APEX 3 User Manual

Astrid van Wieringen Lot Van Deun Tom Francart

June 2, 2016

Warning

This is a preliminary version of the APEX 3 user manual. It is still incomplete and bound to contain errors and inaccuracies. A message will be sent to the apexannounce mailing list when a full version is available.

Contents

Chapter 1

Introduction

In order to be able to perform auditory psychophysical and speech perception experiments a versatile research platform has been developed at ExpORL, Dept. Neurosciences, KULeuven (Laneau et al., 2005; Francart et al., 2008). APEX 3[1], allows most auditory behavioral experiments to be performed without any programming, both for acoustic stimulation, direct electric stimulation via a CI or any combination of devices.

This manual describes APEX 3. The idea behind APEX 3 is that one should be able to set up an experiment without any programming knowledge. It is a generic platform with abstract interfaces to the computer monitor, computer input devices such as keyboard, mouse, and touch screen, and output devices such as sound cards or interfaces to cochlear implants. The user should be able to use any of the interfaces without programming any device specific details.

Experiments are defined in the XML format[2], allowing for a structured experiment definition in a generic format. A Matlab and R toolbox is distributed together with APEX 3 to ease the automatic generation of experiment files and analysis of results files.

1.1 Registration

After registration APEX 3 can be downloaded from http://www.kuleuven.be/exporl/apex and can be used free of charge. The hardware requirements are limited to a personal computer running the Linux or Windows operating system and the necessary stimulation devices.

1.2 Citing

For proper citation see License Agreement at http://www.kuleuven.be/exporl/apex. Any publication that deals with APEX 3 should cite: Francart, T., van Wieringen, A., Wouters, J. APEX3: a Multi-purpose test platform for auditory psychophysical experiments. Journal of Neuroscience Methods, vol. 172, no. 2, pp. 283–93, 2008.

[1] Application for Psychophysical EXperiments

[2] The complete XML specification can be found at http://www.w3.org/TR/xml11/

When using the animated child mode, you should also refer to: Laneau, J., Boets, B., Moonen, M., van Wieringen, A. and Wouters, J., "A flexible auditory research platform using acoustic or electric stimuli for adults and young children", Journal of Neuroscience Methods, 142, pp. 131-136, 2005.

1.3 Main features of APEX 3

- No programming is required to set up an experiment.
- Multiple platforms are supported, including MS Windows and Linux.
- Multiple output devices are supported, including sound cards, an interface to cochlear implants from Cochlear and an interface to cochlear implants from Advanced Bionics. The supported devices can be used in any combination, a CI (or hearing aid) in both ears (bilateral electrical stimulation) or simultaneous stimulation via a CI in one ear and acoustical stimulation in the other (bimodal stimulation).
- Several psychophysical procedures are readily available and custom procedures can easily be added (plug-in procedure).
- As much information as possible is stored per trial in the result file. This includes the subject's response, but also response times, calibration values and much more.
- Visual feedback can be given after each trial.
- There is a special animated interface for testing (young) children.
- There is a Matlab toolbox for experiment file creation and advanced result file analysis.
- Custom signal processing filters can be added. (plug-in filter)
- Custom interfaces to external controllers can be added. (plug-in controller)
- There is a GUI (Graphical User Interface) to calibrate parameters.

1.4 Getting help - documentation

There are 4 forms of documentation available for APEX 3:

The paper The APEX 3 paper gives a concise high level overview. It is advisable to read it first. Note that since it was published some implementation details have changed, so please refer to the other documentation for up to date details.

The manual You are currently reading the manual. It does not cover every feature of APEX in detail, but endeavours to describe the most often used features in more detail than the paper.

The schema documentation The APEX schema defines the structure of an experiment file (see section 2.3.2). It systematically contains documentation for each element, and exhaustively determines which elements can occur. It is the most up to date and most complete source of documentation, but requires some insight in the general function of APEX, which is provided by the paper and manual. The schema documentation can be consulted in 3 different ways: (1) using the HTML format schema documentation that comes with APEX, in folder `documentation` , (2) when editing an experiment file with the oXygen editor, it will show the documentation for the current element in a different pane and suggest elements when you start typing, or (3) opening the schema `experiment.xsd` directly in the oXygen editor.

check

The examples APEX is shipped with a large number of examples, illustrating most of the features. They are stored in folder `examples` in the main APEX folder, and are documented in chapter **??**. The best way to start a new experiment, is probably to take one of the examples and modify it until it suits your needs.

1.5 Style conventions of the manual

In this manual different style conventions are used:

- <**xml**/> example fragment of XML
- Information specific to the use of the program OxygenXML is formatted as follows:

  OxygenXML is an XML editor

- a different font is used `to indicate file names`

1.6 Basic setup

APEX 3 is installed by either running the installer (`apex.msi`), or by simply copying the APEX folder to your computer. After installation, there is a main APEX 3 directory (default: `c:/program files/apex3`) under which the following subdirectories exist and contain the necessary files:

bin Binary files: the main APEX 3 executable (`apex.exe`), the Qt dll's and some Qt plugins

schemas The experiment file schema (`experiment.xsd`) and the apexconfig (`apexconfig.xsd`) schema. You can point your XML editor to the former schema when editing experiment files (section 2.3.2).

config The `apexconfig.xml` file contains general APEX 3 settings that are applied to all experiments.

plugins contains different plugins (cf. appendix C.1).

check

amt contains the APEX Matlab Toolbox (AMT), for automatic generation or analysis of experiment files (cf. appendix B).

check folder name

examples contains example experiments for nearly every feature of APEX 3

APEX 3 makes use of an experiment file. An experiment file contains all the necessary information to run an experiment, such as the layout of the screen, the workings of the procedure, references to stimulus files and pictures, the way in which the stimuli are routed to a device etc. Several tools exist to create experiment files easily and analyze results, e.g. an XML editor (next section) or the AMT.

1.7 XML editor: to create or edit experiment files

While any text editor, such as wordpad, notepad and many others, can be used to create or edit experiment files, the use of an editor specifically suited for editing XML files has many advantages, such as syntax highlighting, automatic completion and validation.

At ExpORL we use the OxygenXML editor and will therefore give hints on how to use the syntax in this manual. You are, of course, free to use any other editor.

A free demo version of OxygenXML is available from `http://oxygenxml.com`. If you would decide to buy the full version, you will get a 10% discount if you enter the following coupon code during the ordering process:

oXygen-Kuleuven

A free version is available for people working in the field of life sciences.

1.8 Overview of the APEX 3 user manual

Chapter 2 discusses the basic concepts of APEX 3. Chapter 3 describes the structure of experiment files. Chapter C describes how to extend APEX 3 using plugins. Chapter 5 discusses 4 example experiments in detail. These experiments are also stored in the folder `examples/manual`. Chapter 6 shows how to implement some common features of psychophysical and perceptual experiments. Chapter 7 deals with how APEX 3 result files can be analyzed. Finally, four appendices are included, to describe the use of the L34 device (appendix A), the Matlab toolbox (appendix B), the use of plugins (appendix C.1), and customizing appearance (appendix D).

An exhaustive list of all elements that can occur in an APEX 3 experiment file are given in a the schema documentation.

Chapter 2

Basic concepts

2.1 Introduction

An APEX 3 experiment is defined in an experiment file. Experiment files are defined in the XML format (see section 2.3), and it is advised, but not compulsory, to use OxygenXML to edit the XML format (see section 1.7).

While experiment files can have any filename and any extension, it is advised to give your experiment file the extension `.apx`, such that is immediately associated with APEX 3. Also, there is a clear distinction between *experiment* files and *result* files, which will receive the extension `.apr`.

> When opening a file from OxygenXML, it may not recognize the experiment file with extension `.apx` immediately. If so, select "All files" under "Files of type", and select the desired experimental file.

2.2 Terminology

APEX 3 is based on a few basic concepts that are common to all psychophysical experiments. Here we define the concepts device, controller, datablock, stimulus, filter, screen, response, trial, procedure, experiment, result, ID, and parameter.

device is a system connected to the computer that can be controlled by APEX 3. Devices can send signals to a transducer. Examples are sound cards and interfaces to cochlear implants (CIs). Devices can have parameters that are controlled by APEX 3.

controller is a system connected to the computer that -unlike a filter- does not accept/transfer signals (e.g., gain), but has parameters that are controlled by APEX 3. An example is a programmable attenuator that controls the gain of an amplifier.

datablock is an abstraction of a basic block of data that can be processed by APEX 3 and can be sent to a matching device. In the case of a sound card, the datablock would be an audio signal in the form of a series of samples that is commonly stored on disk as a "wave file".

stimulus is a unit of stimulation that can be presented to the subject, to which the subject has to respond. In the simplest case it consists of a single datablock that can be sent to a single device. More generally it can consist of any combination of datablocks that can be sent to any number of devices, simultaneously or sequentially.

filter is a data processor that runs inside APEX 3 and that accepts a block of data, e.g., a certain number of samples from an audio file, and returns a processed version of the block of data. An example is an amplifier that multiplies each sample of the given block of data by a certain value.

screen is a GUI (Graphical User Interface) that is used by the subject to respond to the stimulus that was presented.

response is the response of the test subject. It could, for example, be the button that was clicked or the text that was entered via the screen.

trial is a combination of a screen that is shown to the subject, a stimulus that is presented via devices and a response of the subject.

procedure The procedure controls the flow of an experiment. It decides on the next screen to be shown and the next stimulus to be presented. Generally, a procedure will make use of a list of predefined trials.

experiment file : a combination of procedures and the definitions of all objects that are necessary to conduct those procedures.

result is associated with an experiment and contains information on every trial that occurred.

ID is a name given to an object defined in an experiment. It is unique for an experiment. If, for example, a device is defined, it is given an ID by which it can be referred to elsewhere in the experiment.

parameter is a property of an object (e.g. a device or filter) that is given an ID. A filter that amplifies a signal could for example have a parameter with ID *gain* that is the gain of the amplifier in dB. The value of a parameter can be either a number or text.

standard In a multiple alternatives forced choice procedure the reference signal is defined as the standard. Internally, APEX 3 does not differentiate between a stimulus and a standard. See example 2: the standard is defined under `<stimuli>` and should be referred to by its ID.

2.3 XML

2.3.1 Basic XML: elements, tags, attributes

Advantages of the XML format are that it is human readable, i.e., it can be viewed and interpreted using any text editor, and that it can easily be parsed by existing and freely available parsers[1]. Moreover, many tools exist for editing, transforming or otherwise processing XML files. In addition, a good XML editor can provide suggestions and documentation while constructing an experiment file. An XML file consists of a series of *elements*. Elements are started by their name surrounded by < and >, the begin tag, and ended by their name surrounded by </ and >, the end tag.

```
1    <b>1</b>
```

Every element can have content. There are two types of content: *simple* content, for example a string or a number, and *complex* content: other elements. An element can also have attributes: extra properties of the element that can be set. In the following example, element <a> is started on line 1 and ended on line 7. Element <a> contains *complex* content: the elements <b> and <c>. Element <b> contains *simple* content: the numerical value 1. Element <c> again contains *complex* content: the elements <c1> and <c2>. Element <c1> has an attribute named attrib1 with value 15. Element <c2> on line 5 shows the special syntax for specifying an empty element. This is equivalent to <c2></c2>. The reference manual lists all possible elements and attributes.

```
1 <a>
2    <b>1</b>
3    <c>
4       <c1 attrib1="15"> </c1>
5       <c2/>
6    </c>
7 </a>
```

The begin tag and end tag are case sensitive. A comment is inserted as follows:

```
1 <!--
2 This is a comment.
3  -->
```

If you start typing a name of an element in OxygenXML, several options will be offered. Choose one, press “Enter” and add the content.

A document that complies with the XML specifications is “well-formed”. A document is checked for well-formedness before any further processing is done.

[1] APEX 3 uses the Xerces-c parser for parsing XML files. http://xerces.apache.org/xerces-c/

2.3.2 Schemas

A schema describes the structure of the document and specifies where each element is allowed to occur. **In addition it contains exhaustive documentation**. The APEX 3 schema's are located under `/schemas` in the main APEX 3 directory.

A document is called *valid* when it adheres to the associated schema and if the document complies with the constraints expressed in it [2]. If the document is valid the experiment file should run directly after having opened it with APEX 3 (although errors that are specific to APEX 3 can still occur). APEX 3 will generate warnings and errors if the experiment file is not valid [3].

> For autocompletion and display of documentation it is necessary to work in OxygenXML. You need to associate the experiment file with the main schema, i.e. experiment.xsd. Click on "associate schema" in Oxygen (pushpin icon) and browse for experiment.xsd, which is usually located under `schemas` in the main APEX 3 directory[4]. Click on OK. Once an XML-file is associated with the schema, you can check whether your document is valid (menu bar: blue "v" icon, bottom row, second from left) and well-formed (menu bar: red "v" icon, bottom row, left).

Try the following:

- open the experiment file `examples/closedsetword.apx` using APEX 3. Run the experiment if you wish.
- open with OxygenXML
- associate the APEX 3 schema
- check validity
- make the file NOT well-formed, e.g. by removing an end tag. Check validity again.
- fix it again
- make the file invalid, e.g. by changing the name of an element
- run with APEX 3. Read error messages.
- undo the errors
- check well-formedness
- check validity
- Start typing < somewhere in the experiment file, and observe how OxygenXML suggests the different options

[2] a document can only be valid if it is well-formed

[3] by means of message window

[4] The APEX 3 schema can be found in the APEX 3 directory structure under `schemas/experiment.xsd`

2.4 Shortcuts

The following shortcuts are available:

F4 Stop experiment

F5 Start experiment

F6 Pause experiment

F7 Skip intro/outro movie

F9 Repeat last trial

Additional shortcuts for screen elements can be defined in the experiment file.

Chapter 3

Creating experiment files

An APEX 3 experiment file is an XML file. Like any XML file, it contains a root element (<apex>), which contains various main elements. The general structure of an APEX 3 experiment file could be as follows:

```
1  <?xml version='1.0' encoding='UTF-8'?>
2  <apex>
3    <procedure xsi:type="apex:adaptiveProcedure">
4      <parameters> ... </parameters>
5      <trials> ... </trials>
6    </procedure>
7
8    <screens> ... </screens>
9    <datablocks> ... </datablocks>
10   <devices> ... </devices>
11   <stimuli> ... </stimuli>
12 </apex>
```

The main elements are listed in the next section. Subsequently, each main element is discussed briefly.

3.1 Main APEX elements

Here we describe the main elements of the experiment, and the order in which they should occur. Some of them are compulsory, some are optional, and some contain child elements (nested, see section 2.3.1). Only the first child element is given. A complete listing of all the elements is given in the schema documentation.

- <procedure> see section 3.1.2
- <corrector> see section 3.1.2

- `<screen>` see section 3.1.3
- `<datablocks>` see section 3.1.4
- `<devices>` see section 3.1.5
- `<filters>` *optional* see section 3.1.6
- `<stimuli>` see section 3.1.7
- `<connections>` *optional* see section 3.1.8
- `<randomgenerators>` *optional* used to set parameters to random values, for example useful to implement level roving. An example of level roving is given in section 6.2
- `<calibration>` *optional* see section 3.1.9
- `<results>`*optional* see section 3.1.10 and section 7.3.4
- `<interactive>`*optional* see section 3.1.11
- `<general>`*optional* see section 3.1.12

In the next paragraphs these elements will be described in more detail and their main child elements will be given.

3.1.1 Declaration of Main <apex> element

To associate an experiment file with the correct name space and schema, the following attributes are necessary:

```
1 <apex:apex xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
2    xmlns:apex="http://med.kuleuven.be/exporl/apex/3.1.1/experiment">
3 ...
4 </apex:apex>
```

For more information on name spaces and schema declarations, we refer to the XML specifications. For APEX 3 it is sufficient to copy the text above. Note that the namespace contains a version number, in the example 3.1.1. When APEX opens an experiment file with a lower schema version number than its own version number, it will attempt to upgrade the experiment file to the new version. Any warnings or errors in the conversion process will be shown in the message window. Alternatively, you can manually upgrade the experiment file, by changing the version number, and fixing the validation errors that OxygenXML may show. Alternatively, you can use the experimentupgrader tool in the APEX 3 binaries folder.

explain some more, move elsewhere?

One modification to the above attributes can, however, be useful. If you use an XML editor with schema support (to allow autocompletion and showing documentation from the schema), it

needs to know where the APEX 3 schema (`experiment.xsd`) is located. This can be indicated by adding an absolute or relative path to the apex schema to the `xsi:schemaLocation` attribute. If for example APEX 3 is installed in the default location (`C:/Program Files/apex3`), the result would be as follows:

```
<apex:apex xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:apex="http://med.kuleuven.be/exporl/apex/3.1.1/experiment"
 xsi:schemaLocation="http://med.kuleuven.be/exporl/apex/3.1.1/experiment
␣file:C:/Program\%20Files/apex3/schemas/experiment.xsd>"
...
</apex:apex>
```

Or you could refer to the schema that we provide on our web server. This has the advantage that you don't have to refer to the absolute path on a particular computer, but has the downside that it requires an internet connection:

```
<apex:apex xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xmlns:apex="http://med.kuleuven.be/exporl/apex/3.1.1/experiment"
 xsi:schemaLocation="http://med.kuleuven.be/exporl/apex/3.1.1/experiment
␣https://exporl.med.kuleuven.be/apex/schemas/3.1.1/experiment.xsd">
...
</apex:apex>
```

3.1.2 Procedures

The Procedure controls the flow of an experiment. It decides on the next screen to be shown and the next stimulus to be presented. Generally a procedure will make use of a list of predefined trials.

e.g.

```
<procedure xsi:type="apex:constantProcedure">
   <parameters>
      ...
   </parameters>

   <trials>
      <trial id="trial1" >
         ...
      </trial>
      <trial id="trial2" >
         ...
      </trial>
   </trials>
</procedure>
```

3.1. Main APEX elements

The `xsi:type` attribute determines the type of procedure that will be used. There are 5 procedure types implemented:

constantProcedure: each trial is presented a specified number of times

adaptiveProcedure: the dependent variable (e.g.: level) of the stimulus depends on the response of the subject

trainingProcedure: a stimulus is presented AFTER the user has indicated the trial by pressing a button on the screen. If no corresponding trial is found, an error message is shown. This procedure is designed to be used as a replacement for the constant procedure to allow subjects to perform training before starting the actual experiment

pluginProcedure: allows the user to implement a custom procedure using ECMAScript, see section C.1.

multiProcedure: contains several child procedures of any of the other 4 types and selects between them. This procedure ends when all child procedures have ended

Every `<procedure>` contains the following elements:

- `<parameters>` specifies the workings of a procedure, e.g., specifies the number of presentations per trial. Please read section 3.2
- `<trials>` specifies each trial that can be presented. A trial consists of a screen to be shown, a stimulus to be presented and the correct response. Each of these can be defined here.

The procedure determines whether the experiment is completed.

One particular parameter for the procedure is the corrector. It compares the user's response to the correct response and can be used by the procedure to determine the next trial. The type of corrector to be used is defined using the `xsi:type` attribute. E.g.:

```
1
2  <procedure xsi:type="apex:constantProcedure">
3    <parameters>
4     <presentations>2</presentations>
5     <order>sequential</order>
6     <corrector xsi:type="apex:isequal"/>
7    </parameters>
8    ...
9  </procedure>
10
11 \begin{lstlisting}
12    <corrector xsi:type="apex:isequal"/>
```

Two main Correctors exist:

- `isequal`: compares whether the text defined in the `<answer>` element in `<trial>` and the result from the screen (the subject's response) are exactly the same
- `alternatives`: this corrector is used together with `<choices >1>` in `<procedure/paramete`

too much detail about corrector here? The element is now optional.

3.1.3 Screen

A screen is a GUI (Graphical User Interface) that is to be used by the subject to respond to the stimulus that was presented. The layout of the screen can be changed in different ways and a variety of elements can be added to the screen, such as buttons, labels, pictures and movies (take a look in the APEX reference manual for an exhaustive list of all elements that can be added or changed). A brief overview of the possible elements is given below.

```
1  <screens>
2     <general>
3       ...
4     </general>
5     <reinforcement>
6       ...
7     </reinforcement>
8     <style_apex> ... </style_apex>
9     <style> ... </style>
10    <childmode> ... </childmode>
11    <defaultFont> ... </defaultFont>
12    <defaultFontsize> ... </defaultFontsize>
13    <screen id="screen1">
14      ...
15    </screen>
16 </screens>
```

- `<general>` Some general properties of the entire APEX window are set in this section, such as removing the panel on the right hand of the screen or adding a repeat button. The properties are applied for each particular screen defined below.
- `<reinforcement>` indicates whether a progress bar is shown and whether feedback is given, and defines what kind of feedback about the procedure and correctness of answers is shown to the user. This is valid for the whole experiment. `<reinforcement>` includes:

  - <progressbar> as the value is **true** a progress bar will be displayed in the right hand bottom corner of the screen. The bar indicates how many trials have been completed and how many remain. When conducting an experiment in an adaptive way, it shows when a reversal occurs. The progress bar will increase at every reversal while the number of trials varies. The progress bar is not functional for multiprocedure or trainingprocedure.
  - <feedback> shows visual feedback according to the correctness of the last answer. Feedback is shown using an upward or downward pointing thumb in the right hand panel when the value is set to true. No feedback is given as the value is **false**.

  Other elements can be added to <reinforcement> but are optional:

  - <feedback length> duration of time after response (in msec) that APEX 3 waits before presenting the next trial. During this interval feedback can be displayed. If feedback is false, but a length is present, the specified time will be the time between to trials.
  - <feedback on> When set to **clicked**, the clicked screenelement is highlighted (i.e. the subject's answer). The correct screenelement is highlighted (irrespective of the answer of the subject) when the value is set to **correct**. No highlighting occurs as the value is **none**.
  - <feedback picture positive> and <feedback picture negative> can be used to set a feedback picture different from the standard thumb up/down. A prefix for the filename has to be specified.
  - <showcurrent> When set to **true**, the element corresponding to the currently playing stimulus is highlighted.

- <style_apex> Style that has to be applied to the whole of Apex.
- <style> Style that has to be applied to all screens.
- <childmode> Defines the elements used in child mode. In <childmode>, the experiments are adapted to the interest of children. It mainly involves a different screen panel and enables intro/outro movies.
- <defaultFont> and <defaultFontsize> Name and size of the default font to be used for all elements of every screen.
- <screen> a screen is defined for each stimulus. Each screen has an ID by which it can be referred to elsewhere in the experiment file (cf in <procedure>/<trial> to associate it with a stimulus).

3.1.4 Datablocks

A datablock is an abstraction of a basic block of data that can be processed by APEX 3 and can be sent to a matching device. In the case of a sound card, the datablock would be an audio signal in the form of a series of samples that is commonly stored on disk as a so-called wave file.

```
1 <datablocks>
2    <uri_prefix>...</uri_prefix>
3    <datablock id="datablock_star">
4       <device>soundcard</device>
5       <uri>star.wav</uri>
6    </datablock>
7 </datablocks>
```

Basically, in element `<datablock>`, a wave file on disk is assigned an ID, such that it can be referred to elsewhere in the experiment file.

- `<uri_prefix>` indicates the path where APEX will look for the file, relative to the path where the experiment file is stored. A prefix can be specified inline or by specifying the ID of a prefix in the APEX 3 config file and setting attribute `source` to `apexconfig`. You can specify a complete URI or part of it. See section 3.3

- `<datablock>` for each wave file a datablock is defined, with an ID.

