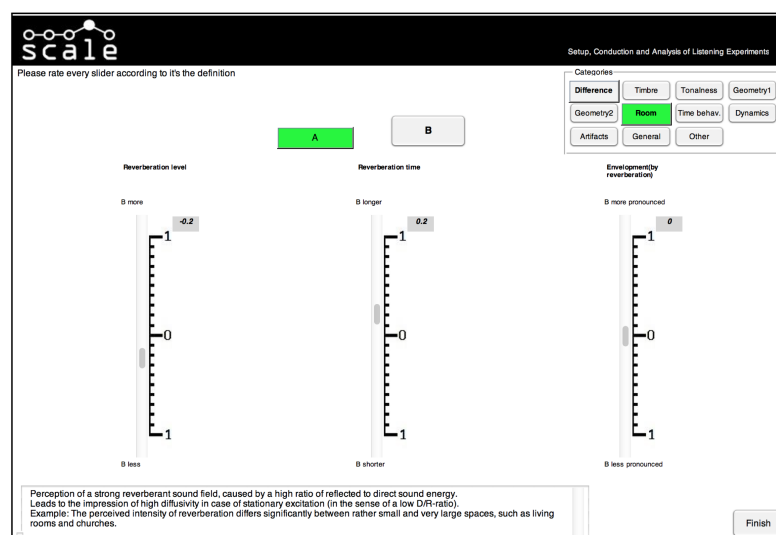


Figure 5: Scale's interface during a SAQI test

2 Type of stimuli available

In this version of Scale two different types of stimuli are available. Those are *.wav* stimuli and *.wav combined with SSR*. Other types of stimuli can be added by modification of the existing code files.

2.1 *.wav* stimuli (WAV)

The *.wav* stimuli are just audio files in the mentioned format which will be played by *Matlab* using the function *play* of the audioplayer objects.

2.2 .wav combined with the SSR (BINAURAL_SSR)

This type of stimuli are obtained by a combination of the rendering software Sound Scape Renderer (SSR) [9] and the .wav playback function of Matlab. As shown in Figure 6, *Scale* processes subject's inputs and operates the *SSR* using its network interface via TCP/IP protocol while the test is performed. The *SSR* runs in the background generating stimuli with the combination of the incoming audio signal, the tracker data and the corresponding head related impulse response (HRIR) or binaural room impulse response (BRIR) set.

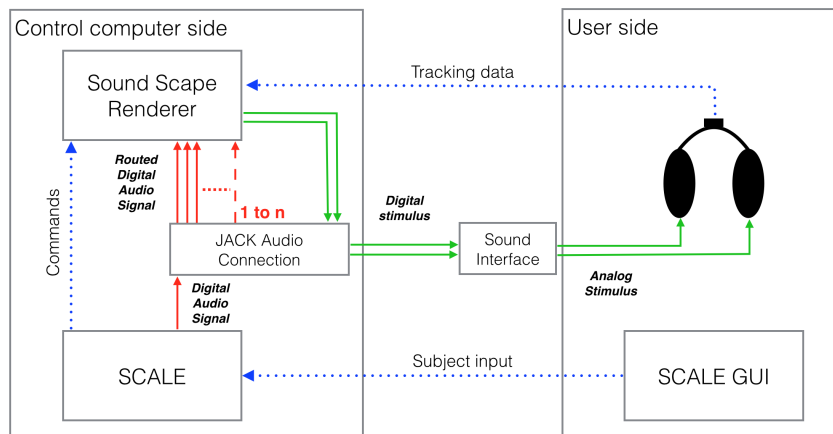


Figure 6: System architecture

2.3 Contact

For any further requests, technical support or bug reports feel free to contact:

- Arnau Vázquez-Giner, arnau.vazquez@gmail.com

For software updates and further information visit:

<http://www.audiogroup.web.th-koeln.de>

3 Installing and starting the program

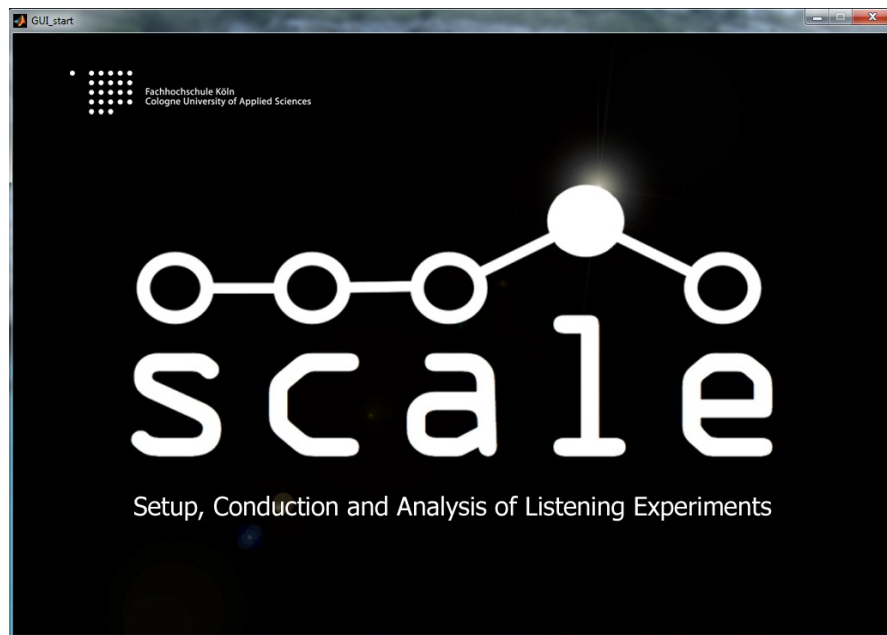


Figure 7: Program loading window

In this chapter, the instructions to start the program will be given.

3.1 Running the program for the first time

1. Download the item **Scale** from the following link:
<http://www.audiogroup.web.th-koeln.de>
2. Extract the file **Scale.zip** to a directory of your choice.
3. Once extracted open *Matlab* and set the current *Matlab* folder to the directory where you extracted the program (you should see a .m file with the name Scale and the folders Scale_code_files and Scale_user_folder).
4. Type **Scale** on the command line and press enter.
5. Scale will open and the window presented in Figure 7 should appear. After this point, you will not need to interact with *Matlab* anymore, but do not close it!

3.2 Running an already installed version

To run an installed version of Scale follow the steps mentioned in the last section starting at point 3.

3.3 Running an installed version from another computer

In case that you want to import your version of Scale, with your already configured test or results, just copy the folder Scale into the new computer. All the implemented tests and results will be automatically imported.

**Please note that if you are using the SSR stimuli type, you will need to adapt the .asd files*

4 Running the program - Overview

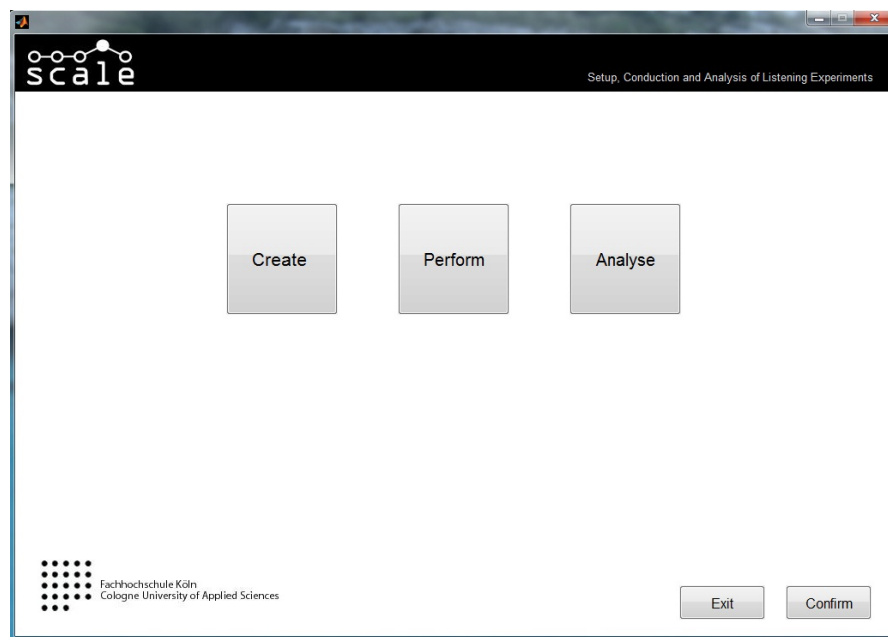


Figure 8: Main window

When the program has finished loading the **Main Program** window appears. This window appears also when the user at some point interrupts the program or decides to go back. The window contains three buttons (**Create**, **Perform** and **Analyse**) referring each to one of the basic features. When pressing any of the buttons, it will turn green and one of the actions described in the next lines will take place.