3.1.5 Devices

A Device is a system connected to the computer that can be controlled by APEX 3. Devices can send signals to a transducer. Examples are sound cards and interfaces to cochlear implants (CIs). Device(s) can have parameters that can be controlled by APEX 3.

```
1 <devices>
2    <device id="soundcard" xsi:type="apex:wavDeviceType">
3       <channels>2</channels>
4       <gain>0</gain>
5       <samplerate>44100</samplerate>
6    </device>
7 </devices>
```

The following device types are currently supported:

wavDeviceType for sound cards

L34Type for an interface to cochlear implants

3.1.6 Filters

Filters are used to process data before sending it to a device. For example, if you want to change the level of your signal, you can use an *amplifier*, which is a specific kind of filter.
However, not every experiment needs filters. If the signals are routed directly to the output device without any processing, the section "<filters>" can be dropped.

Currently implemented filters are (<filter xsi:type= ... >):

amplifier a type of filter which allows to amplify or attenuate sound data

```
1 <filter xsi:type="apex:amplifier" id="ampli">
```

generator a special type of filter which generates a defined stimulus, such as a sine (SineGenerator) or white noise

```
1 <filter xsi:type="apex:generator" id="SineGenerator">
2   <type>sinus</type}
3 </filter>
```

dataloop : a type of filter which can loop a given datablock infinitely (continuous = true), and can jump randomly (randomjump = true)

```
1 <filter xsi:type="apex:dataloop" id="noisegen">
2   <continuous>true</continuous}
3   <datablock>noisedata</datablock}
4   <randomjump>true</randomjump}
5 </filter>
```

PluginFilter a type of filter which is an interface for implementing custom filters (see Chapter 5 Examples).

<filter> contains those elements which specify a filter or a generator

```
1 <filter id="noisegen" xsi:type="apex:dataloop">
2    <device>soundcard</device>
3    <channels>1</channels>
4    <continuous>false</continuous>
5    <datablock>noisedata</datablock>
6    <basegain>0</basegain>
7    <gain>0</gain>
8 </filter>
```

3.1.7 Stimuli

Stimuli are the auditory events presented to the subject. They can consist of one or more datablocks that are routed to any number of devices, simultaneously or sequentially.

```
1 <stimuli>
2   <fixed_parameters>
3   ...
4   </fixed_parameters>
5
6   <stimulus id="stimulus_sentence1">
7   ...
8   </stimulus>
9 </stimuli>
```

- <fixed_parameters> see section 3.2
- <stimulus>

3.1.8 Connections

Connections are defined to connect Datablocks, Filters and Devices. Any Datablock can be connected to any Filter or Device and any Filter can be connected to any other Filter or Device.

```
1   <connections>
2     <connection>
3      ...
4     </connection>
5   </connections>
```

<connections> defines how the different filters are routed to the output device(s)

APEX can visualise the connections by clicking on the "show stimulus connections" item in the menu. An example connection diagram is shown in figure 3.1.

3.1.9 Calibration

In order to present any given stimulus at a known sound pressure level, a calibration gain is applied before the stimulus is played. Calibration is the process of measuring which calibration gain is necessary for a given stimulus to achieve a certain sound pressure level. Calibration parameters are set in the<calibration> element of the apx file. Here a <stimulus> ID and parameters to be calibrated are defined. Since the stimuli used in most experiments are short, it is often helpful to make a longer version of the stimulus for use during calibration. This stimulus must have it's own <datablock> and <stimulus> element definitions. e.g:

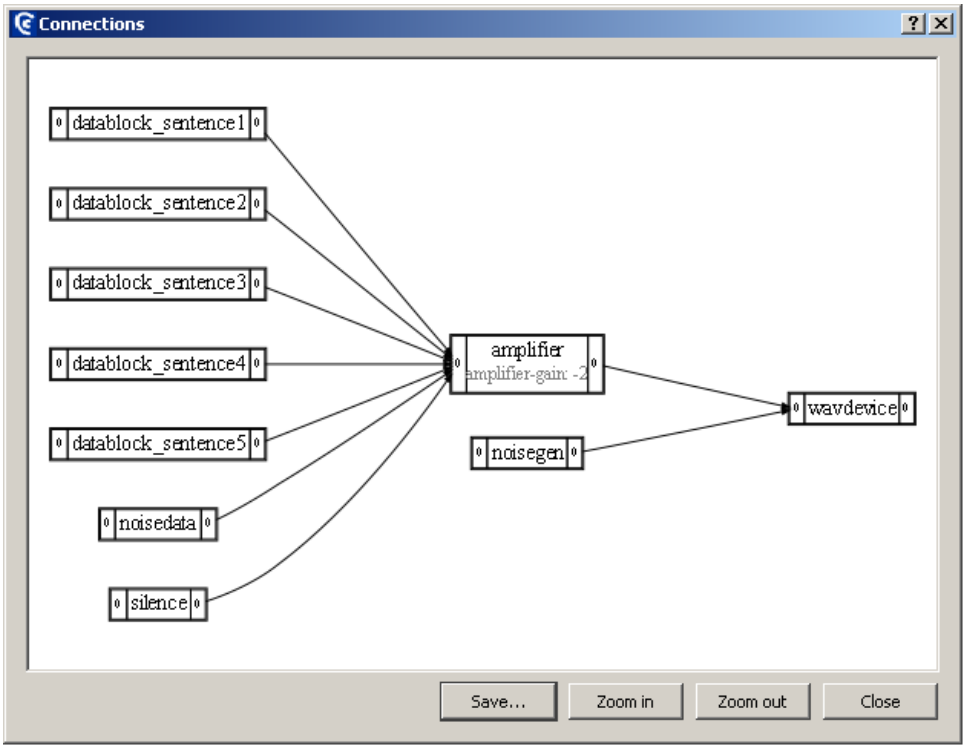

Figure 3.1: The connections window

```
1
2  <datablock id="datablock_tone">
3    <device>wavdevice</device>
4    <uri>tone.wav</uri>
5  </datablock>
6
7  <datablock id="datablock_tone_long_calibration">
8           <device>wavdevice</device>
9           <uri>tone_long_calibration.wav</uri>
10 </datablock>
11
12 <stimulus id="stimulus_tone">
13   <datablocks>
14   <sequential>
15     <datablock id="datablock_tone"/>
16   </sequential>
17   </datablocks>
18   <variableParameters></variableParameters>
19   <fixedParameters></fixedParameters>
20 </stimulus>
21
22 <stimulus id="stimulus_tone_long_calibration">
23          <datablocks>
24             <sequential>
25                <datablock id="datablock_tone_long_calibration␣"/>
26             </sequential>
27          </datablocks>
28          <variableParameters></variableParameters>
29          <fixedParameters></fixedParameters>
30       </stimulus>
31
32 <calibration profile="Calibprofile_HIB">
33       <stimuli>
34          <stimulus id="stimulus_calibration_tone"/>
35 <stimulus id="stimulus_calibration_noise"/>
36       </stimuli>
37       <parameters>
38          <parameter id="gainid_amp_tone">
39             <targetamplitude>30</targetamplitude>
40             <calibrationamplitude>80</calibrationamplitude>
41             <mute>-150</mute>
42             <min>-100</min>
```

```
43              <max>100</max>
44          </parameter>
45          <parameter id="gainid_amp_noise">
46              <targetamplitude>15</targetamplitude>
47              <calibrationamplitude>80</calibrationamplitude>
48              <mute>-150</mute>
49              <min>-100</min>
50              <max>100</max>
51          </parameter>
52      </parameters>
53  </calibration>
```

The parameter to be calibrated is usually a gain parameter for an amplifier element. In the example there are two stimuli that need to be calibrated. The stimuli have datablock ID's `stimulus_tone` and `stimulus_noise`. Since the stimuli are derived from different datablocks and different WAV files, they must be calibrated separately. For this reason, the datablocks are connected to separate amplifier elements with IDs `ampid_sinusoid_F250` and `ampid_TENnoise`. In this example, these two amplifier gain parameters are calibrated.

Calibration consists of a GUI for calibrating parameters and saving and applying calibration results. Any stimulus defined in the experiment files can be used as a calibration stimulus.

Typically, calibration is the process of measuring which value of a digital parameter corresponds to a certain physical magnitude , e.g., determining which internal amplification is necessary for a given wave file to achieve a certain sound pressure level.

APEX 3 provides a GUI to ease calibration (figure 3.2). APEX 3 can only calibrate parameters, i.e., it can set a parameter to a certain value such that the resulting physical magnitude is the one defined in the experiment file in the `<calibration>` element.

Because often the same calibration is useful for a set of experiment files, calibrations are stored under so-called *calibration profiles*. The calibration profile to be used must be specified in the `<calibration>` element. For example if multiple experiment files are used for speech in noise tests, the calibration profile could be `SpeechInNoise`.

As it is possible to use APEX 3 on the same computer with the same experiment files and calibration profile in different contexts (e.g., different types of headphones, which have different calibration settings), APEX 3 makes use of the concept *hardware setup*. A hardware setup associates a label to a certain set of hardware devices. The current hardware setup can be selected at runtime, i.e., cannot be specified in the experiment file.

If `<calibration>` is defined in the experiment file and the calibration profile has not been calibrated before for the current hardware setup, the calibration window will be shown at the start of the experiment.

If the calibration profile was calibrated before, but you wish to recalibrate or change the current hardware setup, select *Recalibrate* from the menu, select *Manage profiles* and add the desired label for the current hardware configuration, eg. "RME + headphones", and click **Add** and **OK**. This configuration will also appear in the APEX 3 window in the lower right hand corner. Under **Details...** the different calibration profiles that have been calibrated before will be

shown (figure 3.3).

On the left hand side of the main calibration window the parameters to be calibrated are shown (as specified in the `<calibration>` element in the experiment file). In this example the parameter `cardgain` is calibrated, i.e. the gain of the sound card. First allow the calibration stimulus ample time to be able to measure an accurate value with a measuring device or use the averaging function of the device. Enter the measured amplitude (in **Measured amplitude**) and click on '**Correct output parameter**'. Repeat until the intended and measured values are the same. To save the result, click **apply**, and click on the OK button. APEX 3 will store this profile and retrieve it each time it is used.

Important

- Remember to check whether the correct *Hardware setup* is selected when starting an experiment. APEX 3 will always use the hardware setup used in the previous experiment unless otherwise specified.

- **Apply** and **OK**. Each calibrated parameter needs to be saved by clicking on Apply and after all parameters are calibrated clicking on OK at the bottom of the dialog.

It can be of interest to calibrate at a higher level than the desired stimulation level, to avoid interference of background noise. If a yellow message appears, the signal has clipped.

Important: Calibration values are stored as a value of a parameter for a combination of calibration parameter ID, calibration profile name, and hardware setup. When APEX opens an experiment file with a calibration section, it will search an internal database for a value corresponding to this combination. The database is stored in the Windows registry or in a text file under `~/.config` in Linux. By default the database is specific for the current user. If you want to share calibration values across users, you need to make a profile global using the `calibrationadmin` tool.

explain more?

Calibration values can be exported and imported to transport them across computers, or simply store them for future reference. This feature is available in the manage hardware setups screen.

explain more?

3.1.10 Results

After completion of the experiment a file containing results can be saved to disk. The default results file is in XML and it has the extension **.apr**. It contains all the information about the course of the completed experiment(e.g. number of reversals/trials, mean of reversals/trials in case of an adaptive procedure). APEX automatically assigns a default name to the results file. Namely, it append **-results** to the name of the experiment file (e.g. gapdetection-results.apr). APEX will never overwrite an existing results file, but will append a number to the results file in case of an existing results file (e.g. gapdetection-results-1.apx). The subjects name can be appended automatically to the suggested results file by specifying it in the **<subject/>** element. This is typically modified at the beginning of an experiment using an interactive entry (see section 3.1.11).

e.g.:

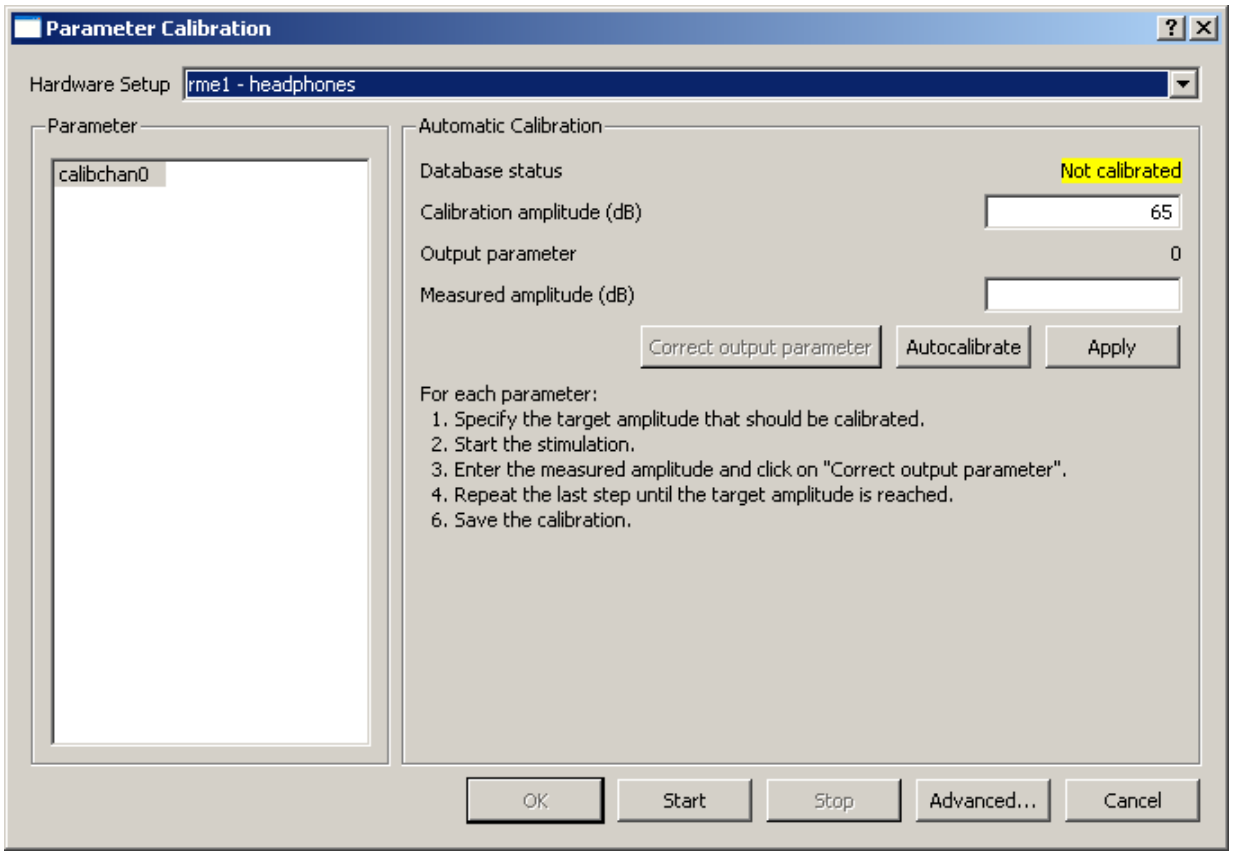

Figure 3.2: The calibration window

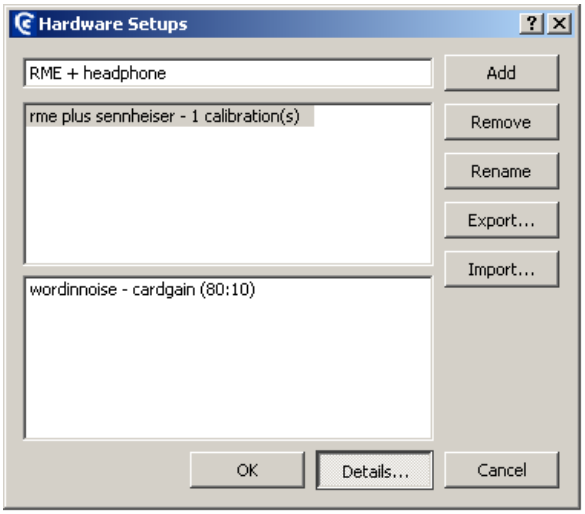

Figure 3.3: The calibration window showing *Manage profiles*

```
<results>
  <subject>John Doe</subject>
</results>
```

Please read section 7 for more information about displaying and analysing results.

3.1.11 Interactive

Interactive allows the experimenter to modify certain aspects of the experiment file right before the experiment is started using a GUI.

- <Entry>

3.1.12 General

General defines some general parameters.

- <show_results>
- <saveprocessedresults>

3.2 Parameters

There are two types of parameters:

fixed parameter: a fixed parameter is a property of a stimulus. It cannot be changed by APEX 3 at runtime and is defined when the experiment file is created. It can be used by the procedure to select a stimulus from a list, it can be shown on the screen or it can be used as a piece of information when analyzing results

```
<stimuli>
   <fixed_parameters>
     <parameter id="gap"/>
   </fixed_parameters>

   <stimulus id="stimulus1" >
     <datablocks>
       <datablock id="gwith5msinterval" />
     </datablocks>
     <fixedParameters>
         <parameter id="gap">5</parameter>
     </fixedParameters>
```

```
13      </stimulus>
14      ....
15  </stimuli>
```

Fixed parameters are often used by adaptive procedures. If a certain fixed parameter is selected to be adapted by a procedure, the stimulus to be presented will be selected using that fixed parameter. The stimuli are selected amongst those listed in the current trial. In this example the duration of the gap is varied in a gap detection task.

variable parameter: a variable parameter is a property of an object of which the value can be changed at runtime. Variable parameters can be set by various APEX 3 modules. Examples of modules that can define variable parameters are AdaptiveProcedure, Calibrator and Screen. Variable parameters can also be defined for a Stimulus. If a stimulus description contains a variable parameter, it will be set just before the stimulus is presented. Examples of modules that can have variable parameters (to be set by another module) are Filter, Controller and Device.

.

```
1
2   <stimuli>
3    <fixed_parameters/>
4     <stimulus id="stimulus0">
5       <description>Stimulus0</description>
6       <datablocks>
7         <datablock id="datablock0"/>
8       </datablocks>
9       <variableParameters>
10        <parameter id="speaker">3</parameter>
11        </variableParameters>
12        <fixedParameters/>
13     </stimulus>
14  </stimuli>
```

A variable stimulus parameter, is a parameter that will be set elsewhere just before the stimulus is sent to the device. Take care that a variable parameter is not set by different modules and/or a stimulus, as the resulting behaviour is undefined.

e.g. In examples 1 and 2 the gain of an amplifier is made a variable parameter by assigning it ID `gain`

3.2.1 Apexconfig

The `apexconfig.xml` file is a configuration file that applies to all experiments. It is stored in the folder `config`, under the main APEX 3 folder. It includes common names of sound cards and drivers and prefixes that can be referred to from any experiment file.

If the apexconfig.xml is not defined for the current system user the one in the APEX 3 folder will be used. It is, however, also possible to have an apexconfig.xml file per user. To add an apexconfig.xml file for a specific user, log in as the user, and copy the apexconfig.xml file to the folder C:/Users/LoginName/Application Data/ExpORL. If the file apexconfig.xml is present in the latter folder, it will override the apexconfig.xml file in the config folder under main APEX 3 directory. APEX 3 can automatically make this copy when you click the "edit apexconfig file" option in the help menu.

3.3 Using prefixes

When specifying a prefix in an experiment file (e.g. in <datablocks> or <screens>), it is possible, but not necessary to write out the entire path in the XML file. This path can also be defined once in the apexconfig.xml file located in the directory config.

In other words, a prefix can be specified inline (i.e.: as the content of the <uri_prefix> element) or by specifying the ID of a prefix in the APEX 3 config file and setting attribute source to apexconfig. You can specify a complete <uri>or part of it.

A relative path given in the <uri_prefix> element is always relative to the path of the experiment file. Since APEX 3 knows the location of the experiment file, only the folder containing the wave files (and pictures) must be specified. If for example the experiment file resides in c:/temp/files/experiment.apx, the prefix ../.. will point to c:/temp.

In the experiment file, it is possible to specify the prefix as a relative path, relative to the location of the apex experiment, as follows:

```
1   <uri_prefix>../stimuli</uri_prefix>
```

Alternatively, it is possible to specify it as:

```
1 <uri_prefix source="apexconfig">speechmaterials</uri_prefix>
```

The prefix *"speechmaterials"* that you are using in the experiment file is defined in apexconfig.xml. Sometimes it is useful to make your own apexconfig file (see section **??**). This way, you can create several prefixes that for example refer to a folder containing all your stimuli (several speech materials, noise, modulated stimuli etc.) You can link your prefix with the absolute or relative path of the folder as follows:

```
1 <prefix id="regression">file:../stimuli</prefix>
```

It is good practice to avoid using absolute paths in experiment files, as this makes your experiments portable to other computers.

Chapter 4

Scripting APEX

APEX is designed to allow setting up experiments without programming. However, in some cases it can be convenient to script some tasks, to avoid repetitive typing work and in case special behaviour is desired which it not built into APEX. In the following sections, we will describe how to implement custom procedures using Javascript (section 4.1), and how to automatically generate trials, datablocks and stimuli (section 4.2).

The programming language used for these scripts is JavaScript, as implemented by Qt. Abundant online documentation of the Javascript language is available , and in most cases modifying examples provided with APEX will suffice to achieve the desired results.

add reference to good tutorial

Note that these are more advanced APEX features, and while they are convenient, in most cases it is possible to avoid using them.

4.1 Plugin procedures

As an example of a plugin procedure, we will discuss the adjustment procedure that is included with the APEX examples (examples/procedure/adjustment-pluginprocedure.apx).

4.1.1 The experiment file

In the experiment file, the plugin procedure is specified as follows:

```
1  <procedure xsi:type="apex:pluginProcedure">
2        <parameters>
3           <presentations>1</presentations>
4           <order>sequential</order>
5           <script>adjustmentprocedure.js</script>
6           <parameter name="screen">screen</parameter>
7           <parameter name="startvalue">-5</parameter>
8           <parameter name="parameter">gain_ac</parameter>
9           <parameter name="maxvalue">10</parameter>
10       </parameters>
```

```
11        <trials>
12           <trial id="trial">
13              <screen id="screen"/>
14              <stimulus id="stimulus"/>
15           </trial>
16        </trials>
17     </procedure>
```

They key here is the use of `xsi:type="apex:pluginProcedure`, and referring to a javascript file in the `script` element. In this case, we refer to `testprocedure.js`. APEX will try to find the in the following folders (order as indicated):

1. the folder where the experiment file resides
2. the pluginprocedures folder in the main APEX folder

In this case, APEX will search for adjustmentprocedure.js in the experiment folder (examples/procedure), will not find it there, then look in the apex/pluginprocedures and find it there.

Next to the default procedure parameters (`<presentations>`, `<order>`, etc.), extra parameters can be passed on to the pluginprocedure as indicated in the example above. They will be made available to the Javascript code in a variable `params`. In Javascript syntax, the parameters above would be specified as:

```
1 params = { "presentations": 1,
2      "order": "sequential",
3      "screen": "screen",
4      "startvalue": -5,
5      "parameter": "gain_ac",
6      "maxvalue": 10 };
```

4.1.2 The javascript file

In the javascript file, a class needs to be implemented, inheriting from ScriptProcedureInterface, as follows:

```
1 adjustmentProcedure.prototype = new ScriptProcedureInterface();
```

Function `getProcedure()` should be implemented, and return an instance of the desired procedure:

```
1 function getProcedure()
2 {
3    return new adjustmentProcedure();
4 }
```

A constructor can be added to initialise some variables:

```
function adjustmentProcedure()
{
   gain = parseFloat(params["startvalue"]);
   screen = params["screen"];
   stim = "stimulus";
   hasMax = !(params["maxvalue"] === null);
   this.button='';
   if (hasMax) {
      maxValue = parseFloat(params["maxvalue"]);
   }
}
```

Here the params variable is used to get access to parameters defined in the experiment file.

The following methods of ScriptProcedureInterface can be reimplemented: processResult, setupNextTrial, firstScreen, resultXml, progress, checkParameters. These methods are documented in detail in section 4.1.3. They will be called by APEX in the following order:

checkParameters checks whether all parameter values make sense. If so, return an empty string, otherwise return an error message:

```
adjustmentProcedure.prototype.checkParameters = function()
{
   if (params['parameter'] === null)
      return "adjustmentProcedure: parameter not found";
   if (params['startvalue'] === null)
      return "adjustmentProcedure: parameter startvalue not found";
   if (params['screen'] === null)
      return "adjustmentProcedure: parameter screen not found";
   return "";
}
```

firstScreen return the first screen to be shown, before the user clicks the start button to start the experiment

```
adjustmentProcedure.prototype.firstScreen = function()
{
   return screen;
}
```

setupNextTrial returns the next trial to be presented. This is where most of the work happens. This function should return an object of class `Trial`. Class Trial is specified in detail in section 4.1.5.

```
1  adjustmentProcedure.prototype.setupNextTrial = function ()
2  {
3    // some code removed for brevity, cf adjustmentprocedure.js
4      var t = new Trial();
5      t.addScreen(screen, true, 0);
6      var parameters = {};
7      parameters[params["parameter"]] = gain;
8      t.addStimulus(stim, parameters, "", 0);
9      t.setId("trial");
10     return t;
11 }
```

processResult After the user has given a response, and therefore the trial has finished, `processResult` is called with argument screenresult. This is where the plugin procedure decides whether the user's response was correct, and stores the response if desired. It should return a class of type ResultHighLight specifying with element should be highlighted on the screen to give feedback. `screenresult` is documented in section 4.1.6 and `ResultHighlight` in section 4.1.7. In this case, we store the name of the button that was clicked and return an empty ResultHighlight element, indicating that nothing should be highlighted.

```
1  adjustmentProcedure.prototype.processResult = function(screenresult)
2  {
3     this.button = screenresult["buttongroup"];
4     return new ResultHighlight;
5  }
```

progress can be implemented to indicate the value to be shown on the progress bar. It can return a value between 0 and 100.

To further aid writing plugin procedures, 2 extra facilities are available: a javascript library with convenient functions and an API to APEX. The javascript library resides in `apex/pluginprocedures` The APEX API is described in section **??**.

4.1.3 ScriptProcedureInterface

ProcedureInterface

Include doxygen documentation

4.1.4 ScriptAPI

`#include <scriptprocedureapi.h>`

Inherits QObject.

Public Slots

- QVariant **parameterValue** (const QString &parameter_id)
- void registerParameter (const QString &name)
- QObjectList trials () const
- float **apexVersion** () const
- QStringList **stimuli** () const
- QVariant **fixedParameterValue** (const QString stimulusID, const QString parameterID)
- void messageBox (const QString message) const
- void **showMessage** (const QString &message) const
- void **abort** (const QString &message) const

Signals

- void **showMessageBox** (const QString title, const QString text) const

Public Member Functions

- **ScriptProcedureApi** (ProcedureApi ∗, ScriptProcedure ∗p)

Detailed Description

API for script procedure Is a proxy to ScriptProcedure, but only exposes relevant methods

Member Function Documentation

void apex::ScriptProcedureApi::messageBox (const QString *message*) const `[slot]`
Shows the user a messagebox with the specified messagebox

void apex::ScriptProcedureApi::registerParameter (const QString & *name*) `[slot]`
Register a new parameter with the manager.

QObjectList apex::ScriptProcedureApi::trials ( ) const `[slot]` Get list of trials for this procedure will return a QList of ApexTrial objects

The documentation for this class was generated from the following files:

- src/plugins/apexprocedures/scriptprocedureapi.h
- src/plugins/apexprocedures/scriptprocedureapi.cpp

4.1.5 Trial

`#include <trial.h>`

Inherits QObject.

Public Types

- typedef QList< Trial > **List**

Public Slots

- virtual void addScreen (const QString &id, bool acceptResponse, double timeout)
- virtual void addStimulus (const QString &id, const QMap< QString, QVariant > &parameters, const QString &highlightedElement, double waitAfter)
- virtual void **setId** (const QString &id)
- virtual void **setAnswer** (const QString &answer)
- virtual bool **isValid** () const
- virtual int **screenCount** () const
- virtual QString **screen** (int screenIndex) const
- virtual bool **acceptResponse** (int screenIndex) const
- virtual double **timeout** (int screenIndex) const
- virtual int **stimulusCount** (int screenIndex) const
- virtual QString **stimulus** (int screenIndex, int stimulusIndex) const
- virtual QMap< QString, QVariant > **parameters** (int screenIndex, int stimulusIndex) const
- virtual QString **highlightedElement** (int screenIndex, int stimulusIndex) const
- virtual bool **doWaitAfter** (int screenIndex, int stimulusIndex) const
- virtual double **waitAfter** (int screenIndex, int stimulusIndex)
- virtual QString **id** () const
- virtual QString **answer** () const
- virtual void reset ()

Public Member Functions

- **Trial** (QObject *parent=0)
- **Trial** (const Trial &other)
- virtual Trial & **operator=** (const Trial &other)

Detailed Description

Configuration of the next trial to be presented

Member Function Documentation

void apex::data::Trial::addScreen (const QString & *id,* bool *acceptResponse,* double *timeout*) `[virtual], [slot]` Add a screen to the list of screens to be shown Every subsequently added stimulus will be associated with this screen

Parameters

id	the id of the screen
acceptresponse	if true, the response of this screen will be reported in the next processResult call
timeout	if timeout >0, the screen wil disappear after timeout s. If acceptresponse is true, timeout must be <=0

void apex::data::Trial::addStimulus (const QString & *id,* const QMap< QString, QVariant > & *parameters,* const QString & *highlightedElement,* double *waitAfter*) `[virtual], [slot]` Add a stimulus to be played during the last added screen

Parameters

id	id of the stimulus to be added
parameters	parameters to be set before the stimulus is played
highlightElement	id of screenelement to be highlighted during presentation of this stimulus
waitafter	time in s to wait after the stimulus was played

void apex::data::Trial::reset ( ) `[virtual], [slot]` Clear all data

The documentation for this class was generated from the following files:

- src/lib/apexdata/procedure/trial.h
- src/lib/apexdata/procedure/trial.cpp

4.1.6 Screenresult

`#include <screenresult.h>`

Public Types

- typedef Parent::const_iterator **const_iterator**

Public Member Functions

- const QString **toXML** () const
- virtual void **clear** ()
- void **SetStimulusParameter** (const QString &parameter, const QString &value)
- const stimulusParamsT & **GetStimulusParameters** () const
- void **setLastClickPosition** (const QPointF &point)
- const QPointF & **lastClickPosition** () const

- Parent::const_iterator **begin** () const
- Parent::const_iterator **end** () const
- const ValueType **value** (const KeyType &key, const ValueType &defaultValue=QString()) const
- ValueType & **operator[ ]** (const KeyType &key)
- const ValueType **operator[ ]** (const KeyType &key) const
- bool **contains** (const KeyType &key) const
- const Parent **get** () const

Friends

- QDebug **operator**<< (QDebug dbg, const ScreenResult &screenResult)

Detailed Description

Map containing the answers of a certain presentation on screen

Author

Tom Francart,,,

The documentation for this class was generated from the following files:

- src/lib/apexdata/screen/screenresult.h
- src/lib/apexdata/screen/screenresult.cpp

4.1.7 ResultHighlight

ResultHighlight (procedureinterface.h)
Inherits QObject, and apex::ResultHighlight.

Public Member Functions

- **ScriptResultHighlight** (QObject ∗parent=0)
- bool **getCorrect** () const
- void **setCorrect** (bool p)
- QString **getHighlightElement** () const
- void **setHighlightElement** (const QString &p)

Properties

- bool **correct**
- QString **highlightElement**

Additional Inherited Members

The documentation for this class was generated from the following file:

- src/plugins/apexprocedures/scriptresulthighlight.h

4.1.8 Debugging

To debug plugin procedures, you can use the debugger. Simply put the command `debugger;` in your code, and when this is reached, a graphical debugger will be shown, allowing to inspect data structures etc.

4.2 Plugin XML

4.2.1 Trials, datablocks and stimuli

start with simpler example, inline javascript that returns straight XML

When many trials/datablocks/stimuli need to be specified in the experiment file, or you want to parametrize them, it can be useful to generate the corresponding XML code programmatically rather than typing it by hand. This can be achieved by the use of plugintrials/plugindatablocks/-pluginstimuli. We will illustrate this according to the example `examples/xmlplugin/autotrials.ap` The experiment file looks like this:

```
<procedure xsi:type="apex:constantProcedure">
...
  <trials>
    <plugintrials>
      <parameter name="path">../stimuli/wd*.wav</parameter>
      <parameter name="screen">screen1</parameter>
    </plugintrials>
  </trials>
</procedure>
...
<datablocks>
  <plugindatablocks>
    <parameter name="path">../stimuli/*.wav</parameter>
    <parameter name="device">wavdevice</parameter>
  </plugindatablocks>
</datablocks>
...
<stimuli>
  <pluginstimuli>
    <parameter name="path">../stimuli/*.wav</parameter>
```

```
21      <parameter name="prefix">stimulus_</parameter>
22    </pluginstimuli>
23  </stimuli>
24  <general>
25    <scriptlibrary>autostimulus.js</scriptlibrary>
26  </general>
27 </apex:apex>
```

At the beginning of the trials/datablocks/stimuli section, the element plugintrials/plugindatablocks/pluginstimuli can be specified, referring to javascript code. There are three options to specify the javascript code:

1. Specify the code directly in the experiment file
2. Refer to a javavascript file from the plugintrials/plugindatablocks/pluginstimuli element
3. Specify a scriptlibrary in `general/scriptlibrary` that contains all functions

The latter approach was chosen for this example.

In element plugintrials/plugindatablocks/pluginstimuli, parameters can be specified that will be made available to the javascript code. In this case the parameters are `path` and `prefix`, which specify the path in which to search for wav files, and the prefix that will be prepended to the trial/datablock/stimulus ID.

The content of autostimulus.js is the following:

```
1  function getStimuli () {
2     misc.checkParams( Array("path", "prefix"));
3     files = api.files( params["path"] );
4     r="";
5     for(i=0; i<files.length; ++i) {
6        print( files[i] );
7        r=r+xml.stimulus( params["prefix"]+files[i], "datablock_" + files[i
8     }
9     return r;
10 }
11 function getDatablocks() {
12   ...
13 }
14 function getTrials() {
15   ...
16 }
```

Functions getStimuli, getDatablocks and getTrials will be called by APEX, and should return the XML code for stimuli, datablocks and trials. A javascript library of convenience function is made available (`apex/pluginprocedures/pluginscriptlibrary.js`), and an API

exposed by APEX, see section 4.2.3 for a complete description. In this case the function api.files is used to get a list of wav files in a specified path. For each of these files, a stimulus/datablock-/trial is then generated. Note the use of convenience function xml.stimulus etc. to generate the XML code.

show example output of getStimuli() etc and walk reader through step by step

4.2.2 Pluginscreens

Similarly to trials, datablocks and stimuli, screens can also be generated using JavaScript code. In this case the `pluginscreens` element is used. See examples/xmlplugin/screenplugin.apx.

4.2.3 XML Plugin API

`#include <xmlpluginapi.h>`

Inherits QObject.

Public Slots

- QString **version** ()
- QStringList files (QString path)
- QString stripPath (QString s)
- void addWarning (const QString &warning)
- void addError (const QString &warning)

Detailed Description

API for XML plugins expanded by scriptexpander

Member Function Documentation

void apex::XMLPluginAPI::addError (**const QString &** ***warning***) **`[slot]`** Add error message to message window

void apex::XMLPluginAPI::addWarning (**const QString &** ***warning***) **`[slot]`** Add warning message to message window

QStringList apex::XMLPluginAPI::files (**QString** ***path***) **`[slot]`** Get all files in the filesystem matching wildcard

QString apex::XMLPluginAPI::stripPath (QString *s*) `[slot]` Remove wildcard from the end of s

The documentation for this class was generated from the following files:

- src/lib/apexmain/parser/xmlpluginapi.h
- src/lib/apexmain/parser/xmlpluginapi.cpp

4.2.4 Debugging

To debug generating XML from JavaScript code, the usual mechanisms are available, such as the use of the `debugger;` statement in JavaScript, which will open a graphical debugger. You can also save the experiment file in its expanded form after opening it in APEX, using the "Experiment|Save experiment as..." option in the menu.

4.3 Screens

To make screens more interactive, `<htmlelement>` can be used.

```
<screen id="screen1">
  <gridLayout width="1" height="2">
    <html row="1" col="1" id="htmlelement">
    <page>page.html</page>
  </html>
</screen>
```

`page.html` can contain HTML and Javascript, allowing for advanced user interaction.

Cf. loudness adaptation example

An API is available.
APEX will call the following functions:
When the screen is enabled:

```
reset(); enabled();
```

When the trial is finished, and the screenresult is needed.

```
getResult(); // returns string
```

When the user has finished answering, `api.answered()` should be called.

4.3.1 HtmlAPI

Inherits QObject.

Signals

- void **answered** ()

Public Member Functions

- **HtmlAPI** (QObject ∗parent=0)

The documentation for this class was generated from the following files:

- src/lib/apexmain/screen/htmlapi.h
- src/lib/apexmain/screen/htmlapi.cpp

Chapter 5

Examples

In this chapter we describe 4 examples: 1) a closed-set identification task, 2) an adaptive speech in noise identification task, and 3) an adaptive just noticeable difference task (gap detection), and 4) same as 3, but modified to the attention span and interest of young children. Each example consists of 1) a general description, 2) the concept, and 3) the implementation in XML.

It is advised to run the experiment before reading the details. The experiment files are stored in the APEX 3 directory under `examples/manual` in the APEX 3 folder, together with sound files and figures of the respective experiments. Note again: if the experiment file has the extension ".apx" it remains an XML file that can be edited with OxygenXML. The results file will automatically have the extension ".results". **(needs to be done)**.

5.1 Overview of examples provided with APEX

5.1.1 calibration

calibration/calibration.apx

Short Simple calibration use

Description The experiment shows how to calibrate using the calibration screen. You can calibrate the left and right channel seperately.

How Use the calibration element with a calibration profile and an id for each channel (left and right)

calibration/calibration-automatic-bk2250.apx

Short example of automatic calibration with B&K SLM 2250 plugin

Description calibrate left and right ear at the start of the experiment

How The <soundlevelmeter> block specifies how to use the sound level meter. If the sound level meter is not found, the regular calibration dialog is shown.

calibration/calibration-automatic-dummyslm.apx

Short example of automatic calibration with dummy SLM plugin

Description Allow to experiment with automatic calibration without having a sound level meter handy

How The <soundlevelmeter> block specifies how to use the sound level meter.

calibration/calibration-automatic-localisation.apx

Short Automatic calibration using interface to sound level meter

Description 13-loudspeaker sound source localisation experiment, with automatic calibration through the interface to the B&K SLM2250 Sound Level Meter

How Each channel of the sound card has a gain. These gains are specified in the <calibration> element. Here the <soundlevelmeter> block specifies how to use the sound level meter. If the sound level meter is not found, the regular calibration dialog is shown.

calibration/calibration-connections.apx

Short

Description

How

5.1.2 childmode

childmode/childmode-car-full.apx

Short Shows use of child mode: intro and outro movies, child panel

Description The experiment starts after a silent introductory movie. The child needs to find the animal in one of the three cars. One car has a different sound than the two other cars.

How Flash elements are introduced that read movie files with extension .swf

childmode/childmode-extramovies.apx

Short Demonstration of extra movies (snail movie)

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three snails. One snail has a different sound than the two other snails. Childfriendly feedback is provided.

How Flash elements of snails

childmode/childmode-full.apx

Short Shows use of child mode: intro and outro movies, child panel, progressbar, shortcuts.

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs. The progressbar and feedback are also childfriendly.

How Flash elements are introduced that read movie files with extension.swf, child panel activated by <panel> in <childmode> (<screens>) + button shortcuts are introduced (button 1 = 1, button 2 = 2, button 3 = 3)

childmode/childmode-full-L34.apx

Short Shows use of child mode: intro and outro movies, child panel, progressbar for L34

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs. The progressbar and feedback are also childfriendly.

How Flash elements, L34 implementation in <devices>

childmode/childmode-justmovies.apx

Short Shows use of child mode: without intro and outro movies, but with the normal panel

Description The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs. The same progressbar is used as in adult experiments, feedback is childfriendly.

How Flash elements, same panel as adult experiments (no childmode selected in <screens>)

childmode/childmode-movies+intro.apx

Short Shows use of flash movies, but without the actual childmode (no childpanel)

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. The same progressbar is used as in adult experiments. Feedback is childfriendly.

How flash elements, same panel as adult experiments (no <panel>reinforcement.swf</panel> in childmode)

childmode/childmode-nofeedback.apx

Short Shows use of child mode without special feedback movies

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs.

How Flash elements, no special feedback with flash elements provided

childmode/childmode-noprogress.apx

Short Shows use of child mode: intro and outro movies, but without the actual childmode (no childpanel)

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs. Feedback is childfriendly.

How Flash elements, no panel or progressbar (<showpanel>false</showpanel> and <progressbar>false</progressbar>)

childmode/childmode-onlyscreen-normalpanel.apx

Short Shows use of child mode: intro and outro movies, but with the normal panel

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs. The same progressbar is used as in adult experiments. Feedback is childfriendly.

How Flash elements, same panel as adult experiments (no childmode selected in <screens>)

childmode/childmode-poll.apx

Short Shows use of child mode: instead of waiting a specific time, 0 is specified as length for intro, outro and feedback, which makes apex wait for the movies to finish.

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs. The same progressbar is used as in adult experiments.

How Flash elements, same panel as adult experiments (no childmode selected in <screens>), length of intro and outro is specified in <childmode>

childmode/childmode-shortcuts.apx

Short Shows use of child mode with shortcut keys

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs. The progressbar and feedback are also childfriendly.

How Flash elements, button shortcuts are introduced (leftarrow = 1, downarrow = 2, rightarrow = 3)

childmode/childmode-skipintrooutro.apx

Short Shows use of child mode: intro and outro movies, but without the child panel; also has button shortcuts to skip the intro or outro movie

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs. Feedback is also childfriendly.

How The F7 shortcut can be used to skip the intro and outro movies when <allowskip>true</allowskip> in <general>. Flash elements, button shortcuts are introduced (leftarrow = 1, downarrow = 2, rightarrow = 3, 's' = skipping intro or outro movie),

childmode/childmode-skipintrooutro-disabled.apx

Short Skip intro or outro movie by hitting 's' on the keyboard is disabled.

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs. Feedback is childfriendly.

How Flash elements, skipping is disabled by setting <allowskip>false</allowskip> in <general>

childmode/childmode-snake-full.apx

Short Shows use of child mode: intro and outro movies, child panel, progressbar

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three snakes. One snake has a different sound than the two other snakes. The progressbar and feedback are also childfriendly.

How Flash elements of snakes, child panel activated by <panel> in <childmode> (<screens>)

childmode/childmode-waitforstart.apx

Short Shows use of child mode, with no shortcut keys and waiting for start before every trial

Description The experiment starts after a silent introductory movie. The child needs to find the stimulus in one of the three eggs. One egg has a different sound than the two other eggs. The next trial is only presented after selecting Start from the Experiment menu or pressing F5.

How Flash elements + <waitforstart> function in <general>

5.1.3 connections

connections/connections-all-mixed-monostereo.apx

Short The use of ALL connections

Description Monaural presentations of digit 1. Level is changed by clicking the buttons quieter and louder

How Connections defines how the different datablocks and filters are routed to the output device

connections/connections-byname.apx

Short Try to autoconnect a stereo wavfile to a mono wavdevice by name

Description Digits are presented dichotically, you can give a verbal response

How Connections are used to route different datablocks and filters to an output device

connections/connections-byregexp.apx

Short Try to autoconnect a stereo wavfile to a mono wavdevice using mode=regexp

Description Dichotic presentation of digits, you can give a verbal response

How Connections are used to route different datablocks and filters to an output device

connections/connections-bywildcard.apx

Short Try to autoconnect a stereo wavfile to a mono wavdevice using mode=wildcard

Description Dichotic presentation of digits, you can give a verbal response

How Connections are used to route different datablocks and filters to an output device

connections/connections-negativechannel.apx

Short Test the use of channel -1

Description If something is connected to channel -1 of the output device, it should be ignored, no output when button wrong is clicked

How Connections are used to route filters to output devices

5.1.4 controller

controller/controller-pa5.apx

Short TDT programmable attenuator (PA5) controller

Description Attenuation of pa5 changes according to adaptive procedure

How

controller/controller-plugincontroller.apx

Short Plugincontroller

Description Gain of plugin controller varies according to user input

How the democontroller is used here to demonstrate how a plugin controller can be implemented.

controller/osccontroller.apx

Short

Description

How

5.1.5 datablocks

datablocks/datablocks-combination.apx

Short Combination of datablocks using sequential and simultaneous

Description Play datablocks similtaneously and sequentially

How Using the <simultaneous> and <sequential> tags in <stimulus>, datablocks can be organised and reused.

datablocks/datablocks-loop.apx

Short Use the same datablock multiple times in a stimulus

Description The datablock wavfile "één" is played twice within one stimulus

How set the number of replications of the datablock by setting the number of the loop, here it is <loop>2</loop>

datablocks/datablocks-nodevice.apx

Short demonstration of omission of device in datablock

Description When there is only one device in the experiment, it does not need to be explicitly mentioned in each datablock.