Create: Select this option to create a new test.

Perform: Select this option to conduct a test. A list with the available tests in the `scale_program_folder\tests` will appear. Select from the list the test to be performed and press **Continue**.

Analyse: A list with the available tests in the `scale_program_folder\tests` will appear. Select one test from the list and press the button **Continue**.

**If the test does not contain any result files, the action will be avoided!*

5 Running the program - Create

Results from two similar tests can be affected by several factors. On the one hand there are the external factors like the devices used along the playback chain and their settings or the characteristics of the experiment's location. On the other hand there are factors like e.g. the sequence of the presented stimuli or the information given to subjects in advance of or during the test. Taking the latter into account, different settings are adjustable in *Scale*. The settings are accessed in the creation module where different configuration options are available depending on which type of procedure is selected.

In this section the steps to follow during the creation of a test are described.

5.1 General configuration window

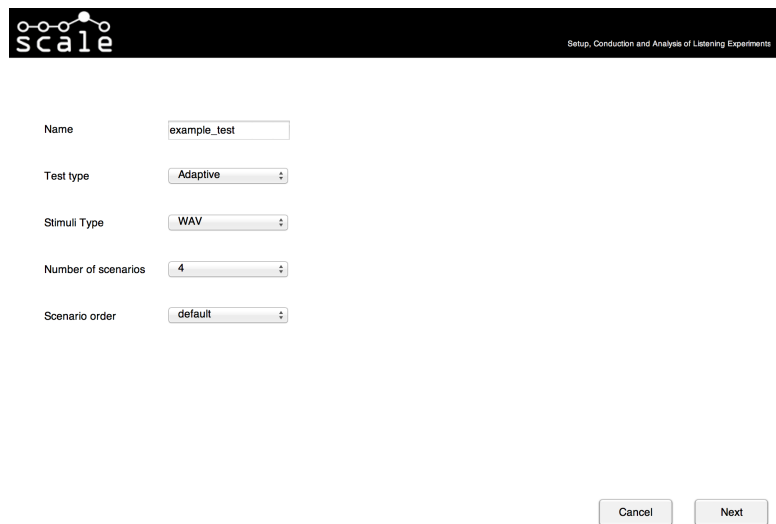


Figure 9: General configuration window

Figure 9 shows the general configuration window. This window is the first one to appear after selecting the option create in the main program window. Here the name of the test, its procedure, the stimuli type, the number of scenarios and the presentation order of them can be set. The order of the scenarios can be set to default or random. Default, means that scenario 1 will be performed in the first place, scenario 2 in the second place and scenario n in the nth place.

5.2 Set Instructions Window

Scale allows the researcher to write the instructions and commands that are shown to the subject during the test performance. The second window of the test configuration, presented in Figure 10, is used for setting those instructions. On the one side, the instructions and/or information that the subject will see at the beginning of their test is written on the panel **Instructions on test start** (upper panel). On the other side, the instructions that appear when the subject performs a run are written on the panel **Instructions on trial window** (lower panel). Although the instructions can be the same

for each scenario, the process must be done separately for each trial by clicking on the scenario buttons. To go faster, you can copy paste the text from one scenario to the other.

Figure 10: Set Instructions Window

5.3 Procedure specific options

In this window the specific configuration for each test type can be done. Depending on the procedure, the window will have different parameters. In some cases no specific parameters will be needed. Following, the window is presented and the options are explained separately for each procedure.

5.3.1 Adaptive

For the configuration of the adaptive tests, the configuration window will look like in Figure 11. The **general configuration panel** (upper panel) includes the parameters which are common for all adaptive tests. Those are, the number of repetitions, the paradigm and the adaption method.

Repetitions - Select the maximum number of times that a subject can perform a repetition in one run.

Paradigm - Select the paradigm (2AFC, 3AFC or Yes/No)

Adaptation method - Select the adaption method from one of the available (1up1down, 2up1down, 1up2down and QUEST).

Depending on the adaption method chosen, you will have to configure one of the panels below. For *1up1down*, *2up1down*, *1up2down* the panel to configure is the lower-left one (**staircase**). For *QUEST* it is the lower right one (**QUEST**).

Figure 11: Specific options for the adaptive tests

The options for the staircase methods are the following:

Number of runs - The trials end after a certain number of performed runs. If the option is left to 10000, this ending criteria will be ignored.

Number of reversals - The trials end after a certain number of reversals (changes from correct to false or false to correct subject decisions). If the option is left to 10000, this ending criteria will be ignored.

Activate half step width - If selected, next played stimuli after a correct/false decision will be the stimuli situated two positions above/below the preceding one. After the first reversal, the steps will be reduced to only one position.

The options for the *QUEST* test are: **tGuess**, **tGuessSD**, **pThreshold**, **stepSize**, **range**, **nTrials**, **beta**, **delta** and **gamma**. Those are described in the documentation of the *QUEST* Toolbox in [4].

5.3.2 SAQI

For the configuration of the *SAQI* tests, the configuration window will look like in Figure 12. The window includes a panel with general procedure options (lower panel) and the panels with the qualities that the researcher wants to get the ratings for.

Reference Type - In a *SAQI* test, the reference can be a stimuli (external) or a concept (internal). An internal reference example could be the sound of a guitar, so the subject would have to compare some stimulus with the idea of the sound of a guitar that he has in his memory instead of hearing a real guitar.

Figure 12: Specific options for the SAQI tests

Slider steps - In a *SAQI* test, every quality is evaluated using a slider. Here the number of steps of the sliders can be selected.

test language - Two languages are available for this type of test, those are English and German. The names of the qualities and the descriptions of them appearing during the test will be shown in the language selected.

To select a quality just click on the button next to its name. You can select all of them at once by clicking the **select all** button. You can also deselect qualities by clicking on their buttons again. Clicking on the button of a quality will also display an explanation of itself on the lower-right corner of the window.

5.3.3 ABC-HR

For the *ABC-HR* tests, the window will look like in Figure 13. Following each parameter, button and element in the window is explained.

Instructions - Here can be written the instructions that the subject will see right at the top of the question.

values... (*input text field*) - In those fields must be written the labels that the subject will see under the slider and the scale picture when performing the rating. The label in the left will correspond to the value 0 of the slider while the label in the right will correspond to the *max* value of the slider.

Figure 13: Specific options for the ABC-HR tests

preview - A short preview of two seconds showing how will look like the question in a real test situation. This option is interesting to see if the text in the labels will be correctly placed in the real test.

scale picture (*list*) - This list let the researcher chose the image that will be displayed between the slider and the labels. When a picture is selected it is automatically displayed.

values (*list*) - Select the number of values that will appear in the rating scale. The number of fields will be automatically modified, and so their positions under the slider that will be always symmetrically distributed. The value in the left corresponds always to 0 while the value in the right, *max*, will correspond to the number selected in values, *n*, minus 1 ($max = n - 1$).

slider size (+/-) - Press this button to increment/decrement the width of the slider thumb.

slider size (default) - Press this button to set the width of the slider thumb to a default value depending on the number of values.

5.3.4 ABX and MUSHRA

For both *ABX* and *MUSHRA* tests, no specific options need to be configured. The window will look in both cases like in Figure 14.

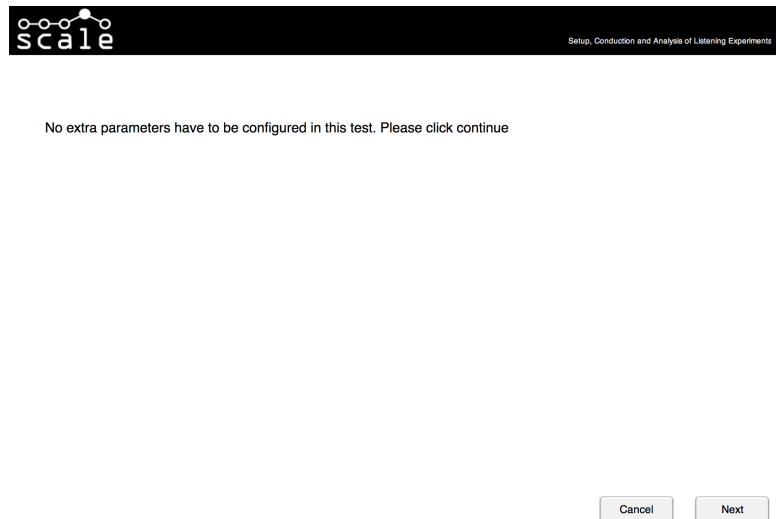


Figure 14: Specific options for the MUSHRA and ABX tests configuration

5.4 Stimuli configuration

In this version of *Scale* two types of stimuli are available, those are simple *.wav* samples and *.wav* samples processed by the SSR. The process of adding the stimuli is different for each type of stimuli, for this reason both procedures are explained separately. At the end of the section, some hints about the number of stimuli to be added in each type of test are given.

5.4.1 WAV

Figure 15 shows the window where the *.wav* samples are added. The first step is to add the samples into the program folder so they are selectable for the test. This is done by pressing the button **add** located inside the **Available samples** panel (right). When the button is clicked a dialog window opens and the user can select the samples from the directory. Samples can be selected one by one or several at the same time. This action can be performed as many times as the user wants. Once the samples are loaded they will appear in the list of the same panel.

The panel samples contains the slots where the samples can be added to the test. To add or delete slots the buttons **erase last slot** or **add new slot** at the lower right corner of the panel can be used. The lists of samples inside the slots, correspond to the list on the panel of available samples. For each slot we will select one sample. This process must be made for each scenario. To change the scenario, use the scenario buttons on the lower-left panel **Scenario**. More information about the samples order in each type of test is given at the end of this section.

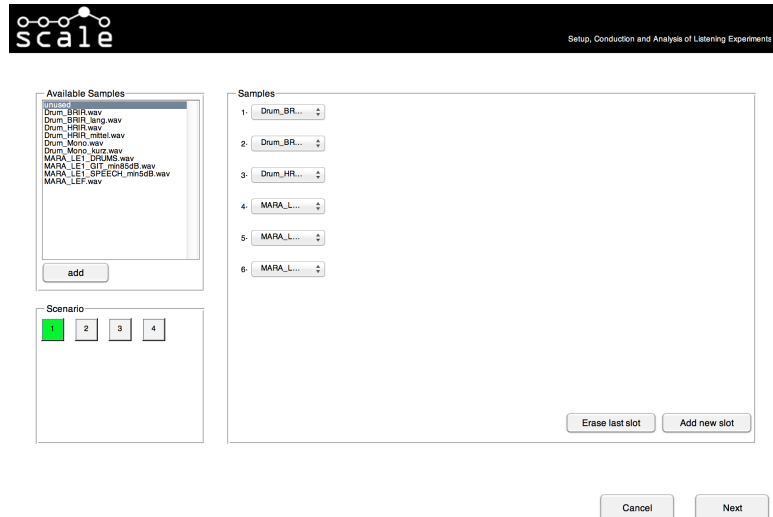


Figure 15: .wav stimuli setting window

5.4.2 BINAURAL SSR

BINAURAL SSR refers to the type of stimuli that use the *SSR* renderer. Two parts are involved in the addition of the stimuli in this case. The first part is the setting of the filters and the second the setting of the .wav file.

Figure 16 shows the window where the filters used for the stimuli are added. We will add one filter for each stimulus. On the right of the window we can select from the options available, the sample rate of the filter files and also the tracker that will be used. On the main panel (left) the user will add the filters on the slots. Each slot corresponds to one stimuli. The variable **volume** refers to the volume that the source will have in the .asd file and **x** and **y**, the position on the scene. Since the *SSR* will work only in BRIR mode, **x** and **y** will not affect the sound at all. To select the filters, we must click the button **select**.

After clicking, a dialog window will open where the folder containing the BRIR files can be selected. Once selected, the files will be selectable on each slot. To add or delete slots, the buttons **Erase last sample** and **Add sample** can be used. More information about the samples order in each type of test is given at the end of this section.

The second part of the stimuli configuration consists in adding the .wav files. Only one .wav file will be added for each trial or scenario.

After adding the filters, and .asd file (scene file used by the *SSR*) is created. This file contains the paths to the filters, for this reason, the filters should not be moved from their folders after a test is created. If the test needs to be played in another computer, the .asd file should be modified and the new paths should be added by hand. The .asd files are placed in *Scale/Scale_user_folder/Tests/<nameOfTheTest>*.

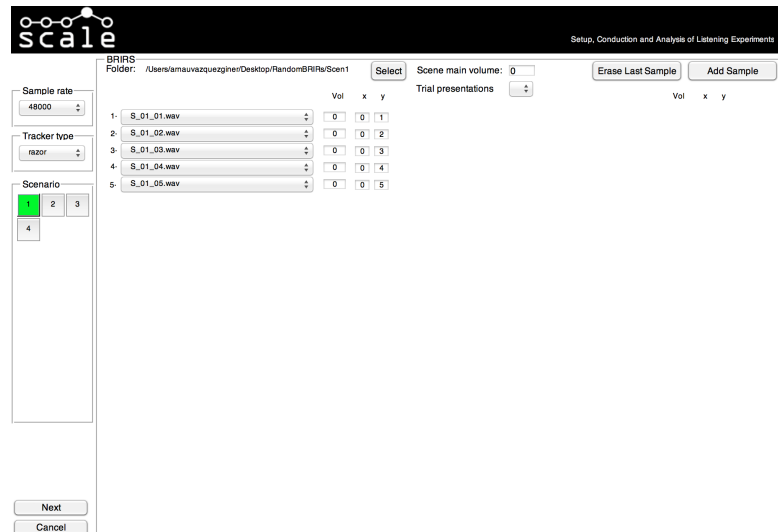


Figure 16: Setting of the filters for the BINAURAL SSR stimuli type

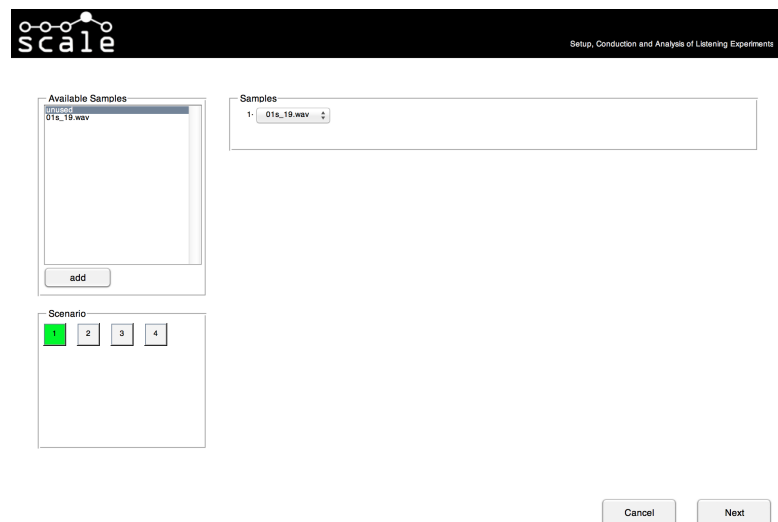


Figure 17: Setting of the .wav files for the BINAURAL SSR stimuli type

5.4.3 Order and number of stimuli

Adaptive - An adaptive test can have a maximal of 50 stimuli. The first stimuli on the first slot will correspond to the reference stimuli, the second will correspond to sample most similar to the reference and so on. The last slot will correspond to the sample with the biggest difference to the stimuli.