How APEX will automatically assign the device to each datatblock

datablocks/datablocks-repeat.apx

Short Use the same datablock multiple times in a stimulus

Description The datablock wavfile "één" is played four times within one stimulus

How sequential datablocks within stimulus1

datablocks/datablocks-silence.apx

Short Combination of silence and signal for 1 stimulus presentation

Description The stimulus in this example contains first silence (a delay) before you hear "één"

How Use sequential datablocks that include silence and wavfile/signals, e.g. also possible to put silence datablocks after the wavfile

datablocks/datablocks-simultaneous.apx

Short Play datablocks simultaneously

Description Demonstrate the use of simultaneous datablocks in a stimulus

How When datablocks are listed in the <simultaneous> element in <stimulu> they are played simultaneously

5.1.6 filters

filters/filters-amplifier-defaultparameters.apx

Short Show amplifier default parameters

Description the stimulus wd1.wav is played in the left ear, when you click the button 'play with default parameters', the stimulus is played again with the same gain (=0) (10 trials)

How fixed_parameters

filters/filters-amplifier-setgain.apx

Short Adapting device parameters: gain of device varies according to user input

Description the stimulus wd1.wav and noise are played simultaneously in the left ear, when you click the button 'higher'/'lower', the stimulus and the noise are played again with a different gain for the noise (= gain + 2/- 2, see output box 'parameterlist')

How adaptiveProcedure: gain is adjusted with stepsize = 2

filters/filters-complexprocessing.apx

Short More complex processing, can be used for testing skips - 2 noise stimuli (sinus & wivineruis.wav) with different gains (filters)

Description When you click on 'wrong' you hear in both ears two noise stimuli and in the left ear 2 times a sentence, the same happens when you click on the 'wrong' button again

How trainingProcedure, creating a sinus as noise-stimulus

filters/filters-dataloop-2simultaneous.apx

Short Shows use of dataloop generator : uses the same datablock twice (for 2 different dataloops)

Description When you click on the button 'correct/wrong', in both ears a noise stimulus is played and in the left ear 'one-two'(=stimulus1)/'silence-two'(=stimulus2) is presented - Dataloop: playing noise and according to the users input (correct/wrong) stimulus1 or stimulus2 is presented

How 2 dataloops: noise for trial1 and trial2

filters/filters-dataloop.apx

Short Shows use of dataloop generator - wivineruis.wav should play !not! continuously over trials (see filters - noisegen - apex:dataloop)

Description When you click on the button 'wrong/correct', in both ears a noise stimulus is played and in the left ear 'silence-two(twee)'(=stimulus2)/'one-two(een-twee)'(=stimulus1) is presented

How trainingProcedure - dataloop generator - continuous: false

filters/filters-dataloop-continuous.apx

Short Shows use of dataloop generator - wivineruis.wav should play continuously over trials (see filters - noisegen - apex:dataloop)

Description The noise is playing continuously over trials. When you click on the button 'dataloop - silence/dataloop - 1', in both ears a noise stimulus is played and in the left ear 'silence-two(twee)'(=stimulus2)/'one-two(een-twee)'(=stimulus1) is presented

How trainingProcedure - dataloop generator - continuous: true

filters/filters-dataloop-jump.apx

Short Shows use of jump parameter of dataloop generator

Description

How

filters/filters-dataloop-randomjump.apx

Short

Description

How

filters/filters-noisegen-setgain.apx

Short Basegain of noise: 0 - Aapting device parameters - gain of device varies according to user input

Description Noise and a stimulus (one-een) are presented simultaneously in the left ear, when you click on 'higher/lower', you hear again the noise and stimulus in the left ear but the gain of the noise is adjusted with +2/-2 (=stepsize, see output box)

How adaptiveProcedure - stepsize of gain - adapt_parameter (Procedure) - continuous:true

filters/filters-plugin.apx

Short

Description example already somewhere else as well!

How

filters/filters-plugin-matlabfilter.apx

Short Example using the matlabfilter plugin.

Description The plugin will use matlab to process data, one buffer at a time. Note that you should make sure your system can find the Matlab eng library.

How <plugin>matlabfilter</plugin> in <filters>.

filters/filters-plugin-scamblespectrumfilter.apx

Short Demonstrate scramblespectrum filter

Description The scramblespectrum filter will randomize the spectrum to reduce monaural spectral cues in a localisation experiment

How parameters to the scramblespectrumfilter are specified in <filter>

filters/filters-plugin-scamblespectrumfilter_calibration.apx

Short

Description

How

filters/filters-pulsegen.apx

Short Shows synchronisation for L34 using a soundcard pulse

Description When you click on send, a pulse is presented in your left ear

How filters: generator makes a soundcard pulse => 2 channels of stereo file are mixed into output ch 0, the pulse goes to ch 1 (left ear)

filters/filters-sinegenerator.apx

Short Test of setting frequency parameter of sine generator - frequency of sinus varies according to user input

Description A sinus is playing in the left ear when you click on one of the 3 buttons. If you click on 100Hz/1000Hz/defaults, the frequency of the sinus changes to 100Hz/1000Hz/10 000Hz.

How filter: 1 generator: sinus (10 000Hz) - 3 stimuli: 3x silence with adjusted parameters (frequency: 100Hz/1000Hz/no change) - continuous: false

filters/filters-sinegen-setfrequency.apx

Short Regression test for adapting device parameters - gain of device varies according to user input

Description A sinus is played in the left ear (sinus is played continuously over trials), when you click the button 'higher'/'lower', the sinus is played again with a different frequency for the sinus (= frequency + 100Hz/- 100Hz, see output box 'parameterlist')

How adaptiveProcedure: frequency (adapt_parameter) is adjusted with stepsize = 100Hz - datablock/stimulus: silence

filters/filters-sinegen-setgain.apx

Short Regression test for adapting device parameters - gain of device varies according to user input

Description A sinus is played in the left ear. The sinus is played continuously over trial but when you click on 'higher/lower' the sinus is played again but the gain of the sinus is adapted with +5/-5 dB.

How adaptiveProcedure - datablock/stimulus: silence - filter: generator: sinus - continuous: true - adapt_parameter: gain - (!basegain has to be low enough otherwise there is clipping!)

filters/filters-singlepulse-defaultparameters.apx

Short Shows synchronisation using a soundcard pulse

Description When you click on 'play with default parameters', a pulse is presented in your left ear simultaneously with a stimulus (wd1.wav - een/one)

How datablock/stimulus + filter: generator makes a soundcard pulse (negative polarity)

filters/filters-singlepulse-polarity.apx

Short Shows synchronisation for L34 using a soundcard pulse with a negative polarity (filter - pulsegen)

Description When you click on send, a pulse is presented in your left ear and a stimulus (datablock: _Input44_2.wav) is presented in your right ear

How filters: generator makes a soundcard pulse with a negative polarity

filters/filters-wavamplifier.apx

Short Regression test for training procedure - Output: number according to last input

Description When you click on the button 'play': in the left ear/channel: "een", is presented and in the right ear/channel: "twee", 20dB louder compared to the left ear, is presented.

How filter: amplifier => basegain (-10) + gain left channel(-10)/right channel(+10)

filters/filters-wavfadeincos.apx

Short Regression test for fader (begin/end of stimulus is not abrupt)

Description When you click on the button 'play fade/play NO fade', in both ears a sinus is presented with at the beginning of the stimulus a fade-in/NO fade-in (click)

How filter: fader in begin (cosine) => length: 400ms, direction: in (fade-in: begin of stimulus)

filters/filters-wavfadeinlin.apx

Short Regression test for fader (begin/end of stimulus is not abrupt)

Description When you click on the button play fade/play NO fade', in both ears a sinus is presented with at the beginning of the stimulus a fade-in/NO fade-in (click)

How filter: fader(linear) => length: 400ms, direction: in (fade-in: begin of stimulus)

filters/filters-wavfadeoutcos.apx

Short Regression test for fader (begin/end of stimulus is not abrupt)

Description When you click on the button 'play fade/play NO fade', in both ears a sinus is presented with at the end of the stimulus a fade-out/NO fade-in (click)

How filter: fader(cosine) => length: 400ms, direction: out (fade-out: end of stimulus)

filters/filters-wavfadeoutlin.apx

Short Regression test for fader (begin/end of stimulus is not abrupt)

Description When you click on the button 'play fade/play NO fade', in both ears a sinus is presented with at the end of the stimulus a fade-out/NO fade-in (click)

How filter: fader(linear) => length: 400ms, direction: out (fade-out: end of stimulus)

5.1.7 general

general/general-autosave.apx

Short Automatically save results to apr-file after experiment

Description Closed-set identification task with automatic saving of results

How General - autosave

general/general-exitafter.apx

Short Exit experiment when finished

Description Closed-set identification task, exit after finish

How General - exitafter=true

general/general-prefix-absolutepath.apx

Short Get the datablock prefix from the absolute path

Description Closed-set identification task

How datablocks uri_prefix full path

general/general-prefix-frommainconfig.apx

Short Get the datablock prefix from the main configfile

Description Closed-set identification task

How source=apexconfig

general/general-prefix-invalid.apx

Short Get the datablock prefix from the main configfile

Description Invalid prefix id specified: wavfile not found wd1.wav

How source=apexconfig

general/general-scriptparameters.apx

Short Demonstrate the use of general/scriptparameters

Description Digits 1-7 are presented monaurally, you can click on a button to go to the next one

How Scriptparameters - name=path, scriptparameters.js needed

general/general-showresults.apx

Short Show results after experiment

Description When the experiment is finished and the results are saved, you can choose to see the results

How Results - showafterexperiment=true

general/general-xmlerror.apx

Short Error in XML file

Description

How

5.1.8 interactive

interactive/invalid-entry.apx

Short Change small parameter right before the start of the experiment

Description GUI is shown with the element to be changed

How Interactive

interactive/setgain.apx

Short GUI to change some settings right before the experiment

Description GUI appears to set/change some settings

How Interactive

interactive/subject.apx

Short GUI to change some settings right before the experiment

Description GUI appears to set/change some settings

How The name of the subject, in <results>/<subject> is set from the interactive dialog that appears before the experiment is loaded.

5.1.9 l34

l34/addinvalidfilter.apx

Short Try to add wav filter to L34Device

Description Since it is not possible to add a wav filter to an L34Device, an error occurs

How see filters and device

l34/bilateral.apx

Short Test of synchronized bilateral CI setup

Description Test whether the stimuli are given from the first, the second or both L34 devices

How Including two L34 devices, stimuli can be presented stimultaneously

l34/bilateral-singlepulse.apx

Short Test of synchronized bilateral CI setup with pulse stimuli

Description Test whether the stimuli are given from the first, the second or both L34 devices

How Including two L34 devices, stimuli can be presented stimultaneously

l34/invaliddatablock.apx

Short Try to parse invalid datablock, should return error

Description datablock does not exist in folder stimuli, running the experiment gives an error.

How uri_prefix is used to find stimuli, invalid.qic does not exist

l34/L34DummySync.apx

Short Shows synchronisation for L34 using a soundcard pulse

Description 2 channels of stereo file are mixed into output ch 0, the pulse goes to ch 1

How use dummyDeviceType and wavDeviceType and check connections

l34/L34Sync.apx

Short Shows synchronisation for L34 using a soundcard pulse

Description 2 channels of stereo file are mixed into output ch 0, the pulse goes to ch 1

How use dummyDeviceType and wavDeviceType and check connections

l34/loudness_balancing.apx

Short Loudness balancing between two stimulation CI electrodes

Description Two signals are presented consecutively in the same CI, at two different stimulation electrodes. One electrode is the reference electrode and has a fixed current level, the comparison electrode has changing levels. The participant has to judge which signal is louder. An adaptive procedure is used to determine the balanced level.

How Use of adaptiveProcedure, plugindatablocks, balancing.js needed

l34/mapping1.apx

Short Use default map with current units between 1 and 255

Description

How See defaultmap

l34/mapping2.apx

Short Use real map to check mapping

Description

How See defaultmap values

l34/r126wizard.apx

Short R126 is the clinical fitting software

Description

How Use <defaultmap> <from_r126/>

l34/setvolume.apx

Short Change volume of CI stimuli (send simple XML file to L34 device)

Description Change the volume of the CI stimuli to 10,50,70 or 100 current units

How Add variable parameter id=l34volume, see device: volume 100 is default

l34/simpleAseq.apx

Short

Description

How

l34/simpleQIC.apx

Short power-up test?

Description

How

l34/simpleQIC-dummy.apx

Short

Description

How

l34/simpleQIC-SP12.apx

Short

Description

How

l34/triggertest.apx

Short

Description

How

l34/wordsQIC.apx

Short

Description

How

5.1.10 manual

manual/closedsetword.apx

Short Closed-set identification of words in noise with figures

Description A word is presented in noise and the subject responds by clicking on one of the 4 figures on the screen, repeated 3 times. Initial SNR is set via interactive GUI.

How full experiment

manual/gapdetection.apx

Short Gap detection

Description

How

manual/gapdetectionchild.apx

Short gap detection for children

Description

How

manual/opensetidentification.apx

Short Open set identification task

Description A word is presented in noise and the subject responds by typing the word, repeated 3 times. Initial SNR is set via interactive GUI.

How Full experiment

manual/sentenceinnoise.apx

Short Sentences in noise task

Description A sentence is presented in noise, the subject responds verbally, the test leader can indicatie whether the response was correct or incorrect, repeated 4 times. Initial SNR is set via interactive GUI.

How Full experiment

manual/trainingprocedure.apx

Short Training for constant procedure

Description A stimulus is presented after the user had indicated the trial by pressing a button on the screen

How apex:trainingProcedure

5.1.11 parameters

parameters/parameters-connection-filter.apx

Short Change the channel of a connection + stimulus has gain

Description Click on the right (or left button) to hear the sound in your right ear (or left ear).

How Training procedure, Fixed (stimulus) and variable parameters (channel) + possibility in code to change the gain of the stimuli with filter (amplifier)

parameters/parameters-connection-soundcard.apx

Short Change the channel of a connection using a parameter

Description Click on the right (or left button) to hear the sound in your right ear (or left ear)

How training procedure, Fixed (stimulus) and variable parameters (channel)

parameters/parameters-device-allchannels.apx

Short Change parameter (gain) of device channel

Description gain of device varies in BOTH channels according to user input (1 button = louder, 1 button = quieter)

How adaptive procedure, fixed (stimulus) and variable (gain) parameters, stepsize of 2 dB

parameters/parameters-device-singlechannel.apx

Short Change parameter (gain) of device channel

Description Gain of device varies in LEFT channel according to user input (1 button = louder, 1 button = quieter)

How Adaptive procedure, fixed (stimulus) and variable (gain) parameters, stepsize of 2 dB

parameters/parameters-filter.apx

Short Change parameter (gain) of device channel

Description gain of device varies in BOTH channel according to user input (1 button = louder, 1 button = quieter) + gain is visible on the screen

How adaptive procedure, fixed (stimulus) and variable (gain) parameters, stepsize of 2 dB, Parameterlists are introduced that show Left or Right gain on the screen

parameters/parameters-filter-channel.apx

Short Change parameter (gain) of device channel

Description gain of device varies in RIGHT channel according to user input (1 button = louder, 1 button = quieter) + gain is visible on screen

How adaptive procedure, fixed (stimulus) and variable (gain) parameters, stepsize of 2 dB, Parameterlists are introduced that show Left or Right gain on the screen

parameters/parameters-filter-channel-probe.apx

Short Change parameter (gain) of device channel

Description gain of device varies in LEFT channel according to user input (1 button = louder, 1 button = quieter) + gain is visible on screen

How adaptive procedure, fixed (stimulus) and variable (gain) parameters, stepsize of 10 dB, Parameterlists are introduced that show Left or Right gain on the screen. Additionally there is a probe filter that saves the output to disk.

parameters/parameters-restore.apx

Short Demonstrate restoring parameter values

Description When a parameter is set from a stimulus, and subsequently a stimulus is played in which the parameter is not set, it should be restored to its default value

How Parameter gain is set in stimulus1, and is not set in stimulus2. Therefore a gain of 0dB should be applied for stimulus2

parameters/parameters-spinbox.apx

Short Adjust noise manually while presenting speech and noise

Description gain of noise varies in right channel according to user input (1 button = arrow up, 1 button = arrow down) while the stimulus stays the same.

How constant procedure, Fixed (stimulus) and variable (noise gain), stepsize of 1 dB, noise is generated by filter

parameters/parameters-wavfadeinout.apx

Short Set parameters of fader

Description The fader filter allows to apply an up and down ramp to a channel

How There are two faders in <filters, faderin and faderout. Their parameters are set from the stimuli.

5.1.12 procedure

procedure/adaptive-1up-1down.apx

Short 1up-1down Adaptive procedure: Frequency of ups-downs is the same - Experiment stops after 6 reversals

Description When you click on the button 'correct or wrong', you hear a stimulus 'wd1.wav (een/one)' in both ears. When you click again, the gain of the stimulus decreases/increases according to the button (correct/false)

How adaptiveProcedure - stop_after_type: reversals - nUp/nDown - adapt_parameter (gain: gain of stimulus) -

procedure/adaptive-1up-2down.apx

Short 1up-2down Adaptive procedure: Frequency of ups-downs is NOT the same - Experiment stops after 6 reversals

Description When you click on the button 'correct or wrong', you hear a stimulus 'wd1.wav (een/one)' in both ears. When you click again, the gain of the stimulus StaysTheSame/Decreases/Increases according to the button (correct/false) and the number of ups and downs

How adaptiveProcedure - stop_after_type: reversals - nUp/nDown - adapt_parameter (gain: gain of stimulus)

procedure/adaptive-2up-1down.apx

Short 2up-1down Adaptive procedure: Frequency of ups-downs is NOT the same - Experiment stops after 6 reversals

Description When you click on the button 'correct or wrong', you hear a stimulus 'wd1.wav (een/one)' in both ears. When you click again, the gain of the stimulus StaysTheSame/Decreases/Increases according to the button (correct/false) and the number of ups and downs

How adaptiveProcedure - stop_after_type: reversals - nUp/nDown - adapt_parameter (gain: gain of stimulus)

procedure/adaptive_stopafter_presentations.apx

Short Stop after a specified number of presentations (3) - should in this case stop after 6 trials (presentation: every trial is presented once => 2 trials => 2x3 = 6)

Description When you click on the button 'correct or wrong', you hear a stimulus 'wd1.wav (een/one)' in both ears. When you click again, the gain of the stimulus StaysTheSame/Decreases/Increases according to the button (correct/false) and the number of ups and downs

How adaptiveProcedure - stop_after_type: presentations - adapt_parameter (gain: gain of stimulus)

procedure/adaptive_stopafter_presentations_invalid.apx

Short Error message because #presentations is not equal to #stop_after

Description Point of interest if you want to stop the experiment after some presentations! => #presentations = #stop_after

How procedure: presentations - stop_after

procedure/adaptive_stopafter_reversals.apx

Short Stop after a specified number of reversals (6) - changes between correct-false

Description When you click on the button 'correct or wrong', you hear a stimulus 'wd1.wav (een/one)' in both ears. When you click again, the gain of the stimulus Decreases/Increases according to the button (correct/false) and the stepsizes changes after 3 reversals

How adaptiveProcedure - stop_after_type: reversals - adapt_parameter (gain: gain of stimulus) - stepsize: change_after: reversals

procedure/adaptive_stopafter_trials.apx

Short Stop after a specified number of trials (10)

Description When you click on the button 'correct or wrong', you hear a stimulus 'wd1.wav (een/one)' in the left ear. When you click again, the gain of the stimulus Decreases/Increases according to the button (correct/false)

How adaptiveProcedure - stop_after_type: trials - adapt_parameter (gain: gain of stimulus)

procedure/adjustment-pluginprocedure.apx

Short Adjustment of stimuli with a pluginprocedure

Description There are 6 buttons:

How adaptiveProcedure - stop_after_type: trials - adapt_parameter (gain: gain of stimulus)

procedure/adp1.apx

Short Regression test for ADP - Experiment stops after 4 reversals

Description adaptive procedure, one of the following sequences are played: noise een noise, een noise noise, noise noise een => answer: one of the three sequences. Stepsize changes after 2 trials => stepsize determines the parameter 'snr'

How adaptiveProcedure - adapt_parameter (snr: snr (order, not in dB) - intervals - stepsize determines the change in snr (1-2-3) not the value in dB

procedure/adp2.apx

Short Regression test for ADP - Experiment stops after 10 reversals

Description adaptive procedure, one of the following sequences are played: noise een noise, een noise noise, noise noise een => answer: one of the three sequences. Stepsize changes after 2 trials => stepsize determines the parameter 'snr'

How adaptiveProcedure - adapt_parameter (snr: snr (order, not in dB) - intervals - stepsize determines the change in snr (1-2-3) not the value in dB

procedure/afc-choices.apx

Short Regression test - 4 Intervals - 4 Choices, select interval 2-3

Description Four intervals are possible= 'noise noise noise een', 'noise noise een noise', 'noise een noise noise' and 'een noise noise noise'. However only possibility number "2" and "3" are selected

How constantProcedure - intervals- count/possibilities: 4, select(ion): 2 and 3

procedure/afc-uniquestandard.apx

Short Regression test for Intervals, Standard(reference signal), Uniquestandard(true/false = If uniquestandard is true and multiple standards are defined per trial, Apex will try to present another standard in each interval of the trial)

Description In each trial, the 3 standards should be used (uniquestandard = true)

How constantProcedure - intervals- uniquestandard: true

procedure/aid1.apx

Short Regression test - Different stimuli can occur during one trial (according to the difficulty of the experiment ('snr') (- no connections or gain!)

Description adaptive procedure, one of the following stimuli are played: 'een', 'twee', 'drie', 'vier' or 'vijf' => If you click correct/false the experiment becomes more difficult/easy corresponding with the 'snr' parameter - the larger the values of snr, the easier the experiment

How adaptiveProcedure - adapt_parameter (snr: snr (order, not in dB) - different stimuli (trial) - stepsize determines the change in snr (1-2-3-4-5) - changes afther 5 trials

procedure/aid-selectrandom.apx

Short Regression test - Different stimuli can occur during one trial (according to the difficulty of the experiment ('snr')

Description adaptive procedure, one of the following stimuli are played: 'een', 'twee', 'drie', 'vier' or 'vijf', each with a different gain => If you click correct/false the experiment becomes more difficult/easy corresponding with the 'snr' parameter

How adaptiveProcedure - adapt_parameter (snr: snr (order, not in dB) - different stimuli (trial) - stepsize determines the change in snr (1-2-3-4-5), gain(filters)

procedure/choices-randomstimulus-randomstandard.apx

Short Regression test - Show use of intervals and random selection of stimuli/standard(reference) - 4 presentations (2 trials => 8x)

Description adaptive procedure, select random stimulus, random standard (no selection) of the 2 trials => If you click correct/wrong (trial1) or true/false (trial2) your response is correct/false. The stimuli changes according to the 'snr' (the larger the values of snr, the easier the experiment)

How adaptiveProcedure - intervals(two choices: correct/wrong - true/false) - larger is easier:true - adapt_parameter (snr: snr (order, not in dB - see stimuli)

procedure/continuous-waitbeforefirst.apx

Short Shows how to start continuous filters etc before the first trial is presented

Description The noise is playing continuously over trials (dataloop) - the noise starts 5s before the first trial (presenting: wd1.wav: 'een') begins

How procedure: time_before_first_trial: in seconds - filters: dataloop - continuous: true

procedure/cst-multistimulus.apx

Short multiple stimuli per trial - one should be chosen randomly - 2 presentations (2trials => 2x2 = 4)

Description The experiment has 2 trials (trial1: button 1-2-3 - trial2: button a-b-c) => In trial1/trial2=> correct answer is always: button1,buttonb.

How constantProcedure - 2 trials with !each! 3 different stimuli but with the !same! buttons

procedure/extrasimple.apx

Short Click on the corresponding button - Stop after 2 presentations (2trials => 2x2 = 4)

Description The experiment has 2 trials with different stimuli (trial1: house - trial2: mouse) => In trial1/trial2 => correct answer is: house/mouse (stimulus is heard in the left ear)

How constantProcedure - 2 trials with different stimuli corresponding with the buttons

procedure/fixedparameternotfound.apx

Short Errormessage - fixed parameter with id="test" not defined

Description example of an error message about an undefined fixed parameter of a stimulus

How stimuli - fixed parameters

procedure/heartrainprocedure.apx

Short Regression test for heartrainprocedure.js

Description example of an error message about an undefined fixed parameter of a stimulus

How stimuli - fixed parameters

procedure/heartrainprocedure_short.apx

Short

Description

How

procedure/idn1.apx

Short Matching of stimuli and buttons - Different trials (+ answers) - 1 presentations (6 trials)

Description auditive stimulus 1 2 3 4 5 6 - input: buttons 1 2 3 4 5 6 => match the stimulus with the button

How 6 trials with each trial a different correct answer - 6 stimuli and 6 buttons - order: sequential

procedure/idn1-mono.apx

Short Matching of stimulus and button - 5 presentations (1 trial)

Description auditive stimulus in right ear '1' - input: buttons 1 2 3 4 5 6 => match the stimulus with the button (stimulus1 => button1)

How 1 trial - 6 stimuli and 6 buttons - 1 correct answer

procedure/idn1-skip.apx

Short skip = 2: Number of trials that will be presented before the actual presentations start.