SAQI - If internal reference was selected, only one stimuli will be added for each trial. If external reference was selected, the first slot will contain the reference and the second the stimulus to be compared.

ABC-HR - In ABC-HR tests, each run compares two different stimuli. The slots will represent those pairs. Slot 1 and 2 will be the first pair, slot 3 and 4 the second, 5 and 6 the third and so on. The presentation of the pairs will be randomized differently for each subject, however, in the results analysis they will be presented in the order they were added.

ABX - In ABX tests, each run compares two different stimuli. The slots will represent those pairs. Slot 1 and 2 will be the first pair, slot 3 and 4 the second, 5 and 6 the third and so on. The presentation of the pairs will be randomized differently for each subject, however, in the results analysis they will be presented in the order they were added.

MUSHRA - In a MUSHRA test, a maximum of 15 stimuli can be presented in a single trial. The reference will be placed in the first slot and the rest of the samples in the remaining slots. The reference will be played when clicking on the reference button during the trial, but will also be presented in one of the sliders like the rest of the stimuli. The presentation of the stimuli will be randomized differently for each subject, however, in the results analysis they will be presented in the order they were added.

5.4.4 Finish

Once the stimuli have been added, the test configuration is finished and the user just needs to click on the button **next** on the window presented in Figure 18. If the user cancels the configuration of the test or closes *Scale* before this point, all the files created during the steps mentioned in this section will be deleted.

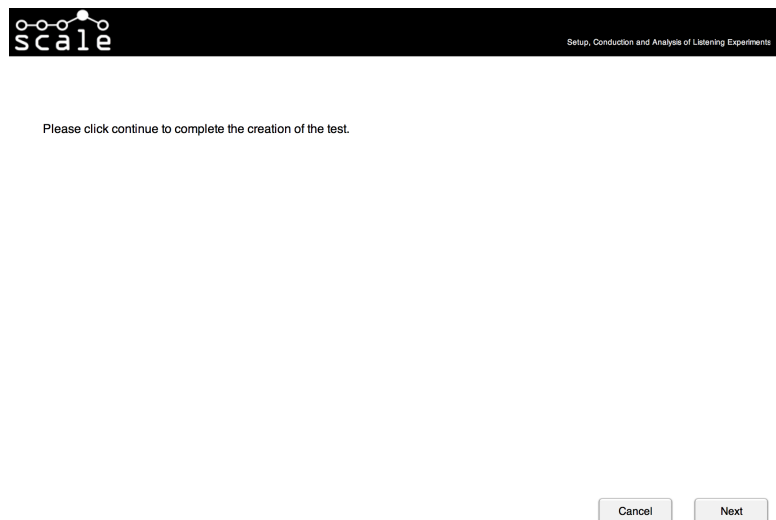


Figure 18: Completion of test creation

5.4.5 Files created

Once a test is configured, a folder with the name of the test is created inside *Scale/Scale_user_folder/Tests*. This folder contains the struct **test_info** which contains the information of the test, it also contains the *.wav* samples that will be played, the *.asd* files in case of *BINAURAL_SSR* stimuli type and a folder called *results* containing the files corresponding to the results. One result file is created each time a test is performed.

6 Running the program - Perform

In this section the process of performing a test is explained step by step. Specific information about how the stimuli are selected and presented and which are the tasks of the subjects will be given for every type of test.

6.1 Test preparation

The first window to appear is dedicated to prepare the system for the test. If you are using *.wav* stimuli type, nothing will be done at this point.

If you are using the *SSR*, Figure 19 will open. On this window, the processes of turning on the *SSR* and achieving a connection are done. The user can also click on the option **SSR Visible** if he wants to see the *SSR* user's interface during the test. Sometimes it is useful to leave it on to control that everything is working. In case no connection is not achieved, try several times before restarting the program. Please note that the *JACK* interface should be turned on before *Matlab* is started. In case it was not you will get an error message. The first time that *Scale* is used on a new system, this window will ask the sudo password, this is needed during the *SSR* starting process. The password will be saved and it will not be asked again.

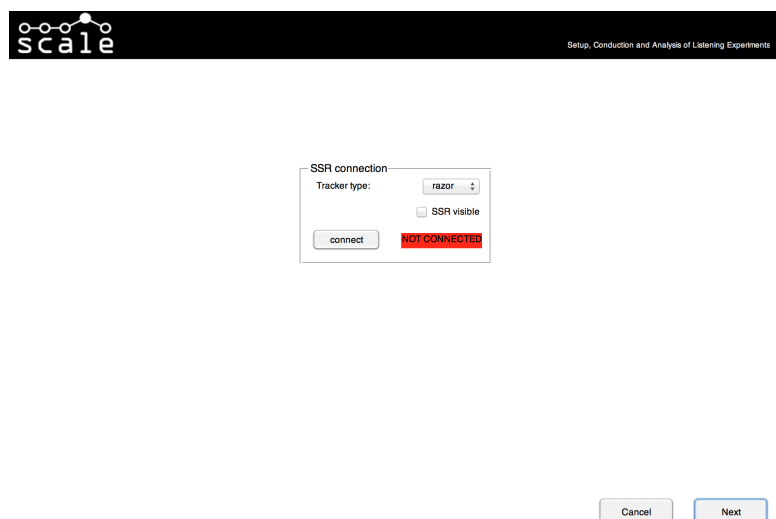


Figure 19: System preparation window

6.2 Subject login

In order to perform a test, the subject must be logged in, to log in, introduce the user data in the required fields on the window shown in Figure 20. If the data is correct, the subject will be logged in and the test can be started by pressing the button **next**. In case that the subject did already started the test, a new button will appear on the lower-left corner called **restart**. If **restart** is pressed, the results of that subject will be deleted and the test will be restarted. Please note that only a registered subject can perform a test. To register a subject, use the button **Register subject** on the main program window shown at the beginning of the manual (Figure 8).

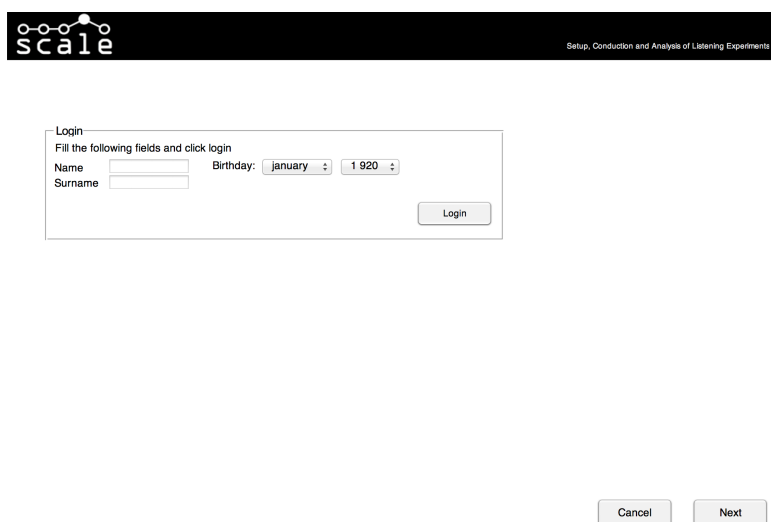


Figure 20: Subject login window

6.3 Test Introduction Window

Figure 21 shows the **Introduction** window of the test. As an example of how the introduced text look like, the text written in the field general instructions of Figure 10 is shown. In this window it would be useful a description of how the subject must interact with the **Scenario Selection** window, see Figure 22, due to the fact that the scenario order can be different from one test to another.

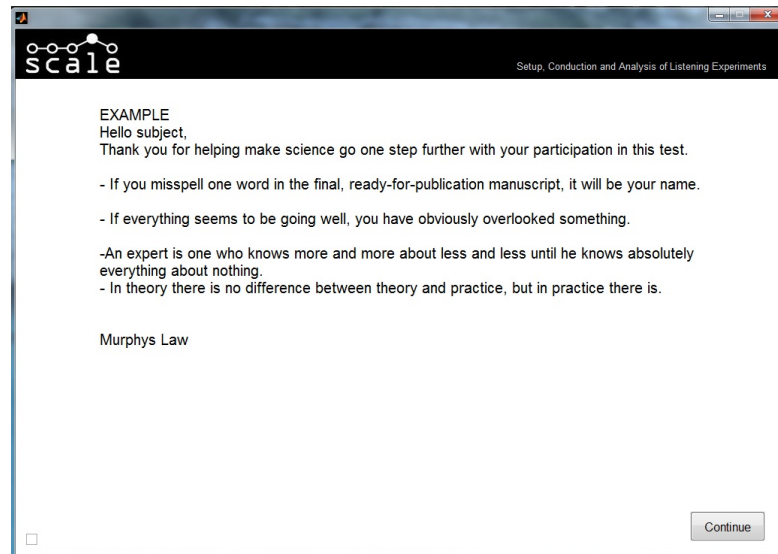


Figure 21: Test Introduction window

6.4 Scenario Selection Window

Figure 22 represents the **Scenario Selection** window. Subjects interact with this window at the beginning of the test and every time they complete a scenario.

In this example the test consists in six scenarios; tests with a different number of scenarios will make the window look different because a specific distribution of the buttons in the window is performed in every case. The text written in the window explains to the subject the meaning of the different buttons and what they have to do with them. To perform a scenario, the subject must click **next**. The scenario that he is going to perform is the one that has its button number painted in green. Buttons corresponding to completed scenarios are unenabled and the word "completed" is written in them. When all scenarios are completed, button **Next** will change its text to **Finish**. The selection of the next scenario is done automatically by *Scale*.

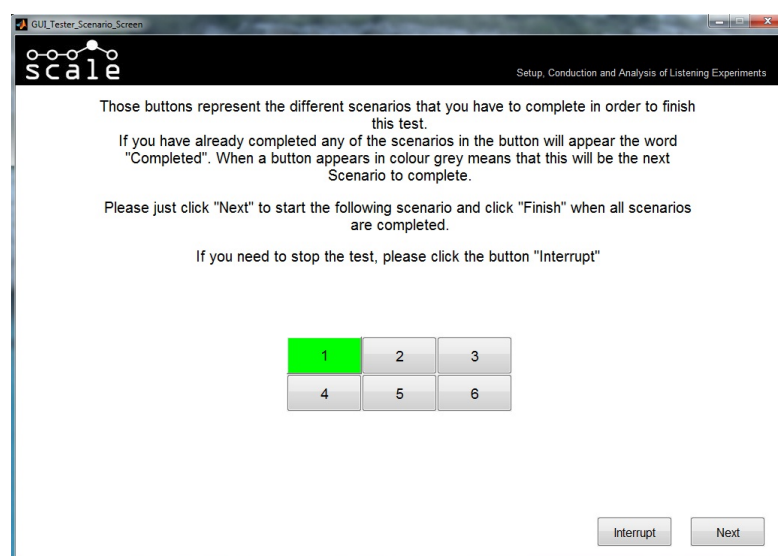


Figure 22: Scenario Selection window

The button **Interrupt** stops the test. The information of the results until this point will be saved and the test will be able to be continued later. It is only possible to interrupt a test in the pause between scenarios. If a test is interrupted during the performance of an scenario, the information will be saved only until the last completed scenario.

6.5 Trial Window

Each type of test uses a different trial window, following each procedure is individually explained together with screenshots.

6.5.1 Adaptive Methods

In the adaptive methods stimuli that differ in a certain parameter are presented. The subject is aimed to find that difference. The decision can vary depending on the adaptation method and on the paradigm; *Scale* offers three paradigms which are **Yes/No**, **Two Alternative Forced Choice (2AFC)** and **Three Alternative Forced Choice (3AFC)**. In a Yes/No paradigm the subjects are asked directly if they could find that mentioned difference after hearing a single interval. In alternative forced choice paradigms they must find the difference indirectly comparing two or more intervals. In all cases, when the subject give a positive answer, next run should present a situation where the difference is lesser. This process should be performed until the point where the subject reaches the border between noticeable and unnoticeable. This point corresponds to the psychophysical threshold. Depending on the adaption method, the stimuli used and other specific settings, more or less runs will be needed to reach that threshold. Figure 23 presents an example of the *Yes/No* and *3-AFC* paradigms.



Figure 23: Adaptive Method Run windows

The sample to play in each run will depend on the answer given and on the adaptation method. Following, a list explaining the different adaption methods is given. For a better understanding it will be assumed that the last presented stimuli was the one corresponding to sample n .

- **oneUp1Down** - The first sample to be compared to the reference will be sample max , that means the last sample of the samples vector. Reference will be always sample 1. If the answer is correct, next sample will be $n-1$. If the answer is wrong next sample will be sample $n-1$.

- **oneUp2Down** - The first sample to be compared to the reference will be sample *max*, that means the last sample of the samples vector. Reference will be always sample 1. If both current and previous answer are correct, next sample will be $n-1$. If current answer is correct but previous answer is wrong, next sample will be n . If current answer is wrong next sample will be $n+1$.
- **twoUp1Down** - The first sample to be compared to the reference will be sample *max*, that means the last sample of the samples vector. Reference will be always sample 1. If current and previous answer are wrong, next sample will be $n+1$. If current answer is wrong but previous answer is correct, next sample will be n . If current answer is correct next sample will be $n-1$.
- **QUEST** - Reference will be sample 1. The samples to play will be calculated using the *QUEST* algorithm which varies in function of the parameters given by the user, explained in [4].

Every time the subject gives an answer (*positive/correct or negative/wrong*) that is different from the previous, a **reversal** occur. For a non *QUEST* test, there are three ways to finish a scenario and two of them can be set by the researcher in the **ending criterion** options:

- **Number of runs** - The scenario will end after a certain number of runs.
- **Number of reversals** - The scenario will end after a certain number of reversals occur.
- **Sample index above the limits** - The scenario will end when the samples having indexes 0 or $n+1$ are required. This means that the subject has not only achieved the maximum or the minimum, but surpassed it. This would mean that the threshold is out of range.

6.5.2 ABC-HR method

ABC-HR follow the "double blind principle". Those tests are aimed to compare two samples: the original sample, that works as a reference, and a modified sample. To make this comparison, the subject hears three intervals (*A*, *B* and *C*). Interval *A* contains always the reference. Intervals *B* and *C* contain one the reference and the other one the modified. Nor the researcher or the subject will know which one is each. This assignation is always randomly made for every run.

Subject hears first interval *A* followed by interval *B* and then interval *A* followed by interval *C*. After hearing them for the first time, he can listen them again an unlimited number of times just clicking buttons **A**, **B** or **C**. After hearing the intervals, the two questions must be answered. First question should be "How would you evaluate the difference between samples *A* and *B*?" and second question, "How would you evaluate the difference between samples *A* and *C*?". The answer is performed positioning the slider thumb in a certain place in the horizontal scroll. After the user has clicked at least once in every slider the button *Continue* will set to enable and once pressed the run end. The recommendation *ITU-R BS.1116 for the evaluation of small impairment in coded*

Figure 24: ABC-HR Method Run window

audio signals [6] uses this method under certain conditions. Figure 24 shows an example of a run in an ABC-HR test using the mentioned recommendation.