Description skip=2 and presentations=2 => first 2 trials and then 2*6trials = 12 trials. If the order is sequential, the skipped trials will be the first skip trials from the trial list, repeated if necessary.

How 2 presentations - 6 trials - skip = 2 (6 stimuli - 6 buttons)

procedure/idn1-waitforstart.apx

Short Demonstrate use of wait for start

Description The next trial is only presented after selecting Start from the Experiment menu or pressing F5.

How <waitforstart> function in <general>

procedure/idn2.apx

Short Matching of stimuli and buttons - Different trials (+ answers) - 1 presentations (6 trials)

Description auditive stimulus 1 2 3 4 5 6 - input: buttons 1 2 3 4 5 6 => match the stimulus with the button

How 6 trials with each trial a different correct answer - 6 stimuli and 6 buttons - order: random

procedure/idn2-skip.apx

Short skip = 3: Number of trials that will be presented before the actual presentations start.

Description skip=3 and presentations=2 => first 3 trials and then 2*6trials = 12 trials. If the order is random, the skipped trials will be picked from the trial list without replacement, repeating this procedure if necessary.

How 2 presentations - 6 trials - skip = 3 (6 stimuli - 6 buttons)

procedure/input-during-stimulus.apx

Short Input during stimulus is allowed

Description skip=3 and presentations=2 => first 3 trials and then 2*6trials = 12 trials. If the order is random, the skipped trials will be picked from the trial list without replacement, repeating this procedure if necessary.

How Input during stimulus: true - trials with buttons (input) - stimuli (output)

procedure/kaernbach.apx

Short 1up-1down Kaernbach procedure: Frequency of ups-downs is the same - Up stepsize is 1, down stepsize is 7 - Experiment stops after 6 reversals

Description When you click on the button 'correct or wrong', you hear a stimulus 'wd1.wav (een/one)' in both ears. When you click again, the gain of the stimulus decreases/increases according to the button (correct/false)

How adaptiveProcedure - stop_after_type: reversals - nUp/nDown - adapt_parameter (gain: gain of stimulus) -

procedure/matrix_adaptief_spraak_original_01.apx

Short

Description

How

procedure/multiparameters-fixed.apx

Short Adaptation of multiple parameters (order(fixed) & gain(adapt))

Description Two buttons: louder - quieter => louder: increasing of order and gain with stepsize=2, quieter: decreasing of order and gain with stepsize=2 (see output box)

How Adapt_parameter => order & gain - stepsize for gain(filter) & order(stimuli) = 2

procedure/multiparameters-invalid.apx

Short Error message - parameter you want to be adapted is a fixed parameter (2nd adapt_parameter: order: is a fixed parameter)

Description error message: only first adaptive parameter can be a fixed parameter

How Adapt_parameter => gain(adapt) & order(fixed)

procedure/multiparameters-variable.apx

Short Adaptation of multiple parameters: GainL & GainR

Description 2 buttons: louder - quieter - when you click on louder/quieter the gain in right and left channel is increased/decreased with 2 dB (stepsize) - see output box

How Adapt_parameter => gainL(adapt) & gainR(adapt)

procedure/multiprocedure-aid.apx

Short Multiprocedure - 2 adaptiveprocedures with each their own parameters, procedure and trials

Description 2 procedures => procedure1/procedure2: 1trial - button_correct/button_wrong - When you click correct, the experiment in the next trial of the same procedure becomes more difficult (you hear a higher number => larger value of snr = stimulus (twee, drie, vier, vijf: stepsize:1)

How procedure => multiProcedure - larger_is_easier => snr(not in dB): larger value: easier (so if you click on the wrong_button => the experiment becomes easier)

procedure/multiprocedure.apx

Short Multiprocedure - 2 constantprocedures with each their own parameters, procedure and trials

Description 2 procedures => procedure1/procedure2: 3 trials - button1/button4 - button2/button5 - button3/button6 but different stimuli in each trial (Match the stimulus with the button)

How procedure => multiProcedure

procedure/multiprocedure-choices.apx

Short Multiprocedure - 2 constantprocedures with each their own parameters, procedure and trials - 2 different choices of intervals

Description 2 procedures => procedure1/procedure2: 1trial - 3intervals - Match the stimulus (place of stimulus within noise) with the correct button

How procedure => multiProcedure - intervals/choices - 3 choices - standardstimulus:noise

procedure/multiprocedure-choices-corrector.apx

Short Multiprocedure - 2 constantprocedures with each their own parameters, procedure and trials, #presentations

Description 2 procedures => procedure1: 3presentations/1trial => 3x - 3intervals: Match the stimulus (place of stimulus within noise) with the correct button

How procedure => multiProcedure - intervals/choices - standardstimulus:noise

procedure/multiprocedure-constant-train.apx

Short Multiprocedure - 2procedures (1constant - 1training) with each their own parameters, procedure and trials

Description 2 procedures => constantprocedure1: 4presentations/1trial => 4x - 3intervals: Match the stimulus (place of stimulus within noise) with the correct button

How procedure => multiProcedure - intervals/choices - standardstimulus:noise - constantProcedure - TrainingProcedure

procedure/multiprocedure-idn.apx

Short Multiprocedure - 2constantprocedures with each their own parameters, procedure and trials

Description 2 procedures => constantprocedure1: order: onebyone - 4presentations/1trial => 4x - 3intervals: Match the stimulus (place of stimulus within noise) with the correct button

How procedure => multiProcedure - intervals/choices - standardstimulus:noise - constantProcedure

procedure/multiprocedure-mixed.apx

Short Multiprocedure - 2constantprocedures with the SAME procedure, parameters and trials

Description 2 procedures => constantprocedure1: 1presentations/3trials => 3x - 3buttons: 1-2-3 Match the stimulus (place of stimulus within noise) with the correct button

How procedure => multiProcedure - constantProcedure

procedure/multiprocedure-onebyone.apx

Short Multiprocedure-order! - 3constantprocedures with the SAME procedure, parameters and trials (different stimuli)

Description 2 procedures => constantprocedure1: 2presentations/1trial => 2x - 2buttons: name of procedure & number: click on number (=1) if you have heard the stimulus (corresponding with the number)

How procedure => multiProcedure: order: ONEBYONE - constantProcedure

procedure/multiprocedure-random.apx

Short Multiprocedure-order! - 3constantprocedures with the SAME procedure, parameters and trials (different stimuli)

Description 2 procedures => constantprocedure1: 2presentations/1trial => 2x - 2buttons: name of procedure & number: click on number (=1) if you have heard the stimulus (corresponding with the number)

How procedure => multiProcedure: order: RANDOM - constantProcedure

procedure/multiprocedure-train-train.apx

Short Multiprocedure-order - 2trainingProcedures with DIFFERENT procedure, parameters and trials (different stimuli)

Description 2 procedures => trainingProcedure1: 3trials => 3x???? - 3buttons: click on a button and hear the corresponding stimulus

How procedure => multiProcedure: order: ONEBYONE- trainingProcedure

procedure/multiscreen-idn1.apx

Short Multiscreen - 1 procedure with for the first trials: screen 1 and the last trials: screen 2 => Same outcome as sequential Multiprocedure

Description 6 trials - 1 presentation: 3 buttons(screen1): 1-2-3 and 3 buttons(screen2): 4-5-6 => Match the stimulus with the corresponding button

How procedure => 1procedure but several screens: multiscreen- constantProcedure

procedure/multistandard-unique.apx

Short Multiple standards - Unique standard

Description 2 trials: trial1 = match one of the button with the noise-stimulus (standards/reference-signals = numbers)

How 2 trials - constantProcedure

procedure/noanswer.apx

Short Show warning if no answer is defined for a stimulus

Description Error message => Cannot show feedback because: no screen was found for (button turns red)

How procedure - trial1 - no answer (see comment)

procedure/nostandardfound.apx

Short Show warning if no answer is defined for a stimulus

Description Error message => Cannot show feedback because: no screen was found for (button turns red)

How procedure - trial1 - no answer (see comment)

procedure/open-set-constant.apx

Short Open set experiment - Constant stimuli

Description 6 trials (1 presentation) - You hear a stimulus (1-2-3-4-5-6) in both ears - Typ the answer/stimulus (een, twee, drie, ...) in the textbox below 'answer'.

How trial - answer>ANSWER THAT THE SUBJECT HAS TO TYPE< - corrector xsi:type="apex:isequal"

procedure/pause_between_stimuli.apx

Short a pause of 500ms should be introduced between the stimulus and the standards

Description 1 trials (10 presentations => 10x) - You hear a stimulus (1-2-3-4-5-6) in both ears - Typ the answer/stimulus (een, twee, drie, ...) in the textbox below 'answer'.

How pause between stimuli (in seconds)

procedure/pluginprocedure.apx

Short Pluginprocedure => script: testprocedure

Description When you click on a button with a certain number, the next time you have to click on the number you just have heard - parameter: stepsize & startvalue

How pluginProcedure with a certain script

procedure/randomchannel.apx

Short Randomgenerator test

Description the stimulus "een" & "twee" are played simultaneously but randomly in the left or right channel according the value returnd by random1

How trainingProcedure - randomgenerator - connections - stimuli:datablocks: simultaneous

procedure/repeatuntillcorrect-endaftertrials.apx

Short The first selected trial (of the first presentation) shoud be repeatedly presented (stop after 5 presentations, several trials)

Description 5 presentations x 5 trials = 25x - When you click correct => the next trial begins / When you click wrong, the first trial of the first presentation is repeated

How repeat_first_until_correct: true

procedure/repeatuntillcorrect-multitrial.apx

Short The first selected trial (of the first presentation) shoud be repeatedly presented (stop after 5 reversals, multiple trials)

Description 5 trials, stop after 5 reversals - When you click correct => the next trial begins / When you click wrong, the first trial of the first presentation is repeated

How adaptiveprocedure - REPEAT-FIRST-UNTIL-CORRECT: TRUE - adapt_parameter

procedure/repeatuntillcorrect-singletrial.apx

Short The first selected trial shoud be repeatedly presented: DOESN'T WORK! => single trial + adapt_parameter

Description 1 trial, stop after 5 reversals - When you click correct => the next trial begins / When you click wrong, the first trial begins

How adaptiveprocedure - repeat_first_until_correct: true - ADAPT_PARAMETER

procedure/selectrandom.apx

Short Test of random stimulus output when more than one stimulus defined in the experiment

Description Random selection of stimuli in a trial if there are more than one (=> sequential presentation of stimuli => more trials) - When you click on 1, 'een', 'twee', 'drie' is presented randomly in both ears

How constantprocedure - more than 1 stimulus in 1 trial => random selection of the stimulus

procedure/simple.apx

Short Randomgenerator test

Description the stimulus "house"/"mouse" are played in the left channel => click the right button (house/mouse)

How constantProcedure - stimuli - datablocks

procedure/soundquality.apx

Short Interval - select element ok, number 2

Description

How

procedure/time-between-trials.apx

Short Demonstrates the use of feedback to enforce a pauze between trials

Description If you click on a button, you hear a stimulus (1-2-3-4-5-6) in both ears => click on the button corresponding with the stimulus

How feedback length (in ms)

procedure/train1.apx

Short Trainingprocedure: demonstrate the presentation of a stimulus with a number according to last input

Description when you click on 1/2/3/4/5/6 => you hear een/twee/drie/vier/vijf/zes

How trainingProcedure with 6 trials and buttons

procedure/train2.apx

Short Show the presentation of a stimulus with a random number out of the possible choices according to last input

Description 3 buttons: 1-2 3-4 5-6 => when you click on 5-6, you hear either 5 or 6

How trainingProcedure - multiple trials => button5 is the answer of the trial corresponding with stimulus5 & stimulus6

procedure/trial-nostimulus.apx

Short Make a trial whithout a stimulus

Description when you click on stimulus => apex shuts down

How no stimuli - datablock

procedure/uniform-double.apx

Short Random generator test - gain of stimulus is randomly chosen

Description You hear 'een' in your left ear - the gain of this stimulus changes randomly, each time you click the button '1'

How random generator - type: DOUBLE (fractional number, e.g. 0.1) - parameter => see filter

procedure/uniform-int.apx

Short Random generator test - gain of stimulus is randomly chosen

Description You hear 'een' in your left ear - the gain of this stimulus changes randomly, each time you click the button '1'

How random generator - type: WHOLE(fractional number, e.g. -5) - parameter => see filter

5.1.13 randomgenerator

randomgenerator/multi-interval.apx

Short Randomgenerator: random gain of each datablock of a multi-interval stimulus

Description The stimulus "één één één" is heard each time by pressing on the "1" button, and the gain of each "één" has a random gain

How The id of the gain is set to be the parameter of the randomgenerator, three intervals are defined in procedure

randomgenerator/randomchannel.apx

Short Randomgenerator: random channel for stimulus

Description The stimulus "één" is played on randomly the left or right channel

How The value returnd by random1 decides whether it is channel 0 or channel 1, see <random-generators>

randomgenerator/uniform-double.apx

Short Randomgenerator: random gain of stimulus presented in 1 channel, possibility to use non-integer values (double type)

Description The stimulus "één" is heard in the right channel each time by pressing on the "1" button, and the gain of "één" has a random gain

How See <randomgenerators>, the id of the gain is set to be the parameter of the randomgenerator, minimum value contains a decimal value

randomgenerator/uniform-int.apx

Short Randomgenerator: random gain of stimulus presented in 1 channel, only possible to use integer values

Description The stimulus "één" is heard in the left channel each time by pressing on the "1" button, and the gain of "één" has a random gain

How See <randomgenerators>, the id of the gain is set to be the parameter of the randomgenerator, minimum and maximum values are integers

randomgenerator/uniform-int-invalid.apx

Short Randomgenerator should have integer values, similar to uniform-int.apx

Description Warning (no error) appears in messages window because random generator limits are not integer

How Randomgenerator max value is 2.5

randomgenerator/uniform-smallint.apx

Short Randomgenerator: random gain of stimulus presented in 1 channel, only possible to use integer values, only small range is used

Description The stimulus "één" is heard in the left channel each time by pressing on the "1" button, and the gain of "één" has a random gain (small range in this case)

How See <randomgenerators>, the id of the gain is set to be the parameter of the randomgenerator, minimum and maximum values are integers

5.1.14 results

results/results-adaptive-saveprocessedresults.apx

Short Demonstrate the use of result parameters

Description The parameters under 'results' will be written to the results file.

How Use of rtresults-test-procedureparameter.html

results/results-confusionmatrix.apx

Short Demonstrate the use of a confusionmatrix

Description Results are converted into a confusionmatrix

How Default results file for a confusionmatrix

results/results-procedureparameter-after.apx

Short Demonstrate the use of result parameters

Description Results are shown after the experiment is done.

How Results: show results after experiment

results/results-procedureparameter-during.apx

Short Demonstrate the use of real-time results

Description Real-time results are shown during the experiment in a separate window

How Results: show results during experiment

results/results-psignifit.apx

Short Demonstrate the use of 'psignifit' results

Description APEX contains an implementation of the psignifit library, which can fit psychometric functions to data

How Psignifit is called from the resultsviewer and the SRT and slope are shown numerically

results/results-resultparameters.apx

Short Demonstrate the use of resultparameters

Description Resultparameters specified in <results>/<resultparameters> will be passed on to the resultviewer

How These parameters will be made available in a hash params. Additionaly, the javascript code in <resultscript> will be executed when the resultsviewer is loaded.

results/results-saveprocessedresults.apx

Short Demonstrate the use of the saveprocessedresults tag

Description Results in CSV format will be appended to the results file, for easy importing in other software

How <saveprocessedresults> is set to true

results/results-subject.apx

Short Demonstrate how you can add the subjects name to the results file

Description When the subject types his/her name in the interactive, this will automatically appear in the results file and be appended to the results filename

How subject function in 'results', 'interactive' entry is added

5.1.15 screen

screen/button-shortcuts.apx

Short shows how to use shortcuts to answer

Description The screen shows different buttons: you can either click on them, or use a predefined shortcut button on the keyboard

How shortcuts are implemented for the buttons

screen/currentfeedback.apx

Short highlighting of the played stimulus

Description the currently playing stimulus/button is highlighted

How <showcurrent>true

screen/feedback-answer.apx

Short feedback with picture and highlighting of the CLICKED element

Description When clicking a button, a feedback picture (thumb up or down) is shown in the right panel and the CLICKED button is highlighted.

How <feedback_on> clicked

screen/feedback-both.apx

Short highlighted stimulus + feedback with picture and highlighting

Description The currently playing stimulus/button is highlighted. When clicking a button, a feedback picture (thumb up or down) is shown in the right panel and the correct button is highlighted.

How <showcurrent> true, <feedback> true

screen/feedback-ledfeedback.apx

Short

Description

How

screen/feedback-ledfeedback-localization.apx

Short

Description

How

screen/feedback-ledfeedback-showcurrent.apx

Short

Description

How

screen/feedback-multistimboth.apx

Short highlighted stimulus + feedback with picture and highlighting for multiple stimuli

Description The currently playing stimulus/button is highlighted. When clicking a button, a feedback picture (thumb up or down) is shown in the right panel and the correct button is highlighted.

How <showcurrent> true, <feedback> true

screen/feedback-multistimnormal.apx

Short feedback with picture and highlighting for multiple stimuli

Description When clicking a button, a feedback picture (thumb up or down) is shown in the right panel and the correct button is highlighted.

How <feedback> true

screen/feedback-multistimshowcurrent.apx

Short highlighted stimulus for multiple stimuli

Description the currently playing stimulus/button is highlighted

How <showcurrent> true

screen/feedback-nohighlight.apx

Short feedback without highlighting

Description When clicking a button, a feedback picture (thumb up or down) is shown in the right panel and NO elements are highlighted.

How <feedback_on>: none

screen/feedback-normal.apx

Short feedback with picture and highlighting

Description When clicking a button, a feedback picture (thumb up or down) is shown in the right panel and the correct button is highlighted.

How <feedback> true

screen/feedback-onlywait.apx

Short no visual feedback

Description No feedback is given after clicking the correct/wrong button (no highlighting/picture). Subject has to wait between trials.

How <feedback length= ...> false

screen/feedback-plugin.apx

Short Feedback using a plugin

Description Own feedback can be tested if dummyfeedbackplugin is changed

How dummyfeedbackplugin

screen/feedback-showcurrent.apx

Short highlighted stimulus

Description the currently playing stimulus/button is highlighted

How <showcurrent> true

screen/fgcolor.apx

Short shows how to change the font color of a button/label

Description A label and button with a different font color are shown on the screen

How <fgcolor>

screen/flash-disabled.apx

Short Shows full use of flash movies, but without the actual childmode. Shows how to disable a button.

Description the middle egg should not be clickable (disabled)

How <disabled>true

screen/flash-flashfeedback.apx

Short Shows full use of flash movies, but without the actual childmode

Description Corresponding flash movies are shown during presentation of the stimuli and after clicking the correct/wrong button.

How introduce flash elements: <flash>, <uri>, <feedback>

screen/flash-normalfeedback.apx

Short Pictures instead of buttons

Description Three eggs instead of buttons are shown.

How flashelements for screen

screen/fullscreen.apx

Short shows how to conduct an experiment in full screen modus

Description A typical experiment is shown, but the Windows titlebar/taskbar/... is hidden

How <fullscreen>true<fullscreen>

screen/general-itiscreen-annoyingtext.apx

Short shows how to use an intertrial screen

Description A screen is shown between two trials.

How <screens> <general> <intertrialscreen>iti

screen/general-itiscreen-picture.apx

Short shows how to use an intertrial picture

Description A picture is shown between two trials.

How <screens> <general> <intertrialscreen>iti

screen/layout-arc.apx

Short example of different arc layouts for buttons

Description Buttons are placed in several semi-circles on the screen

How <arcLayout> upper, lower, left, right, full

screen/layout-arc-invalidwidth.apx

Short Shows problems with arc layout for buttons

Description Error when loading the experiment: something went wrong with the placing of the buttons.

How <arcLayout> The number of x-values can't exceed the width of the arc. In this case, the width has to be changed to 9 (instead of 2)

screen/layout-arcs.apx

Short example of different arc layouts for buttons

Description Buttons are placed in several semi-circles on the screen

How <arcLayout> upper, lower, left, right, full

screen/layout-arc-single.apx

Short example of simple archwise organisation of buttons

Description Buttons are placed archwise/in a semi-circle on the screen.

How <arcLayout> upper, lower, left or right

screen/layout-grid.apx

Short example of gridlayout

Description buttons are placed in a grid

How <gridLayout>

screen/layout-grid-nested.apx

Short example of nested gridlayout

Description buttons are placed in nested grids

How <gridLayout> in <gridLayout>

screen/layout-grid-rowcol.apx

Short shows the use of the row and col attributes instead of x and y

Description first screen you see is made with col & row, second screen with x & y

How <gridLayout> replacing "x" & "y" by "row" & "col". col = column in the grid, is the same as x. row = row in the grid, is the same as y

screen/layout-grid-rowcol-invalid.apx

Short shows wrong usage of the row and col attributes

Description first screen you see is made with col & row, second screen with x & y

How <gridLayout> col should not be used together with x! col is the same as x, refers to column in the grid

screen/layout-grid-stretchfactors.apx

Short shows how to change the size of the buttons

Description Buttons showed on the screen have different sizes/stretchfactors.

How Stretch factor for the columns/rows: a list of integers separated by comma's. The width of the columns/rows will be proportional to the numbers.

screen/layout-grid-stretchfactors-invalid.apx

Short Shows problems with changing the size of the buttons

Description Error when loading the experiment: something went wrong when changing the size of the buttons.

How The number of the column stretch factors has to be equal to its width.

screen/layout-hlayout.apx

Short shows horizontal layout

Description buttons are placed next to each other, in a horizontal layout

How <hLayout>

screen/layout-mixed.apx

Short example of mixed layouts

Description buttons are placed in semi-circles and grids on the screen

How <arcLayout>, <gridLayout>

screen/nomenu.apx

Short shows how to hide the menu bar

Description A typical experiment is shown, but the menu bar on top of the screen is hidden.

How <showmenu>false<showmenu>

screen/panel-feedbackpictures.apx

Short shows how to implement a feedback picture, other than the traditional thumb

Description A green square is shown as feedback when clicking the correct button, a red square is shown when you give a wrong answer

How <feedback_picture_positive> & <feedback_picture_negative>

screen/panel-feedbackpictures-filenotfound.apx

Short shows error message for feedback picture

Description Experiment doesn't load, because the feedback pictures are not found.

How Wrong naming of feedback pictures / pictures don't exist

screen/panel-nopanel.apx

Short shows how to hide the panel

Description A typical experiment is shown, but the panel on the right side is hidden. Start the experiment by pressing F5.

How <showpanel>false<showpanel>

screen/panel-showrepeatbutton.apx

Short shows the use of a repeatbutton

Description A repeatbutton is shown on the screen, just above the progressbar. When clicking the button, the last stimulus is repeated.

How <repeatbutton>true<repeatbutton>

screen/panel-showstatuspicture.apx

Short shows the use of a statuspicture

Description A statuspicture (dot) is shown, just above the progress bar. The dot changes color: red when presenting the stimulus, green when waiting for an answer

How <statuspicture>true<statuspicture>

screen/panel-showstopandrepeatbutton.apx

Short shows how to use a stopbutton & repeatbutton

Description A red stopbutton and a repeatbutton are displayed on the screen, just above the progressbar. When clicking the stopbutton, all output is immediately stopped & apex is shut down. When clicking the repeatbutton, the last stimulus is repeated.

How <stopbutton> & <repeatbutton>

screen/panel-showstopbutton.apx

Short shows how to use the stopbutton

Description A red stopbutton is displayed on the screen, just above the progressbar. When clicking this button, all output is immediately stopped & apex is shut down

How <stopbutton>true>stopbutton>

screen/prefix-apexconfig.apx

Short shows how to use an 'apexconfig' as a prefix for a filename

Description idem

How <uri_prefix source="apexconfig">pictures</uri_prefix>

screen/prefix-inline.apx

Short shows how to use an 'inline' as a prefix for a filename

Description idem

How <uri_prefix>../../pictures</uri_prefix>

screen/screenelements-all.apx

Short example of a multitude of screenelements

Description

How

screen/screenelements-allbutflash.apx

Short assembly of different kinds of screenelements

Description Different labels and buttons are shown on the screen. First row containg a button and answerlabel, second row containing a label and a parameterlist, last row a textedit and picture

How <button>, <answerlabel>, <label>, <parameterlist>, <textEdit>, <picture>

screen/screenelements-answerlabel.apx

Short shows the use of an answerlabel

Description An answerlabel is shown on the screen, containing the correct answer for the current trial.

How add <answerlabel> to <gridLayout>

screen/screenelements-button-disabled.apx

Short shows how to disable a button

Description Two buttons are shown on the screen: nothing happens when clicking the disabled one

How <disabled> true

screen/screenelements-checkbox.apx

Short shows how to use a checkbox

Description Checkboxes are shown on the screen: if a checkbox is clicked, the subject is assumed to have responded to the trial

How <checkBox>

screen/screenelements-html.apx

Short how to use a html page

Description a html page is shown on the screen

How <html> <page>

screen/screenelements-label.apx

Short shows how to add text to a label

Description screen shows a large label filled with text

How <label> <text>

screen/screenelements-label-bgcolor.apx

Short shows how to change background and font color of a label/button

Description screen shows different labels/buttons with a different background color or font color

How <bgcolor> changes the background color, <fgcolor> changes the font color

screen/screenelements-label-html.apx

Short shows how to use a bold font by using html code

Description screen shows the text written in html code

How text

screen/screenelements-label-resize.apx

Short shows how to resize the text on a label

Description screen shows different labels, with and without resizing of the text

How <text> split text into different lines with 'enter' & '

screen/screenelements-matrix.apx

Short example of a matrix layout for buttons

Description Screen shows buttons in a 2x2 matrix. After listening to the stimulus, you have to select one button in each column before clicking OK.

How <matrix>

screen/screenelements-parameterlabel.apx

Short shows how to implement a parameterlabel on the screen

Description A parameterlabel is shown on the screen, i.e. a label containg the value of a parameter.

How screen: <parameterlabel>, stimuli: define parameter

screen/screenelements-parameterlist.apx

Short shows how to implement a parameterlist on the screen

Description Parameters of the played stimuli are shown on the screen. This example shows two different parameterlists (two different trials).

How <parameterlist>

screen/screenelements-picture-changepicture.apx

Short example of a picture-button

Description Click the picture after listening to the stimulus

How <picture>

screen/screenelements-picture-notfound.apx

Short shows error message for picture

Description experiment doesn't load, error message: picture not found

How wrong <uri> ('abc'-picture doesn't exist, can't be found in the pictures-folder)

screen/screenelements-pictures.apx

Short shows how to implement different pictures in the screen

Description 4 pictures are shown

How <picture>

screen/screenelements-pictures-disabled.apx

Short shows how to disable a picture-button

Description 4 pictures are shown on the screen, one is disabled (number 4)

How <picturelabel> instead of <picture> + picture 4 is not included in the button group

screen/screenelements-pictures-feedback.apx

Short

Description

How

screen/screenelements-pictures-invalid.apx

Short shows error message for picture

Description experiment doesn't load, error message: picture invalid

How wrong file extension for picture (xxx in stead of .png)

screen/screenelements-pictures-resize2.apx

Short example of screen layout with pictures

Description Screen shows 4 pictures, after hearing a stimulus you should click one

How <picture>

screen/screenelements-pictures-resize.apx

Short example of screen layout with pictures

Description Screen shows 4 pictures, after hearing a stimulus you should click one

How <picture>

screen/screenelements-samename.apx

Short Check whether everything is OK when two screenelements in different screens have the same name

Description /

How /

screen/screenelements-slider.apx

Short shows how to implement a vertical slider on the screen

Description Slider is shown on the screen. Sliding the bar doens't change anything to the presented stimuli.

How <slider>

screen/screenelements-slider-horizontal.apx

Short shows how to implement a horizontal slider on the screen

Description Slider is shown on the screen. Sliding the bar doens't change anything to the presented stimuli.

How <slider>

screen/screenelements-spinbox.apx

Short example of a spinbox

Description a spinbox is an input field that only accepts numbers and has 2 buttons to respectively increase or decrease its value

How add <spinBox> to Layout, define: startvalue, minimum & maximum

screen/screenelements-spinbox-parameter.apx

Short example of a spinbox with adjustable parameter

Description a spinbox is an input field that only accepts numbers and has 2 buttons to respectively increase or decrease its value

How add <spinBox> to Layout, define: startvalue, minimum & maximum, parameter to be adjusted

screen/screenplugin-emptyresult.apx

Short shows example of error message

Description Demonstrate a screen generated by javascript, show error message because of invalid xml

How wrong naming of source <pluginscreens>

screen/stylesheet-elements.apx

Short example of a different screenstyle

Description screen has an other background color, buttons changed color, highlighting changed, ...

How <style_apex>

screen/textinput-font.apx

Short example of font input

Description screen shows two 'input fields' with different fonts

How <textEdit> <font>

screen/textinput-inputmasks.apx

Short example of various restricted inputs

Description Screen shows four 'entry fields', but input is restricted (for example, only numbers allowed)

How <textEdit>, <inputmask> (Defined in the Qt documentation!)

screen/textinput-setfocus.apx

Short example of restricted input

Description Screen shows two 'entry fields', but input is restricted (for example, only numbers allowed)

How <textEdit>, <inputmask> numbers = only numeric input will be allowed

5.1.16 screenplugin

screenplugin/screenplugin-emptyresult.apx

Short

Description

How

5.1.17 soundcard

soundcard/soundcard-asio-multiplecard.apx

Short Demonstrate the usage of multiple soundcards

Description two soundcards are defined in <devices>

How an extra soundcard is defined in <devices>, each with their own parameters

soundcard/soundcard-cardname-inline.apx

Short Specify the name of the sound card to be used

Description Specify the sound card to be used.

How The name of the sound card to be used can be specified in devices/device/card. A list of devices in your system can be obtained by running apex from the commandline with parameter –soundcards

soundcard/soundcard-clipping.apx

Short Demonstrate what happens when the output clips

Description The output clips of the first two trials of this experiment

How In the results file, the following will be shown to indicate clipping occured: <device id="soundcard"> <clipped channel="0"/> </device>

soundcard/soundcard-defaultsettings.apx

Short Demonstrate default wavdevice settings

Description Shows default settings in 'devices'

How Devices: type of soundcard, number of channels (2), gain (0), samplerate (44100)

soundcard/soundcard-driver-asio.apx

Short Demonstrate the usage of different drivers and their parameters

Description In this example asio is used as a driver (basicwav-coreaduio/jack/portaudio show examples of different drivers)

How Devices: <driver>asio</driver>

soundcard/soundcard-driver-coreaudio.apx

Short Demonstrate the usage of different drivers and their parameters

Description In this example coreaudio is used as a driver (basicwav-asio/jack/portaudio show examples of different drivers)

How Devices: <driver>coreaudio</driver>

soundcard/soundcard-driver-jack.apx

Short Demonstrate the usage of different drivers and their parameters

Description In this example jack is used as a driver (basicwav-coreaduio/asio/portaudio show examples of different drivers)

How Devices: <driver>jack</driver>

soundcard/soundcard-driver-portaudio.apx

Short Demonstrate the usage of different drivers and their parameters

Description In this example portaudio is used as a driver (basicwav-coreaduio/jack/asio show examples of different drivers)

How Devices: <driver>asio</driver>

soundcard/soundcard-gain-invalidchannel.apx

Short Demonstrate the misapplication of adjusting gain to the wrong channel

Description Gain is applied to the wrong channel (channel = 10, when there are only 2 channels)

How Devices: <gain channel="10">0</gain> -> it should be channel="0" of channel="1", because there are only 2 channels defined.

soundcard/soundcard-mono-2channel.apx

Short Demonstrate what happens when you use wavdevice with one channel not connected

Description When you only use one of two defined channels you will not get an error, only a message: "not every inputchannel of soundcard is connected"

How Devices: 2 channels defined, only 1 used in <connections>

soundcard/soundcard-portaudio-6channels.apx

Short Demonstrate the usage of multiple channels

Description Add more channels to your experiment

How set more channels (e.g. <channels>6</channels, for 6 channels> and make connections with them so that the right stimulus is send to the right channel

soundcard/soundcard-portaudio-bigbuffersize.apx

Short

Description

How <buffersize>8192</buffersize> is used

soundcard/soundcard-portaudio-cardnotfound.apx

Short Demonstrate the misapplication of <card>

Description <card> specifies the name of the sound card. If you use the wrong cardname, you will get an error when you try to open your experiment in apex.

How wrong card name used in <card> -> 'Soundmax' should be default or Soundmax should be defined in apexconfig.xml

soundcard/soundcard-portaudio-interactivecard.apx

Short Demonstrate the misapplication of <card>

Description <card> specifies the name of the sound card to be used. If you use the wrong cardname, you will get an error when you try to open your experiment in apex.

How wrong card name used in <card> -> 'invalid' should be default or an other name defined in apexconfig.xml

soundcard/soundcard-wav-padzero.apx

Short Demonstrate the use of padzero

Description Add the given number of empty (filled with zero's) buffers to the output on the end of a stream. This avoids dropping of the last frames on some soundcards.

How Devices: Add padzero function (e.g. <padzero>2</padzero>)

5.1.18 xmlplugin

xmlplugin/autostimulus.apx

Short Demonstrate datablocks and stimuli generated by javascript

Description The datablocks and stimuli are automatically generated in this experiment using javascript

How using plugindatablocks and pluginstimuli, setting the pluginparameters and referring to the wanted javascript in <general> (here autostimulus.js)

xmlplugin/autotrials.apx

Short Demonstrate trials, datablocks and stimuli generated by javascript

Description The trials, datablocks and stimuli are automatically generated in this experiment using javascript

How using plugintrials, plugindatablocks and pluginstimuli, setting the pluginparameters and referring to the wanted javascript in <general> (here autostimulus.js)

xmlplugin/datablockplugin.apx

Short Demonstrate a datablock generated by javascript

Description The datablock is automatically generated in this experiment using javascript

How using plugindatablocks and referring to the wanted javascript in <datablocks> (here datablockgenerator.js)

xmlplugin/datablockplugin_l34.apx

Short Demonstrate generating XML code for L34 directly from plugindatablocks and pluginstimuli

Description The datablocks are automatically generated in this experiment using javascript

How using plugindatablocks, pluginstimuli, setting the pluginparameters and referring to the wanted javascript in <datablocks> and <stimuli> (here datablockgenerator_l34.js)

xmlplugin/screenplugin.apx

Short Demonstrate a screen generated by javascript

Description The screen is automatically generated in this experiment using javascript

How using pluginscreens, setting the pluginparameters and referring to the wanted javascript in <screens> (here screengenerator.js)

xmlplugin/screenplugin-inlibrary.apx

Short Demonstrate the misapplication of a screen generated by javascript

Description The screen is not generated in this experiment because the script source is invalid. You will get an error message when you try to open the experiment.

How script source refers to <general> instead of the script itself. It should be: <script source="screengenerator library"/> in <screens> OR <scriptlibrary>screengenerator-library.js</scriptlibrary> in <general>

xmlplugin/screenplugin-invalidscript.apx

Short Demonstrate the misapplication of a screen generated by javascript

Description The screen is not generated in this experiment because the defined javascript is empty.

How invalidxml.js is an empty/invalid javascript so no screen can be generated.

xmlplugin/screenplugin-invalidxml.apx

Short Demonstrate the misapplication of a screen generated by javascript

Description The screen is not generated in this experiment because the defined javascript is empty. You will get an error when you try to open the experiment.

How the javascript contains an error so no screen can be generated.

xmlplugin/screenplugin-multiscreen.apx

Short Demonstrate a screen generated by javascript

Description The screen is automatically generated in this experiment using javascript

How using pluginscreens, setting the pluginparameters and referring to the wanted javascript in <screens> (here screengenerator-multiscreen.js)

xmlplugin/stimulusplugin.apx

Short Demonstrate stimuli generated by javascript

Description The stimuli are automatically generated in this experiment using javascript

How using pluginstimuli, setting the pluginparameters and referring to the wanted javascript in <stimuli> (here stimulusgenerator.js)

Figure 5.1: Example of closed-set identification experiment

5.2 Example 1: Closed-set identification of words in noise with figures

5.2.1 General description of the experiment

See `examples/manual/closedsetword.apx`. This is an example of a closed-set identification test for children. A word (wave file) is presented and the child responds by clicking on one of four figures on the screen(figure 5.1). Subsequently, a new set of figures is shown, and again, a word corresponding to one of these figures is routed to the sound card. This is repeated 3 times. The three words are embedded in noise at a certain signal-to-noise ratio. In this example the level of the noise is fixed and the level of the word varies. At the beginning of the experiment APEX 3 queries for the SNR (signal to noise ratio, in dB). After having entered this value four figures will appear. Press Start to start the experiment and to hear the first stimulus (speech in noise). After the experiment has finished the results are written to a results file and the percentage correct is determined.

5.2.2 Conceptual

The experiment as described in the previous paragraph should first be translated to concepts understood by APEX 3. The main concepts in this example are **datablock**, **stimulus**, **screen** and **procedure**. For each of the 3 words to be presented to the subject, a wave file is available on disk. For each wave file, a datablock is defined, and for each resulting datablock a stimulus is defined. Everything that is defined is assigned an ID, to be able to refer to it later on. Therefore, now we have 3 stimuli with IDs `stimulus_star`, `stimulus_mouse` and `stimulus_fly`.

We should also define the things to be shown on the screen during the experiment. This is done by defining a screen for each word. Each screen contains 4 pictures, of which one corresponds to the word. Each screen again gets an ID, in this case we name the screen by the pictures it contains. Therefore we now have 3 screens with ID `screenstar_horse_vase_moon`, `screenknee_fly_mouse_star` and `screenmouse_fly_star_moon`.

To indicate the order in which the words should be presented, and together with which screen, a procedure is defined. In the procedure a number of trials is defined. Recall that a trial is the combination of a screen, a stimulus and a correct answer. Therefore a trial is a way to link each of our stimuli with a screen.

Now only the output logic remains to be defined. The idea is to continuously present a noise signal and to set the level of the words such that a certain SNR is obtained. To achieve this, we define 2 filters, the first one is a generator (i.e., a filter without input channels) that will generate the noise signal. The other one is an amplifier, which will amplify or attenuate the words to obtain the desired SNR.

In the following sections each of the elements of the `XML` file necessary to implement the latter concepts will be described in detail. The first one is `<procedure>`.

5.2.3 Detailed description of various elements

```
1  <procedure xsi:type="apex:constantProcedure">
2    <parameters>
3      <presentations>1</presentations>
4      <order>sequential</order>
5    </parameters>
6    <trials>
7      <trial id="trial_star">
8        <answer>picturestar</answer>
9        <screen id="screenstar_horse_vase_moon"/>
10       <stimulus id="stimulus_star"/>
11     </trial>
12     <trial id="trial_fly">
13       <answer>picturefly</answer>
14       <screen id="screenknee_fly_mouse_star"/>
15       <stimulus id="stimulus_fly"/>
16     </trial>
17     <trial id="trial_mouse">
18       <answer>picturemouse</answer>
19       <screen id="screenmouse_fly_star_moon"/>
20       <stimulus id="stimulus_mouse"/>
21     </trial>
22   </trials>
23 </procedure>
```

The attribute `xsi:type="apex:constantProcedureType"` indicates that a constant stimuli procedure is used. This means that the procedure will select the next trial from the trial list and that it completes after every trial has been presented a certain number of times.

- `<parameters>` defines the behavior of the procedure
  - `<presentations>` every trial is presented once
  - `<order> sequential` the three trials are presented sequentially, in the order that is specified in the experiment file

- `<trials>` contains different `<trial>` elements to specify a trial. After selecting a trial the `<procedure>` will show the specified screen and send the stimulus to the device. Each trial is given an ID (arbitrary name), eg `TrialStar`, such that it can be referred to later on or viewed in the results file.

- `<answer>` the correct answer, to be used by APEX 3 to determine whether the subject's response is correct. Here, the subject gets the opportunity to click on one of 4 pictures. The result from the screen (the subject's response) will be the ID of the element of the screen

that was clicked. Therefore, in this example, we specify the ID of the picture corresponding to the stimulus that is presented.

A trial must be defined for all the words of the experiment.

```
1    <corrector xsi:type="apex:isequal"/>
```

The corrector checks whether the response is correct or not. The attribute `xsi:type="apex:isequal"` compares whether the two input values are exactly the same. In this example `<corrector>` compares the answer specified under trial and the ID corresponding to the picture that has been clicked.

```
1  <screens>
2     <uri_prefix>closedset</uri_prefix>
3     <reinforcement>
4       <progressbar>true</progressbar>
5       <feedback length="600">false</feedback>
6     </reinforcement>
7     <screen id="screenstar_horse_vase_moon">
8       <gridLayout height="2" width="2">
9         <picture id="picturestar" row="1" col="1">
10          <path>star.jpg</path>
11        </picture>
12        <picture id="picturehorse" row="2" col="1">
13          <path>horse.jpg</path>
14        </picture>
15        <picture id="picturevase" row="1" col="2">
16          <path>vase.jpg</path>
17        </picture>
18        <picture id="picturemoon" row="2" col="2">
19          <path>moon.jpg</path>
20        </picture>
21           </gridLayout>
22      <buttongroup id="buttongroup1">
23        <button id="picturestar"/>
24        <button id="picturehorse"/>
25        <button id="picturevase"/>
26        <button id="picturemoon"/>
27      </buttongroup>
28      <default_answer_element>buttongroup1</default_answer_element>
29    </screen>
30    <screen id="screenknee_fly_mouse_star">
31      <gridLayout height="2" width="2">
32        <picture id="pictureknee" row="1" col="1">
33          <path>knee.jpg</path>
```

```
34        </picture>
35        <picture id="picturefly" row="2" col="1">
36          <path>fly.jpg</path>
37        </picture>
38        <picture id="picturemouse" row="1" col="2">
39          <path>mouse.jpg</path>
40        </picture>
41        <picture id="picturestar" row="2" col="2">
42          <path>star.jpg</path>
43        </picture>
44      </gridLayout>
45      <buttongroup id="buttongroup2">
46        <button id="pictureknee"/>
47        <button id="picturefly"/>
48        <button id="picturemouse"/>
49        <button id="picturestar"/>
50      </buttongroup>
51      <default_answer_element>buttongroup2</default_answer_element>
52    </screen>
53    <screen id="screenmouse_fly_star_moon">
54      <gridLayout height="2" width="2">
55        <picture id="picturemouse" row="1" col="1">
56          <path>mouse.jpg</path>
57        </picture>
58        <picture id="picturefly" row="2" col="1">
59          <path>fly.jpg</path>
60        </picture>
61        <picture id="picturestar" row="1" col="2">
62          <path>star.jpg</path>
63        </picture>
64        <picture id="picturemoon" row="2" col="2">
65          <path>moon.jpg</path>
66        </picture>
67      </gridLayout>
68      <buttongroup id="buttongroup3">
69        <button id="picturemouse"/>
70        <button id="picturefly"/>
71        <button id="picturestar"/>
72        <button id="picturemoon"/>
73      </buttongroup>
74      <default_answer_element>buttongroup3</default_answer_element>
75    </screen>
76  </screens>
```

<screens> contains several <screen> elements that can be referred to elsewhere in the experiment file (e.g., in <procedure> above).

- <uri_prefix> a relative path is specified here (relative with respect to the experiment file). Since APEX 3 knows the location of the experiment file, only the folder containing the wave files and pictures must be specified. It is also possible to give the absolute path, starting at the root. There are 3 ways to specify a prefix: by directly specifying an absolute path, by directly specifying a path relative to the experiment file or by referring to a prefix stored in apexconfig.xml. Please refer to section 3.3 for more information.

- <reinforcement> includes

  - <progressbar> As the value is true a progressbar will be displayed in the right hand bottom corner of the screen that indicates how many trials have been completed and how many remain.
  - <feedback length> duration of time after response (in msec) that APEX 3 waits before presenting the next trial. During this interval, feedback can be displayed. In this case, no feedback (thumb up, thumb down) is given as the value is false.

- <screen> For each word to be presented, a screen is defined. Each screen has an ID by which it can be referred to elsewhere in the experiment file (e.g. in <trials> to associate it with a stimulus).

  - <gridLayout> specifies how the four figures will be arranged on the screen. A GridLayout creates a regular grid on the screen with the specified number of rows and columns. In this example there are 2 rows and 2 columns. Each figure is defined by means of a <picture> element. Such a definition can be seen as associating a graphics file with an ID and specifying at what position of the layout it should be shown. In this case, jpeg files are used, but other formats are also possible (e.g., png, bmp, gif).

    * <picture>
      <uri> since the prefix as specified under <uri_prefix> is preprended to each <uri> it suffices to give the name of the .jpeg file in <path>. The path is relative to the experiment file.

  - <buttongroup> defines a group of screen elements, namely those (four figures) that are displayed on the screen. The ID is defined before.
  - <default_answer_element> As many elements can be defined in a screen, APEX 3 has no way to know which element contains the subject's response. If, for example, a text box is shown and 2 buttons, it is unclear which is to be used to determine whether the answer is correct or not. Therefore, in <default_answer_element>

the element is designated that contains the subject's response. In the case of screen elements that are clicked in order to respond, the example is further complicated by the fact that we cannot specify just one of the elements (buttons, pictures), but the response rather comes from a group of elements. This is when a <buttongroup> can be used to group together different screen elements.

In this example 5 datablocks are defined, 3 for the word files, 1 for the noise and 1 for silence.

```
<datablocks>
  <uri_prefix>closedset/</uri_prefix>
  <datablock id="datablock_star">
    <device>wavdevice</device>
    <uri>star.wav</uri>
  </datablock>
  <datablock id="datablock_fly">
    <device>wavdevice</device>
    <uri>fly.wav</uri>
  </datablock>
  <datablock id="datablock_mouse">
    <device>wavdevice</device>
    <uri>mouse.wav</uri>
  </datablock>
  <datablock id="noisedata">
    <device>wavdevice</device>
    <uri>noise.wav</uri>
  </datablock>
  <datablock id="silence">
    <device>wavdevice</device>
    <uri>silence:500</uri>
  </datablock>
</datablocks>
```

<datablocks> contains 5 <datablock> elements and a prefix.