6.5.3 ABX method

The ABX test is a simplification of the *ABC-HR* test. In a trial of an *ABX* test, a subject is presented with two intervals, which are A and B, followed by a third one called X. After hearing A, B and X, the subject must select which of the stimuli in A and B resembles to X and click on it. In this case, as opposite to the *ABC-HR*, no rating is done. In *Scale*, each scenario can contain many runs and for each run, the position of the reference stimuli in A and B is randomized. Figure 3 shows a trial window of the test.

Figure 25: ABX Method Run window

6.5.4 MUSHRA Method

This procedure is aimed to compare multiple samples with medium or big impairments between them. The advantage of this method is that it enables to compare multiple samples at the same time having also a reference and an anchor. The reference is presented two times, one as a known reference and the other as a hidden reference. The anchor must be also hidden. The rating is performed with vertical sliders placed on the top of every stimuli button.

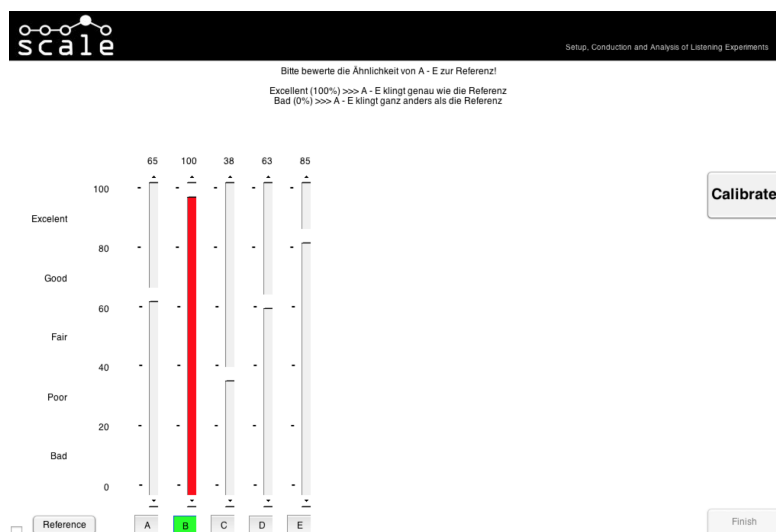


Figure 26: MUSHRA Method Run Window

Figure 26 shows a run window of a **MUSHRA** test, the buttons containing the letters A to O represent the different stimuli and the button *Reference* the reference stimuli. When the subject clicks on one of the buttons, the stimuli contained in that letter will be played. At the right side of the screen there are some options related to the playback such as *Pause*, *Resume*, *Stop* and *Loop*. Also a slider shows the playback instant. On the left side of the window the rating scale is presented and also the rating grades. All sliders have anchors on their side every ten points and at the top a text field show the position of the slider thumb. After the user has clicked at least once in every slider the button *Continue* will set to enable and once pressed the run end.

6.5.5 SAQI method

The spatial audio quality inventory (SAQI) [8] test has been specifically designed for the perceptual evaluation of virtual acoustic environments. In every trial a reference and a stimulus are presented together with 48 verbal descriptors of perceptual qualities that are assumed to be of practical relevance when comparing virtual auditory environments. Each descriptor comprises a rating scale with a pair of opposed adjectives in its scale ends. The task of the subject is to compare the stimulus to a given or inner reference and give a rating for each perceptual quality.

Figure 27 shows a run window of a **SAQI** test. The test window may differ depending on which qualities were selected by the researcher for evaluation. On the upper-right

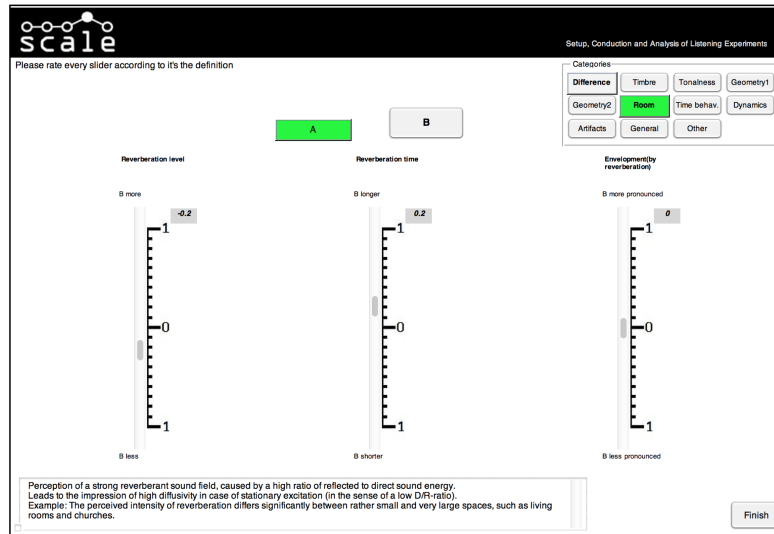


Figure 27: SAQI Method Run window

corner, the panel **Categories** contains a button for each quality category. When pressing on a button, the test window will show all the sliders corresponding to the qualities of the selected category. When the test starts, the category **Difference**, which contains a single slider, is shown. At this point, the other buttons are blocked. This is made on purpose because if no difference is found, all the qualities will be automatically set to 0 and the run will be finished by pressing the button **Finish** (bottom-right corner). In case the slider of the Difference quality is moved, the rest of the Category panel buttons will become active.

When clicking on a slider, a description of the quality corresponding to it will be shown at the bottom of the window. In order to switch from reference to tested sample click on the buttons **A** or **B**. The stimuli are looped, so the subject hears the sound continuously, if the sound needs to be stopped, this is done by clicking on the active sample button **A** or **B**, in green. If the test uses an internal reference, then only one button is present.

7 Running the program - Analyse

Results obtained in tests are analysed at the researcher's discretion. Hence, the data can be represented in many different ways. In *Scale* some of the most common methods of representation have been implemented in order to facilitate this process. The analysis module offers a graphical representation of the collected data and results can be exported to MATLAB Structs (*.mat*) and MATLAB Figures (*.fig*).

7.1 Analyse Window - Overview

The analyse of the results is made through the window represented in Figure 28. This window can vary on the test procedure, but most of the settings are common for all of them.



Figure 28: Example of the analyse window

The common options, Figure 29, include the **Results** panel and the **Scenario** panel. In the results panel, the available results sets from a certain test are presented as a list. To select a result set just click on it and then on the + button, you can add more results by repeating the process or select **all** of them by clicking **select all**. To not include a single result set, select it on the **Selected** panel and click the - button, to deselect all result sets click on **Clean all**.

In the panel **Scenario** the scenarios to be seen can be selected. Each scenario has a color assigned and this color will correspond to the color of the line representing the data of it in the graphic.

The panel **Save Data** contains two buttons. If the user click on **plot** the graphic can be saved as a *Matlab* figure, if **all data** is clicked, the results will be exported in a *Matlab* struct file which can be used outside *Scale*.

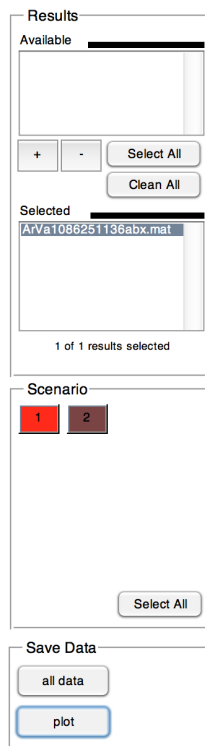


Figure 29: Analyse window common options

7.2 Analyse Window - Adaptive, ABC-HR, ABX and MUHSRA

The window to observe the results is common for the Adaptive, ABC-HR, ABX and MUHSRA procedures, see Figure 30. In the panel **options** three options can be selected.

Single - For every subject and for every scenario a line will be drawn with his obtained value/s.

Average - The values obtained on the all the result sets are averaged and a single value/s are presented.

Combined - Both information shown in **Single** and **Average** options is plotted.

7.3 Analyse Window - SAQI

The **options** panel include several buttons, each one corresponding to a category of qualities, and a list of the visualization possibilities (single, average and combined), see Figure 31. When observing the categories, please take in account that only 4 categories can be plotted simultaneously.

Single - For every subject and for every scenario a line will be drawn with his obtained value/s.

Average - the values obtained on the all the results sets are averaged and a single value/s are presented.

Combined - Both information shown in **Single** and **Average** options is plotted.



Figure 30: Analyse window for Adaptive, ABC-HR, ABX and MUHSRA procedures

7.4 Exported Data

In this section the exportation of the data will be explained. The results data is exported in a single struct called `data_subs`. This struct has several fields which some of them are common for all type of tests. The common data will be now explained and the test specific results will be later explained specifically for each test type.

7.4.1 Common fields

Figure 32 shows all the fields of the results struct. The fields inside the red frame, are common for all type of tests, each field is a cell array. The cell arrays have as many positions as result sets. In the case of the figure, there are two sets of results, for this reason, the length of the cell arrays is 2. The not common part of the results will be saved in the fields `scen<number of scene>`. The contain or the data class of this `scene` structs will be following explained separately for each test.

7.4.2 Adaptive

For the adaptive tests, every trial includes two different values. The first one is the threshold value `Scen<trial_number>_tValue` which in this case is the value of the last stimuli played and then a vector with all the stimuli played in the trial saved in `Scen<trial_number>_runValues`. This vector has a fixed length of 100, and if the trial was performed in less than 100 runs, the extra positions will be filled with `NaN`. In the figure, the struct contains only one set of results, if it contained more sets of results, `Scen<trial_number>_tValue` and `Scen<trial_number>_runValues` would be a vector and a vector of vectors with a length equal to the number of results. This is shown in Figure 33.

**If the staircase method QUEST is used, two more fields are added which are the `Scen<trial_number>_st_dev` containing the standard deviation in each trial and the `Scen<trial_number>_QuestStruct` containing the QUEST information struct in each trial. This is shown in Figure 34.*

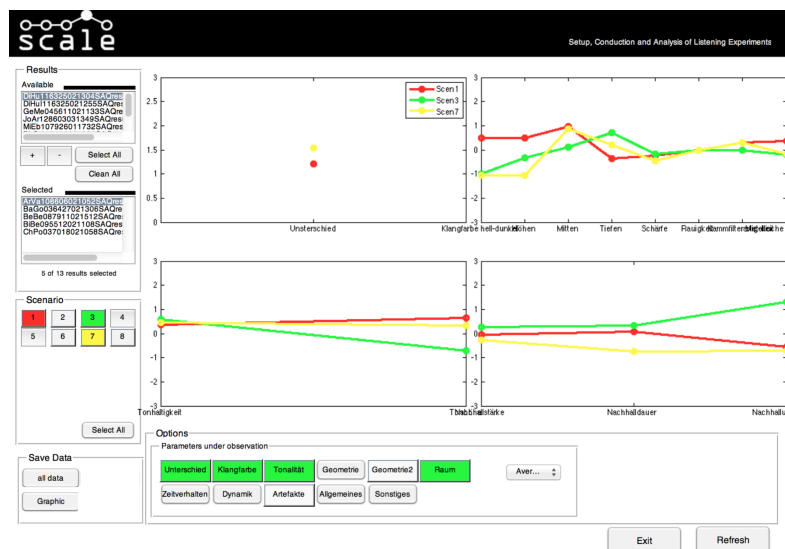


Figure 31: Analyse window common options for SAQI procedure

Variable Editor – data_subs

Stack: Base Select data to plot

Field	Value	Min	Max
subjectID	<2x1 cell>		
resultsCode	<2x1 cell>		
Surname	<2x1 cell>		
Name	<2x1 cell>		
Gender	<2x1 cell>		
Country	<2x1 cell>		
Experience	<2x1 cell>		
Audio_Knowledge	<2x1 cell>		
Hearing_Problems	<2x1 cell>		
Scen1	<2x7 double>	NaN	NaN
Scen2	<2x8 double>	NaN	NaN

Figure 32: Exported results struct

Audio_Knowledge	<1x1 cell>		
Hearing_Problems	<1x1 cell>		
Scen1_tValue	11	11	
Scen2_tValue	10	10	
Scen1_runValues	<1x100 double>	NaN	NaN
Scen2_runValues	<1x100 double>	NaN	NaN

Figure 33: Exported results struct

Scen1_tValue	0	0	0
Scen2_tValue	10	10	10
Scen1_runValues	<1x100 double>	NaN	NaN
Scen2_runValues	<1x100 double>	NaN	NaN
Scen1_stDevValue	0	0	0
Scen2_stDevValue	0.7921	0.7921	0.7921
Scen1_QuestStruct	<1x1 struct>		
Scen2_QuestStruct	<1x1 struct>		

Figure 34: Exported results struct

7.4.3 ABC-HR

In an *ABC-HR* test every trial includes a single vector of values called *Scen<trial_number>*. Those values correspond each to the comparison between a pair of stimuli. For example, the first position of the vector *Scen1* will show the comparison between stimuli added in slot 1 and stimuli in slot 2. The second position, will show the comparison between stimuli in slot 3 and slot 4 and so on. Please note that during the test, the presentation of the stimuli pairs is randomized but here is ordered again. The value itself can have a range from *-nValues* to *+nValues*. The negativity or positivity of the value depends on the answer of the subject. For example if C was the same as A and the subject rated "very different" to the question asking the similarity between A and C, the value would be equal to *-nValues*.

7.4.4 ABX

In an *ABX* test every trial includes a single vector of values called *Scen<trial_number>*. Those values correspond each to the comparison between a pair of stimuli. For example, the first position of the vector *Scen1* will show the comparison between stimuli added in slot 1 and stimuli in slot 2. The second position, will show the comparison between stimuli in slot 3 and slot 4 and so on. Please note that during the test, the presentation of the stimuli pairs is randomized but here is ordered again. The value itself can be 1 or 0. If it is 1 it means that the subject guessed correctly (the reference was pointed correctly to A or B), if it is 0 it means that the subject guessed wrong.

7.4.5 MUSHRA

In a *MUSHRA* test every trial includes a single vector of values called *Scen<trial_number>*. The vector has as many positions as stimuli presented in the trial. The order of the ratings is not related to the order that the stimuli were presented in the trial window because they are randomised for every subject. The order of the results corresponds to the order that the stimuli were added.

7.4.6 SAQI

In a *SAQI* test every trial includes a single vector of values called *Scen<trial_number>*. The vector has 48 positions and the rating for each quality is written in this vector. Since the user can select qualities to not be asked, the positions of the not asked qualities will not be taken in account. Figure 35 show the correspondence between the vector positions and the quality names.

-
- | | |
|-----------------------------|----------------------------------|
| 1 Difference | 25 Envelopment(by reverberation) |
| 2 Tone color bright-dark | 26 Pre-echoes |
| 3 High-frequency tone color | 27 Post-echoes |
| 4 Mid-frequency tone color | 28 Temporal disintegration |
| 5 Low-frequency tone color | 29 Crispness |
| 6 Sharpness | 30 Speed |
| 7 Roughness | 31 Sequence of events |
| 8 Comb filter coloration | 32 Responsiveness |
| 9 Metallic tone color | 33 Loudness |
| 10 Tonalness | 34 Dynamic range |
| 11 Pitch | 35 Dynamic compression effects |
| 12 Doppler effect | 36 Pitched artifact |
| 13 Horizontal direction | 37 Impulsive artifact |
| 14 Vertical direction | 38 Noise-like Artifact |
| 15 Front-back position | 39 Alien source |
| 16 Distance | 40 Ghost source |
| 17 Depth | 41 Distortion |
| 18 Width | 42 Tactile vibration |
| 19 Height | 43 Clarity |
| 20 Externalisation | 44 Speech intelligibility |
| 21 Localizability | 45 Naturalness |
| 22 Spatial disintegration | 46 Presence |
| 23 Reverberation level | 47 Degree-of-liking |
| 24 Reverberation time | 48 Other |

Figure 35: List of SAQI qualities

8 Closing Scale

The correct way to close *Scale* is by clicking **Exit** button in the **Main program** window. The **Main program** window is accessed when a task is completed, for example when a test is created, or when some actions are cancelled, for example when a test is interrupted. In order to avoid errors and to provide a good functionality the **close** button in the upper-right corner of the windows is under normal conditions deactivated; it would be a problem for example when a subject accidentally closed a test. For this reason a little "trick" has been implemented. In the bottom-left corner there is a little input text field. When the number 22 is written in this field and the close button is pressed the program can be closed.