- <uri_prefix> a relative path is specified here. It is also possible to give the absolute path, starting at the root (see section 3.3).
- <datablock> For each wave file a datablock is made, with an ID.
- Each datablock is associated to a <device> (by means of the ID of the device). In datablock silence the special syntax is demonstrated for creating a datablock containing only silence (i.e., all samples are zeros). This is done to prevent the speech and noise from starting at the same time. The length of the silence datablock is specified in ms after the prefix silence. It is added before the signal, not before the noise.

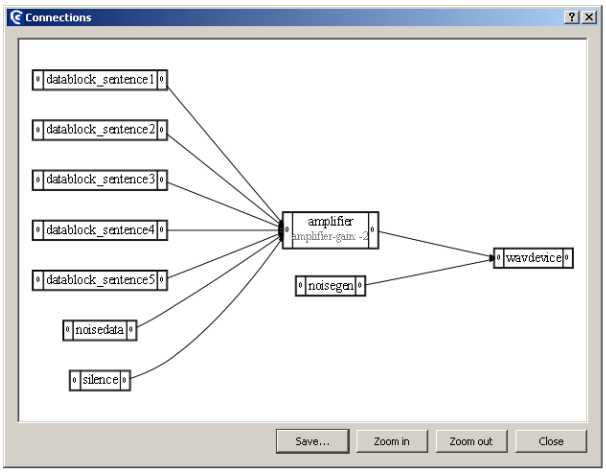

Figure 5.2: The connection window

A datablock must be made for all wave files used in the experiment, including the noise (that will be used by the noise generator).

In the next sections, the output logic will be defined. Figure 5.2 gives an overview of the building blocks that are used in this example. On the left hand side the datablocks are shown. In the middle the noise generator and the amplifier and on the right hand side the sound card. As the words are to be amplified or attenuated according to the desired SNR, the corresponding datablocks are routed through the amplifier.

In the next element the output device is specified.

```
1 <devices>
2   <device id="wavdevice" xsi:type="apex:wavDeviceType">
3     <driver>portaudio</driver>
4     <card>default</card>
5     <channels>1</channels>
6     <samplerate>44100</samplerate>
7   </device>
8 </devices>
```

All devices defined in the experiment file are grouped in `<devices>`. In this example there is only 1 `<device>` element. Its ID is set to `soundcard`. As an ID is unique for an entire experiment file, it can be used later on to refer to this device.

- The `xsi:type="apex:wavDeviceType"` attribute tells APEX 3 that a sound card is used. The experiment only starts if all devices can be opened.

- `<driver>` specifies the software driver to be used for sound output. If unsure, set it to `portaudio`.

- <card> specifies the name of the sound card to be used. The system default card can be used by specifying default as a card name. Other card names can be defined in apexconfig.
- <channels> specifies the number of channels of the sound card to be used. The number of channels is restricted to the selected sound card, with a maximum of 2 for portaudio.
- <samplerate> the sampling rate of the sound card; Not all sampling rates are supported by all devices, and some drivers automatically convert the sampling rate. Check your sound card documentation. The sample rate of the sound card should correspond to the sampling rates of all datablocks used with it. If not, an error message will be shown.

Filters must be defined whenever the signal (or noise) is manipulated. In this example the level of the noise remains constant and the signal is amplified or attenuated using an amplifier filter (loop of <datablock>).

```
1  <filters>
2     <filter xsi:type="apex:dataloop" id="noisegen">
3       <device>wavdevice</device>
4       <channels>1</channels>
5       <continuous>false</continuous>
6       <datablock>noisedata</datablock>
7       <basegain>-5</basegain>
8       <gain id="noisegain">0</gain>
9       <randomjump>true</randomjump>
10    </filter>
11    <filter xsi:type="apex:amplifier" id="amplifier">
12      <device>wavdevice</device>
13      <channels>1</channels>
14      <basegain>-5</basegain>
15      <gain id="gain">0</gain>
16    </filter>
17  </filters>
```

<filters> contains individual <filter> elements, which specify a filter, or as a special case a generator (i.e., a filter without input channels).

- <filter> on line 2 the attribute xsi:type="apex:dataloop" tells APEX 3 that a dataloop generator has to be created. This is a generator that takes a datablock and loops it infinitely. The datablock to be looped is specified by its ID noisedata. The dataloop generator itself is assigned the ID noisegen.
- <filter> on line 11 the attribute xsi:type="apex:amplifier" tells APEX 3 that an amplifier is to be created. The gain of this amplifier will be varied to change the amplitude of the words and thus the SNR. It is assigned ID amplifier. The gain of the amplifier is made a variable parameter by assigning it ID gain

- <device> The device with which the filter is associated
- <channels> The number of channels of the filter. The available number of channels is dependent on the type of filter. An amplifier can have any number of channels.
- If <continuous> is set to true, the filter or generator will keep on running in between two trials (i.e., when the subject is responding). In this example it stops when the stimulus stops playing (false).
- <datablock> The datablock with ID noisedata, specified under <datablocks> will be looped.
- <basegain> the total gain of the amplifier is basegain+gain. Basegain cannot be a parameter, gain can be a parameter. The total gain of the complete output system should not be larger than 0 to avoid clipping of the signal. This is why basegain = -5.
- <gain> extent to which the signal is amplified.
- If <randomjump> is set to true, when the dataloop is started, it will jump to a random sample in the datablock. Thereafter it is looped.

```
<stimuli>
   <stimulus id="stimulus_star">
     <datablocks>
       <sequential>
         <datablock id="silence"/>
         <datablock id="datablock_star"/>
         <datablock id="silence"/>
       </sequential>
     </datablocks>
   </stimulus>
   <stimulus id="stimulus_fly">
     <datablocks>
       <sequential>
         <datablock id="silence"/>
         <datablock id="datablock_fly"/>
         <datablock id="silence"/>
       </sequential>
     </datablocks>
   </stimulus>
   <stimulus id="stimulus_mouse">
     <datablocks>
       <sequential>
         <datablock id="silence"/>
         <datablock id="datablock_mouse"/>
         <datablock id="silence"/>
       </sequential>
```

```
27      </datablocks>
28    </stimulus>
29  </stimuli>
```

<stimuli> contains different <stimulus>, each with an ID <stimulus>

- <datablocks> can be combined in <sequential> order (as opposed to <simultaneous>.

This is repeated for all the stimuli in the experiment.

```
1  <connections>
2     <connection>
3       <from>
4         <id>_ALL_</id>
5         <channel>0</channel>
6       </from>
7       <to>
8         <id>amplifier</id>
9         <channel>0</channel>
10      </to>
11    </connection>
12    <connection>
13      <from>
14        <id>amplifier</id>
15        <channel>0</channel>
16      </from>
17      <to>
18        <id>wavdevice</id>
19        <channel>0</channel>
20      </to>
21    </connection>
22    <connection>
23      <from>
24        <id>noisegen</id>
25        <channel>0</channel>
26      </from>
27      <to>
28        <id>wavdevice</id>
29        <channel>0</channel>
30      </to>
31    </connection>
32  </connections>
```

<connections> defines how the different datablocks and filters are routed to the output device. The ID _ALL_ stands for all the datablocks. In this example they are routed to the first

channel of the filter with ID amplifier (defined under <filters>). In the amplifier the signal is amplified or attenuated and sent to the wavdevice on lines 12 to 21. At the same time the noise, generated by a generator with ID noisegen, is sent to the same channel of the wavdevice. The channels are numbered from 0 onwards. The level of the noise remains constant and does not pass through an amplifier (lines 22 to 31).

A visual representation of connections can be obtained by choosing "Show stimulus connections" under "Experiment"in the main APEX 3 menu (top left menu bar).

idn1 xslt is outdated, we now use javascript results

```
1 <results>
2   <xsltscript>idn.xsl</xsltscript>
3 </results>
```

Even if <results> is not specified in the experiment file APEX 3 will deliver a results file in XML.

- <xsltscript> a script can transform the XML data to an easier readable format; this script can be stored in folder scripts under APEX 3. For more information please read section 7.3.4.

```
1 <interactive>
2   <entry type="int" description="SNR␣in␣dB"
3    expression="/apex:apex/filters/filter[@id='amplifier']/gain"
4    default="0"/>
5 </interactive>
```

If a small part of an experiment file has to be changed right before the start of an experiment (e.g. a start value, a gain value, the subject's name), APEX 3 can show the experimenter a small graphical user interface (GUI) containing the elements to be changed. This is accomplished by defining the <interactive> element in the experiment file.

In this example we will modify the gain of the filter with ID amplifier to a value that is entered by the experimenter at the start of the experiment.

It is only possible to change the value of an existing element of the experiment file, elements cannot be added. For each element to be changed, an <entry> should be defined under <interactive>. <entry> has four attributes that should be defined:

- type specifies the type of input element that will be shown. If it is int, a spinbox[1] will be shown. If it is string a plain text box will be shown. In this case a spinbox will be shown as a gain is always numeric.

- description defines the text to be shown in the GUI next to the input element, such that the experimenter knows exactly what to fill in.

[1] A spinbox is an input field that only accepts numbers and has 2 buttons to respectively increase or decrease its value

- `expression` defines the element of the experiment file to be changed. It is specified by a so-called XPath expression [2]. For a description of XPath, we refer to the according standard or a good XML book.

  OxygenXML can generate XPath expressions for you. First point the cursor to the target element, then click the right mouse button and select. An XPath expression to the clicked location is now in the clipboard and can be pasted at the appropriate place using the paste function.

- `default` specifies the default value to be shown in the input element.

the elements below are now under the <results> tag

```
1 <general>
2     <showresults>true</showresults>
3     <saveprocessedresults>true</saveprocessedresults>
4 </general>
```

`<general>` defines some general parameters. Saving processed results in a results file is only possible if a XSLT-script is defined. See section 7.3.4.

- `<showresults>` If `true` a window will appear after completion of the experiment asking whether the results should be shown on screen.

- `<saveprocessedresults>` If `true` the experimenter will be asked whether the processed results must be appended to the results file.

[2] see http://www.w3.org/TR/xpath

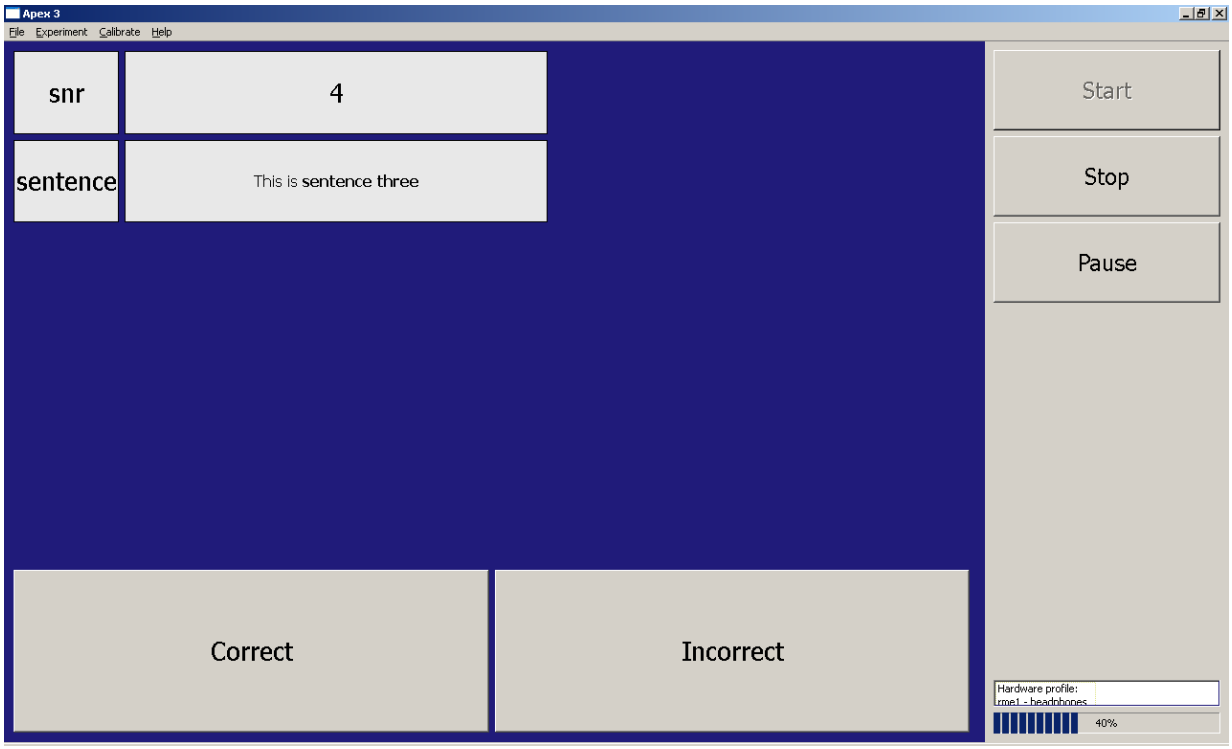

Figure 5.3: Example of adaptive sentence in noise experiment

5.3 Example 2: Identification of speech in noise using an adaptive method

5.3.1 General description of the experiment

See `examples/manual/sentenceinnoise.apx`. This is an example of an adaptive speech in noise test. It determines the speech reception threshold (SRT), the 50 percent correct point, using the 1-down, 1-up method described by **?**. In this adaptive procedure the first sentence is repeated with increasing level until it is identified correctly. Subsequently, the SRT is determined by increasing or decreasing the level in steps of 2 dB, according to the last response. Other decision procedures (eg 2-down 1-up) can also be implemented using APEX 3. In this example the 5 sentences are scored on the basis of their keywords. The keywords are indicated in bold on the screen. The experimenter/clinician is seated in front of the screen and decides whether the subject has repeated the keywords (of the sentence) correctly, after which the correct or incorrect button is clicked(figure 5.3). No feedback is provided. The starting level is given in signal-to noise ratio (the level of the noise remains the same, that of the signal varies). In this example speech and noise are routed to the same channel, i.e. one speaker or one earpiece of the headphone. The results are written to a results file.

5.3.2 Conceptual

The experiment as described in the previous paragraph should first be translated to concepts understood by APEX 3. The main concepts in this example are **datablock**, **stimulus**, **screen**, **procedure**, a **variable** parameter (to change the gain) and **fixed Parameter** to show a sentence on the screen. For each sentence a datablock is defined, and for each resulting datablock a stimulus is defined. As always, everything that is defined, is assigned an ID, to be able to refer to it later on. In this example there are 5 stimuli with IDs `stimulus_sentence1`, `stimulus_sentence2`, `stimulus_sentence3`, `stimulus_sentence4` and `stimulus_senter`
The procedure defines a number of trials. Recall that a trial is the combination of a screen (always the same in this example), a stimulus and a response.

As we are dealing with an adaptive procedure a fixed or variable parameter is adapted. In this example a variable parameter is adapted to change the gain of the sentence, and a fixed parameter is used to show a sentence on the screen. The screen also shows the signal-to-noise ratio (SNR) under test and the response alternatives "correct" and "incorrect".

Now only the output logic remains to be defined. The idea is to continuously present a noise signal and to vary the level of the sentences adaptively. To achieve this, we define 2 filters, the first one is a generator (i.e., a filter without input channels) that will generate the noise signal. The other one is an amplifier, which will amplify or attenuate the sentences to obtain the desired SNR. Both are connected to one channel of the wavdevice.

In the following sections each of the elements of the XML file necessary to implement the latter concepts will be described in detail.

5.3.3 Detailed description of various elements

```
1  <procedure xsi:type="apex:adaptiveProcedure">
2    <parameters>
3      <presentations>1</presentations>
4      <order>sequential</order>
5      <nUp>1</nUp>
6      <nDown>1</nDown>
7      <adapt_parameter>gain</adapt_parameter>
8      <start_value>0</start_value>
9      <stop_after_type>presentations</stop_after_type>
10     <stop_after>1</stop_after>
11     <rev_for_mean>4</rev_for_mean>
12     <larger_is_easier>true</larger_is_easier>
13     <repeat_first_until_correct>true</repeat_first_until_correct>
14     <stepsizes><stepsize begin="0" size="2"/></stepsizes>
15   </parameters>
16
17   <trials>
18     <trial id="trial_sentence1">
```

```
19          <answer>button_correct</answer>
20          <screen id="screen"/>
21          <stimulus id="stimulus_sentence1"/>
22      </trial>
23
24      <trial id="trial_sentence2">
25          <answer>button_correct</answer>
26          <screen id="screen"/>
27          <stimulus id="stimulus_sentence2"/>
28      </trial>
29
30      <trial id="trial_sentence3">
31          <answer>button_correct</answer>
32          <screen id="screen"/>
33          <stimulus id="stimulus_sentence3"/>
34      </trial>
35
36      <trial id="trial_sentence4">
37          <answer>button_correct</answer>
38          <screen id="screen"/>
39          <stimulus id="stimulus_sentence4"/>
40      </trial>
41
42      <trial id="trial_sentence5">
43          <answer>button_correct</answer>
44          <screen id="screen"/>
45          <stimulus id="stimulus_sentence5"/>
46      </trial>
47    </trials>
48  </procedure>
```

The attribute `xsi:type="adaptiveProcedureType"` of `<procedure>` refers to a procedure in which a parameter is changed according to the response of the subject. In this example the gain of the amplifier is adapted. The procedure selects the next trial from the trial list and it completes after every trial has been presented a certain number of times.

- `<parameters>` defines the behavior of the procedure
  - `<presentations>` every trial is presented once
  - `<order>` the trials are presented sequentially, in the order that is specified in the experiment file (if `random` would be specified, they would be presented in random order).
  - `<nUp>` the level is increased after n (here 1) incorrect response(s); cf `<larger_is_easier>`

  - <nDown> the level is decreased after n (here 1) correct response(s); cf <larger_is_easier>
  - <adapt_parameter> to be adapted
  - <start_value> the experiment starts with gain=0 (input of user). The value here will be replaced by the entry value. Please refer to section 3.1.11 for more information.
  - <stop_after_type> the experiment stops after a specified number of presentations of each trial is completed (it is also possible to stop after <reversals>.
  - <stop_after> the experiment stops after 1 presentation
  - <rev_for_mean> the average is determined on the basis of a certain number of presentations (or reversals if <reversals> is specified in <stop_after_type>).
  - <larger_is_easier> If true, then larger values of the parameter are easier than smaller values. It is used to determine <nUp> and <nDown>.
  - <repeat_first_until_correct> the first trial is repeated with increasing gain until it is identified correctly.
  - <stepsizes> from the beginning of the experiment (begin=0) the stepsize is 2 dB.

outdated, now parameter for results script

- <trials> contains different <trial> elements to specify a trial. Once a trial is selected, <procedure> will show the specified screen and send the stimulus to the device. Each trial is given an ID (arbitrary name), eg trialsentence1, such that it can be referred to later on or viewed in the results file. A trial must be defined for all the sentences of the experiment.

- <answer> the correct answer, to be used by APEX 3 to determine whether the subject's response is correct. In this example the experimenter will click on "correct" or "incorrect". The result from the screen will be the ID of the element of the screen that was clicked (button_correct or button_incorrect).

```
1 <corrector xsi:type="apex:isequal" />
```

The corrector checks whether the response is correct or not. In this example <corrector> compares the answer specified under trial (button_correct) with the response given by the subject (either button_correct or button_incorrect).

```
1 <screens>
2       <reinforcement>
3          <progressbar>true</progressbar>
4          <feedback length="600">false</feedback>
5       </reinforcement>
6
7       <screen id="screen">
```

```
8       <gridLayout height="2" width="1" id="main_layout" rowstretch="1,2">
9             <gridLayout width="4" height="4" columnstretch="1,4,2,2"
10            rowstretch="1,1,2,2" id="parameter_layout" row="1" col="1">
11
12    <label id="snrlabel" row="1" col="1">
13    <text>snr</text>
14    </label>
15
16    <parameterlabel id="snr" row="2" col="1">
17     <parameter>gain</parameter>
18    </parameterlabel>
19
20    <label id="sentence" row="1" col="2">
21    <text>sentence</text>
22    </label>
23
24    <parameterlabel id="sentence" row="2" col="2">
25    <fontsize>12</fontsize>
26    <parameter>sentence</parameter>
27    </parameterlabel>
28
29    </gridLayout>
30
31    <gridLayout height="1" width="2" id="answer_layout" x="1" y="2">
32    <button x="1" y="1" id="button_correct">
33    <text>Correct</text>
34     </button>
35    <button x="2" y="1" id="button_wrong">
36    <text>Incorrect</text>
37    </button>
38
39    </gridLayout>
40    </gridLayout>
41    <buttongroup id="buttongroup">
42    <button id="button_correct"/>
43    <button id="button_wrong"/>
44    </buttongroup>
45    <default_answer_element>buttongroup</default_answer_element>
46    </screen>
47    </screens>>
```

<screens> contains <screen> element that is referred to in <procedure>.

- <reinforcement> includes elements on progress and feedback

  - <progressbar> If the value is true a progress bar will be displayed in the right hand bottom corner of the screen that indicates how many trials have been completed and how many remain.
  - <feedback length> duration of the time after response (in msec) that APEX 3 waits before presenting the next trial. During this interval feedback can be displayed. In this example, no feedback (thumb up, thumb down) is given as the value is false.

- <screen> the screen has an ID by which it can be referred to in each trial. In this example the screen displays four blocks in the top left corner to indicate the SNR and the sentence. In addition, the labels "Correct" and "Incorrect" are displayed on the buttons at the bottom of the screen (cfr screenshot, figure...).
  - <gridlayout> places elements in an irregular grid. The screen is divided into sections according to the values. In this example there are equal number of rows and columns. The stretch factor for the columns is a list of integers, separated by comma's. There should be as many integers as columns. The width of the columns is proportional to the numbers. With width="1", and height="2" rowstretch="1,2" implies that the second row will be twice as wide as the first one. If width="4", and height="4" and columnstretch="1,4,2,2" the second column will be four times as wide as the first and two times as wide as the 3rd and 4th. With columnstretch="1,1,2,2" the 3rd and 4th rows will be twice as wide as the 1st and 2nd.
  - <label> the labels on the left are fixed and display the <text> "SNR" and "Sentence". The button "Correct" and "Incorrect" are indicated at the bottom of the screen
  - <parameterlabel> the blocks on the right the values of the fixed parameters. Gain is the (variable) SNR level, and sentence is a fixed parameter, defined in <stimulus>
  - <font size> can be specified in points
  - <buttongroup> defines a group of screen elements (those that are displayed on the screen). As many elements can be defined in a screen, APEX 3 has no way to know which element contains the subject's response. Therefore, in <default_answer_eleme the element is designated that contains the subject's response. In the case of screen elements that are clicked in order to respond, the example is further complicated by the fact that we cannot specify just one of the elements (buttons, pictures), but that the response rather comes from a group of elements. This is when a <buttongroup> can be used to group together different screen elements.

```
1 <datablocks>
2   <uri_prefix>sentences/</uri_prefix>
3   <datablock id="datablock_sentence1">
4     <device>wavdevice</device>
5     <uri>sent1.wav</uri>
6   </datablock>
```

```
7   <datablock id="datablock_sentence2">
8     <device>wavdevice</device>
9     <uri>sent2.wav</uri>
10  </datablock>
11  <datablock id="datablock_sentence3">
12    <device>wavdevice</device>
13    <uri>sent3.wav</uri>
14  </datablock>
15  <datablock id="datablock_sentence4">
16    <device>wavdevice</device>
17    <uri>sent4.wav</uri>
18  </datablock>
19  <datablock id="datablock_sentence5">
20    <device>wavdevice</device>
21    <uri>sent5.wav</uri>
22  </datablock>
23  <datablock id="noisedata">
24    <device>wavdevice</device>
25    <uri>noise.wav</uri>
26  </datablock>
27  <datablock id="silence">
28    <device>wavdevice</device>
29    <uri>silence:500</uri>
30  </datablock>
31 </datablocks>
```

<datablocks> contains a list of <datablock> elements and a prefix.