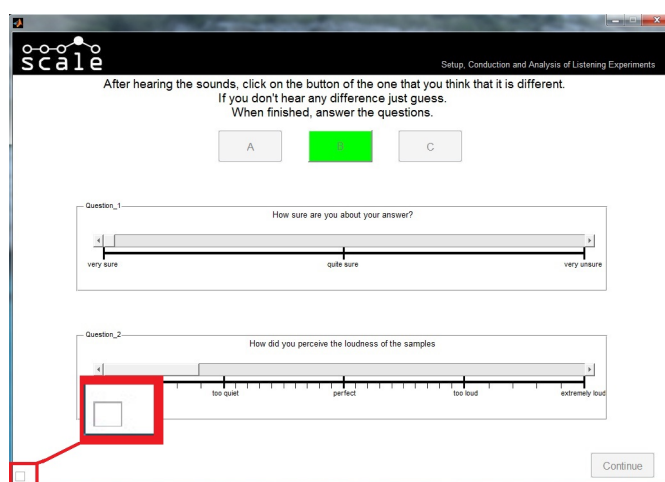


Figure 36: Closing input text field

9 The Sound Scape Renderer

9.1 Introduction

The *SoundScape Renderer (SSR)* [9] is a tool for realtime spatial audio reproduction, which includes several rendering methods. It is free open source software running on *UNIX*-based systems and using the *JACK* [10] audio framework. The *SSR* can be controlled either using a graphical user interface (GUI) or a TCP/IP network interface. Sound source positions and other attributes represented in a spatial audio scene, as shown in Figure 37, can be imported or exported using audio scene description files (.asd) [11].

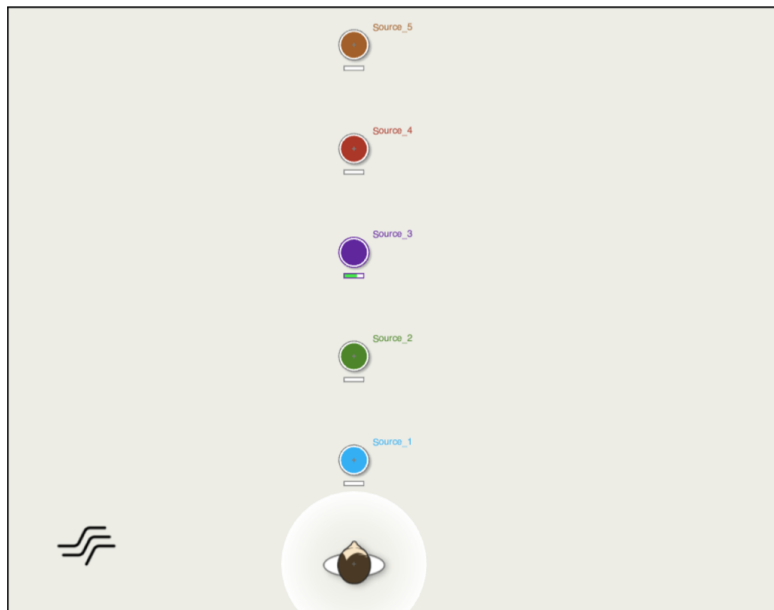


Figure 37: SSR user interface

9.2 Sound Scape renderer and Scale

As shown in Figure 38, *Scale* processes subject's inputs and operates the *SSR* sending *XML* messages through its network interface via *TCP/IP* protocol while the test is performed. The *SSR* runs in the background generating stimuli with the combination of the incoming audio signal, the tracker data and the corresponding head related impulse response (HRIR) or binaural room impulse response (BRIR) set. The extension *Scale-SSR* uses the BRIR mode because it allows the use of arbitrary impulse response datasets in the same scene.

9.3 Installation

The *SSR* needs to be installed in the system in order to be used. Please follow the next steps to complete the installation:

1. Click the link user manual on the site <http://www.spatialaudio.net/>.

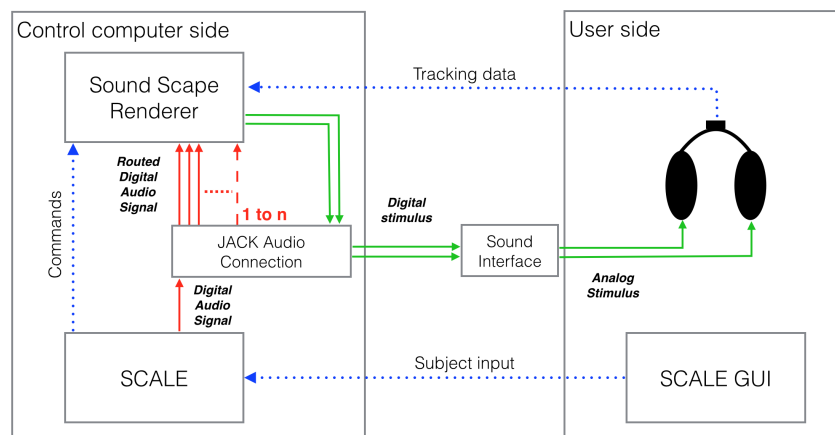


Figure 38: SSR System architecture

2. In the user manual there are the instructions about downloading and installing the SSR.
3. The Scale-SSR extension has been developed using the version 0.4.2 of the SSR. Please note that while older versions should not have any problem, earlier versions could.

10 Jack Audio Connection Kit

The SSR uses the Jack Audio Connection Kit to make the routing of the sound and for this reason it needs to be installed on the system. Remember that the Jack audio connection must be running before you start Matlab. If it is not the case, it will not be detected. To install the software go to the site <http://www.jackaudio.org/downloads/> and download the Jack1 for OSX version 0.124.1. or greater.

References

- [1] A. Vazquez, “Scale, a software tool for listening experiments,” *Proceedings of the DAGA*, 2013.
- [2] A. Vazquez, “Scale, conducting psychoacoustic experiments with dynamic binaural synthesis,” *Proceedings of the DAGA*, 2015.
- [3] H. Levitt, *The City University of New York*. PhD thesis, University of Wollongong, 1970.
- [4] A. Watson and D. Pelli, “Quest: a bayesian adaptive psychometric method,” *Perceptive Psychophysics*, vol. 33(2), pp. 113–120, 1983.
- [5] E. Zwicker and H. Fastl, *Psychoacoustics. Facts and Models*. Springer, 2nd ed., 1999.
- [6] I.T.U., “Methods for the subjective assessment of small impairments in audio systems including multichannel sound systems,” *ITU-R BS*, no. 1116-1, 1997.
- [7] I.T.U., “Method for the subjective assessment of intermediate quality level of coding systems,” *ITU-R*, vol. BS, pp. 1534–2, 2014.
- [8] A. Lindau, V. Lepa, *et al.*, “A spatial audio quality inventory (saqi),” *Acta Acustica united with Acustica*, vol. 100, pp. 984–994, 2014.
- [9] M. Geier, J. Ahrens, and J. Spors, “The soundscape renderer: A unified spatial audio reproduction framework for arbitrary rendering methods,” *Proceedings of the 124th Convention of the AES*, no. 7330, 2008.
- [10] P. Davis *et al.*, “Jack audio connection kit.” <http://jackaudio.org/>, October 2013. Version 1.9.
- [11] M. Geier, J. Ahrens, and S. Spors, “Asdf: ein xml format zur beschreibung von virtuellen 3d-audioszenen,” *Proceedings of the DAGA*, 2008.