- <uri_prefix> a relative path is specified here (relative with respect to the experiment file). Since APEX 3 knows the location of the experiment file, only the folder containing the wave files and pictures must be specified. It is also possible to give the absolute path, starting at the root. There are 3 ways to specify a prefix: by directly specifying an absolute path, by directly specifying a path relative to the experiment file or by referring to a prefix stored in apexconfig.xml. Please refer to section 3.3 for more information.

- <datablock> for each wave file a datablock is made, with an ID.

  Each datablock is associated to a <device> (by means of its ID). In datablock silence the special syntax is demonstrated for creating a datablock containing only silence (i.e., zeros). This is done to put silence before and after the sentence (not the noise), to prevent the speech and noise from starting at the same time. The length of the silence datablock is specified in ms after the prefix silence.

  - the datablock is associated with the sound card with ID wavdevice, as defined in the <devices> section.

- <uri> the URI is appended to the prefix defined above

```
<devices>
   <device id="wavdevice" xsi:type="apex:wavDeviceType">
   <driver>portaudio</driver>
   <card>default</card>
   <channels>1</channels>
   <samplerate>44100</samplerate>
   </device>
</devices>
```

All devices defined in the experiment file are grouped in <devices>.The ID is set to wavdevice. As an ID is unique for a entire experiment file, it can be used later on to refer to this device (eg in datablocks).

- <device> the xsi:type="apex:wavDeviceType" attribute tells APEX 3 that a sound card is used. The experiment only starts if all devices can be opened.
- <driver> specifies the software driver to be used for sound output. If unsure, set it to portaudio.
- <card> specifies the name of the sound card to be used. The system default card can be used by specifying default as a card name. Other card names can be defined in apexconfig.xml.
- <samplerate> the sampling frequency of the sound card. Not all sampling rates are supported by all devices, and some drivers automatically convert the sampling rate. Check your sound card documentation. The sample rate of the sound card should correspond to the sampling rates of all <datablocks> used with it.

```
<filters>
<filter xsi:type="apex:dataloop" id="noisegen">
   <device>wavdevice</device>
   <channels>1</channels>
   <continuous>false</continuous>
   <datablock>noisedata</datablock>
   <basegain>-5</basegain>
   <gain>0</gain>
   <randomjump>true</randomjump>
   </filter>
<filter xsi:type="apex:amplifier" id="amplifier" >
   <device>wavdevice</device>
   <channels>1</channels>
   <basegain>-5</basegain>
```

```
15      <gain id="gain">0</gain>
16  </filter>
17  </filters>
```

<filters> contains individual <filter> elements, which specify a filter, or as a special case a generator (i.e., a filter without input channels).

- The attribute xsi:type="apex:dataloop" tells APEX 3 that a dataloop generator has to be created. This is a generator that takes a datablock and loops it infinitely. The datablock to be looped is specified by its ID noisedata.
- <filter> The attribute xsi:type="apex:amplifier" tells APEX 3 that it is assigned an ID amplifier. The gain of this amplifier will be varied to change the amplitude of the words and thus the SNR. The gain of the amplifier is made a variable parameter by assigning it ID gain
  - <device> the device with which the filter is associated
  - <channels> The number of channels of the filter. The available number of channels depends on the type of filter. An amplifier can have any number of channels.
  - <continuous> If set to false the noise is presented during the speech, but not during the subject's response.
  - <datablock> the datablock with ID noisedata, specified under <datablocks> will be looped.
  - <basegain>: the total gain of the amplifier is basegain+gain. The total gain of the complete output system should not be larger than 0 to avoid clipping of the signal. This is why basegain = -5.
  - <gain> the gain value that is changed. E.g. if the targetamplitude is 65 dB SPL (see Calibration) the signal and noise have equal amplitude if gain = 0. Gain is changed by other modules.
  - If <randomjump> is set to true, when the dataloop is started, it will jump to a random sample in the datablock. Thereafter it is looped.

```
1   <stimuli>
2     <fixed_parameters>
3       <parameter id="sentence"/>
4     </fixed_parameters>
5
6     <stimulus id="stimulus_sentence1">
7       <datablocks>
8       <sequential>
9          <datablock id="silence"/>
```

```
10        <datablock id="datablock_sentence1"/>
11        <datablock id="silence"/>
12      </sequential>
13      </datablocks>
14
15    <fixedParameters>
16      <parameter id="sentence">This is <b> sentence </b> <b>one</b></paramet
17    </fixedParameters>
18    </stimulus>
19
20    <stimulus id="stimulus_sentence2">
21      <datablocks>
22      <sequential>
23        <datablock id="silence"/>
24        <datablock id="datablock_sentence2"/>
25        <datablock id="silence"/>
26      </sequential>
27      </datablocks>
28
29    <fixedParameters>
30      <parameter id="sentence">This is <b> sentence </b> <b>two</b></paramet
31      </fixedParameters> </stimulus>
32
33    <stimulus id="stimulus_sentence3">
34      <datablocks>
35      <sequential>
36        <datablock id="silence"/>
37        <datablock id="datablock_sentence3"/>
38        <datablock id="silence"/>
39      </sequential>
40      </datablocks>
41
42  <fixedParameters>
43  <parameter id="sentence">This is <b> sentence </b> <b>three</b></paramete
44  </fixedParameters>
45  </stimulus>
46
47  <stimulus id="stimulus_sentence4">
48      <datablocks>
49      <sequential>
50        <datablock id="silence"/>
51        <datablock id="datablock_sentence4"/>
52        <datablock id="silence"/>
```

```
53     </sequential>
54     </datablocks>
55
56  <fixedParameters>
57  <parameter id="sentence">This is <b> sentence </b> <b>four</b></parameter
58  </fixedParameters>
59  </stimulus>
60
61  <stimulus id="stimulus_sentence5">
62     <datablocks>
63     <sequential>
64        <datablock id="silence"/>
65        <datablock id="datablock_sentence5"/>
66        <datablock id="silence"/>
67     </sequential>
68     </datablocks>
69
70    <fixedParameters> <parameter id="sentence">This is <b> sentence
71    </b> <b>five</b> </parameter> </fixedParameters> </stimulus>
72  </stimuli>
```

<stimuli> contains different <stimulus>, each with an ID.

- <fixedParameters> is used here to be able to show the sentence under test on the Screen. Fixed parameters are discussed in section 3.2.
  - <parameter> the fixed parameter is defined here and should also be defined in each <stimulus>
- <stimulus> Each stimulus gets an ID (referred to in <trial>)
  - <datablocks> here one refers to the <datablock> described before; it includes the sentence file.
    - <sequential> The sequence of <datablocks> is indicated here.
  - <fixedParameters> sets the fixed parameter for each <stimulus>.
  - <b> indicates which words (the keywords) should appear in bold on screen.

This is repeated for all the sentences.

```
1  <connections>
2     <connection>
3        <from>
4        <id>_ALL_</id>
```

```
5         <channel>0</channel>
6         </from>
7         <to>
8         <id>amplifier</id>
9         <channel>0</channel>
10        </to>
11    </connection>
12    <connection>
13        <from>
14        <id>amplifier</id>
15        <channel>0</channel>
16        </from>
17        <to>
18        <id>wavdevice</id>
19        <channel>0</channel>
20        </to>
21    </connection>
22    <connection>
23        <from>
24        <id>noisegen</id>
25        <channel>0</channel>
26        </from>
27        <to>
28        <id>wavdevice</id>
29        <channel>0</channel>
30        </to>
31        </connection>
32 </connections>
```

- `<connections>` defines how the different datablocks and filters are routed to the output device. The ID `_ALL_` stands for all the datablocks. In this example they are routed to the first channel of the filter with ID amplifier (defined under `<filters>`). In the amplifier the signal is amplified or attenuated and sent to the wavdevice on lines 12 to 21. At the same time the noise, generated by a generator with ID noisegen, is sent to the same channel of the wavdevice. The channels are numbered from 0 onwards. The level of the noise remains constant and does not pass through an amplifier (lines 22 to 31).Although noisedata is a `<datablock>` and connected to *amplifier* it is not specified in `<stimulus>` and does not pass through *amplifier*.

  A visual representation of connections can be obtained by choosing “Show stimulus connections” under “Experiment”in the main APEX 3 menu (top left menu bar).

outdated

```
<results>
   <xsltscript>mao.xsl</xsltscript>
</results>
```

<results> Even if <results> is not specified in the experiment file APEX 3 will save a results file in XML.

- <xsltscript> a script can transform the XML data to an easier readable format; this script can be stored in folder scripts under APEX 3. For more information please read section 7.3.4.

```
<interactive> <entry type="int" description="SNR␣start␣value"
 expression="/apex:apex/procedure/parameters/start_value" default="4" />
</interactive>
```

If a small part of an experiment file has to be changed right before the start of an experiment (e.g. a start value, a gain value, the subject's name), APEX 3 can shown the experimenter a small GUI containing the elements to be changed. This is accomplished by defining the <interactive> element in the experiment file.

In this example we will modify the gain of the filter with ID amplifier to a value that is entered by the experimenter at the start of the experiment.

- <entry type> a value is entered, here corresponding to SNR in dB. It is also possible to enter a string, e.g. name of the subject (see Reference Manual).
- <expression> the interactive window will show 4 as default value; more than one entry may be defined.

```
<general>
  <showresults>true</showresults>
  <saveprocessedresults>true</saveprocessedresults>
</general>
```

<general> defines some general parameters. Saving **processed** results in a results file is only possible if a XSLT-script is defined. See section 7.3.4.

- <showresults> If true a window will appear after completion of the experiment asking whether the results should be viewed on screen. The sequence of responses will be shown, as well as the average SNR (based on the number of reversals specified before).
- <saveprocessedresults> If true the experimenter will be asked whether the processed results must be saved.

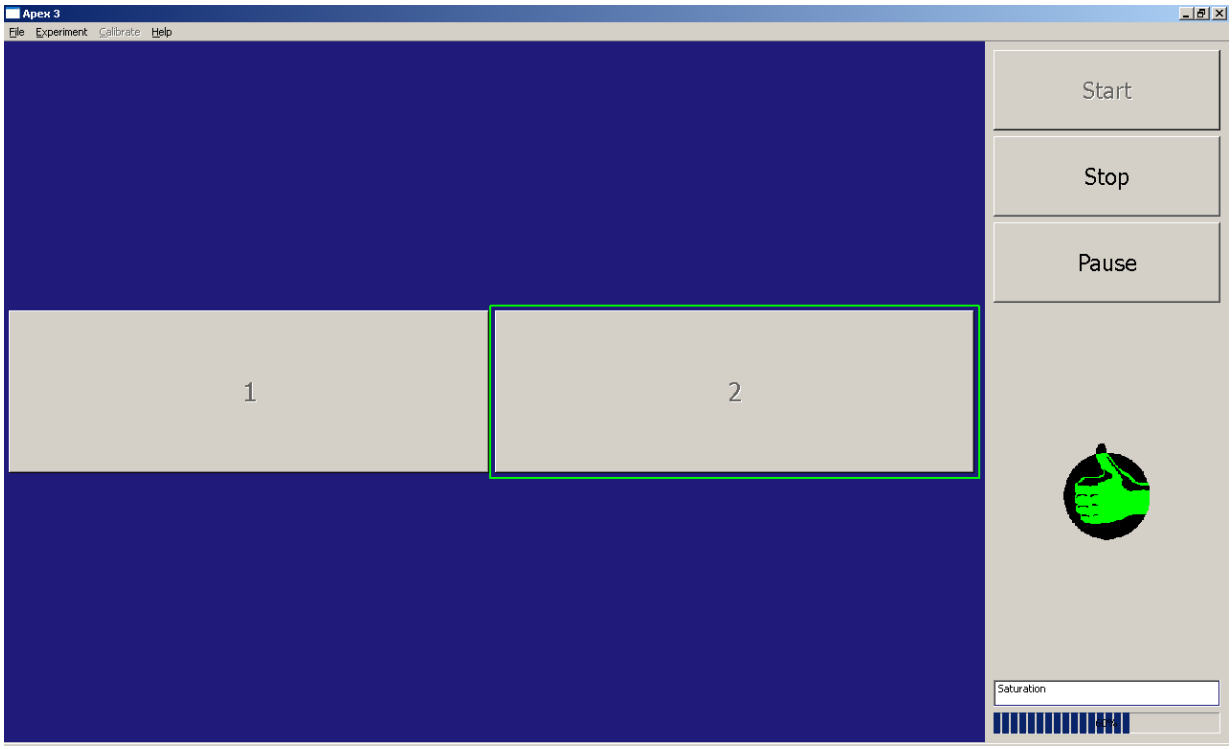

Figure 5.4: Example of adaptive gap detection experiment

5.4 Example 3: Gap detection: determining the Just noticeable difference

5.4.1 General description of the experiment

See `examples/manual/gapdetection.apx`. This is an example of a gap detection task: two stimuli are presented to the listener in random order, a stationary noise and an interrupted noise. During the presentation the intervals are highlighted. The subjects' task is to indicate the interval with the interruption(figure 5.4). Feedback is provided (thumb up, thumb down) and the minimal detectable gap is determined by means of an adaptive procedure (here 2-down, 1-up). Stimuli (wave files) are generated off line. This example only includes 5 wave files. Usually many more are generated. The results are written to a text file.

5.4.2 Conceptual

The experiment as described in the previous paragraph should first be translated to concepts understood by APEX 3. The main concepts used here are **procedure**, **screens**, **datablocks** and **stimuli**.

For each wave file (noise with or without a gap) a `<datablocks>` is defined, and for each datablock a `<stimulus>` is defined. In this example there are 5 wave files, with gap sizes

ranging between 1 and 5 ms. Their IDs are `g5ms`, `g4ms`, `g3ms`,`g2ms` and `g1ms`. In an adaptive procedure either a fixed or variable parameter is defined. In this example a fixed parameter is used, i.e. the gap size in the stimulus. The wave files with different gap sizes are stored on disk, and are assigned a certain gap value. This gap value is used to determine the gap threshold. Also, a `<screen>` is defined that shows the two response intervals indicating "1" and "2".

To indicate the order in which the sentences should be presented a `<procedure>` is defined. An adaptive procedure is defined containing 1 `<trial>` with five variable stimuli (with gap) and the standard stimulus without the gap. Recall that a `<trial>` is the combination of a `<screen>` (always the same in this example), a stimulus and a correct answer. The standard and stimuli occur randomly in one of both intervals.

Now only the output logic remains to be defined. The idea is to present that standard and a variable trial after each other, and the stimulus to be presented depends on the response of the subject. Filters are not used since the signals are not changed.

In the following sections each of the elements of the XML file necessary to implement the latter concepts will be described in detail.

5.4.3 Detailed description of various elements

```
1  <procedure xsi:type="apex:adaptiveProcedure">
2    <parameters>
3      <presentations>1</presentations>
4      <order>sequential</order>
5      <choices select="1,2">2</choices>
6      <pause_between_stimuli>1000</pause_between_stimuli>
7      <nUp>1</nUp>
8      <nDown>2</nDown>
9      <adapt_parameter>gap</adapt_parameter>
10     <start_value>5</start_value>
11     <stop_after_type>reversals</stop_after_type>
12     <stop_after>5</stop_after>
13     <min_value>0</min_value>
14     <max_value>5</max_value>
15     <rev_for_mean>3</rev_for_mean>
16     <larger_is_easier>true</larger_is_easier>
17     <stepsizes>
18       <change_after>reversals</change_after>
19       <stepsize begin="0" size="2"/>
20       <stepsize begin="2" size="1"/>
21     </stepsizes>
22   </parameters>
23
24   <trials>
```

```
25      <trial id="trial1" >
26       <screen id="screen1" />
27       <stimulus id="stimulus1" />
28       <stimulus id="stimulus2"/>
29       <stimulus id="stimulus3"/>
30       <stimulus id="stimulus4"/>
31       <stimulus id="stimulus5"/>
32       <standard id="standard"/>
33      </trial>
34    </trials>
35  </procedure>
```

<procedure> the attribute The attribute xsi:type="adaptiveProcedureType" of <procedure> refers to a procedure in which a parameter is changed according to the response of the subject. In this example 1 trial is defined with different stimuli from which the adaptive procedure chooses.

- <parameters> defines the behavior of the procedure (eg., nr of presentations, order of presentation, response strategy)
  - <presentation> every trial is presented once.
  - <order> it applies to the order of the trials, here <sequential> (if random would be specified, the trials would be presented in random order). Since there is only 1 <trial>, it does not matter here wether the order is specified as <sequential> or <random>.
  - <choices> number of alternatives to choose from (here: 2 intervals)
  - <pause_between_stimuli> a pause of 1000 ms will be introduced between successive wave files;
  - <nUp> number of items after incorrect trials to increase the gap
  - <nDown> number of items after correct trials to decrease the gap
  - <min_value> the gap size cannot be smaller than 0.
  - <max_value> the gap size cannot be larger than 5
  - <adapt_parameter> the parameter to change, here the "gap", see also <fixed_parameter
  - <start_value> the gap at which to start, here 5, see under <stimulus>
  - <stop_after_type> reversals (it can also stop after a number of presentations or trials)
  - <stop_after> number of reversals after which the procedure is stopped
  - <rev_for_mean> the number of reversals that are used for the average threshold
  - <larger_is_easier> here true (the smaller the gap, the more difficult the task)

  - <stepsizes> the stepsize
  - <change_after> the <stepsize> is changed after a specified number of reversals. In this example a stimulus is skipped until the second reversal (start at 5, then 3, etc). Thereafter no stimulus is skipped.

- <trials> only 1 trial is defined
  - <trial> the ID of the trial
  - <screen> the ID of the screen
  - <stimulus> the ID of the stimulus
  - <standard> the ID of the standard

outdated: now belongs under procedure/parameters

```
<corrector xsi:type="apex:alternatives">
 <answers>
  <answer number="1" value="button1"/>
  <answer number="2" value="button2"/>
 </answers>
</corrector>
```

- <corrector> the corrector checks whether the response is correct or not. In this example the number of choices in <procedure> was 2. This means that <procedure> will present the target stimulus in either interval 1 or 2. <procedure> informs <correct> about the correct interval. Corrector also receives the clicked button from the screen and looks up the corresponding number in the <answers> element above and compares it with the number it received from <procedure>.
  - <answer>: the <answer> allocates an interval to a button (on the screen)

```
<screens>
  <reinforcement>
    <progressbar>true</progressbar>
    <feedback length="300">true</feedback>
    <showcurrent>true</showcurrent>
  </reinforcement>

  <screen id="screen1" >
    <gridLayout height="1" width="2" id="main_layout">
      <button x="1" y="1" id="button1" >
       <text>1</text>
```

```
12      </button>
13      <button x="2" y="1" id="button2" >
14       <text>2</text>
15      </button>
16    </gridLayout>
17   <buttongroup id="buttongroup1">
18     <button id="button1"/>
19     <button id="button2"/>
20   </buttongroup>
21     <default_answer_element>buttongroup1</default_answer_element>
22  </screen>
23 </screens>
```

- <screens> contains several screen elements that can be referred to elsewhere in the experiment file (e.g. in <procedure> above).
  - <reinforcement> includes elements on progress and feedback
    - <progressbar> if true a progress bar will be displayed in the right hand bottom corner of the screen. The progress bar can indicate how many trials have been done or it shows when a reversal occurs in an adaptive procedure. In the latter case the progress bar will increase at every reversal while the number of trials varies.
    - <feedback length> duration of the time after response (in msec) that APEX 3 waits before presenting the next trial. During this interval feedback can be displayed.
    - <showcurrent> the interval is highlighted while a signal is presented.
  - <screen> each screen has an ID by which it can be referred to elsewhere in the experiment. In this example the screen displays two intervals.
    - <gridlayout>places elements in an irregular grid. The screen is divided into sections according to the values. In this example there is an equal number of rows (x) and columns (y).
      Gridlayout defines a group of screen elements (those that are displayed on the screen). As many elements can be defined in a screen, APEX 3 has no way to know which element contains the subject's response. Therefore, in <default_answer_el
      the element is designated that contains the subject's response. In the case of screen elements that are clicked in order to respond, the example is further complicated by the fact that we cannot specify just one of the elements (buttons, pictures), but that the response rather comes from a group of elements.
      - <button> for each interval a button is specified; this button (interval) displays a number on the screen
      - <text> the left interval denotes "1", the right one denotes "2".

- <buttongroup> This element defines a group of buttons, namely those two interval that need to be displayed on the screen. The same name (ID) is used as defined before.
  - * <default_answer_element> the response that has been clicked by the subject is sent to the corrector. It is related to the ID of <buttongroup>.

```
<datablocks>
  <uri_prefix>Gapfiles</uri_prefix>
    <datablock id="g5ms" >
      <device>wavdevice</device>
      <uri>g5.wav</uri>
    </datablock>

    <datablock id="g4ms" >
      <device>wavdevice</device>
      <uri>g4.wav</uri>
    </datablock>

    <datablock id="g3ms" >
      <device>wavdevice</device>
      <uri>g3.wav</uri>
    </datablock>

    <datablock id="g2ms" >
      <device>wavdevice</device>
      <uri>g2.wav</uri>
    </datablock>

    <datablock id="g1ms" >
      <device>wavdevice</device>
      <uri>g1.wav</uri>
    </datablock>

    <datablock id="datablockref">
      <device>wavdevice</device>
      <uri>ref500.wav</uri>
    </datablock>
  </datablocks>
```

<datablocks> contains a list of <datablock> elements and a prefix.

- <uri_prefix>: a relative path is specified here. Since APEX 3 knows the location of the experiment file, only the folder containing the wave files and pictures must be specified.

- <datablock> for each wave file a unique datablock is defined by an ID. The standard signal (uninterrupted noise) must also be specified here.
- <device> each datablock includes the audio device to which the signal is routed
- <uri> since the path is defined in <uri_prefix> it is not necessary to specify the entire path again

```
1 <devices>
2  <device id="wavdevice" xsi:type="apex:wavDeviceType">
3  <driver>portaudio</driver>
4  <card>default</card>
5  <channels>2</channels>
6  <gain>0</gain>
7  <samplerate>44100</samplerate>
8  </device>
9 </devices>
```

<devices> all devices defined in the experiment file are grouped in the <devices>. Only 1 <device> is used in this example. The ID is set to soundcard. As an ID is unique for a entire experiment file, it can be used later on to refer to this device.

- <device> the xsi:type="apex:wavDeviceType" attribute tells APEX 3 that a sound card is used. The experiment only starts if all devices can be opened.
- <driver> specifies the software driver to be used for sound output. If unsure, set it to portaudio.
- <card> specifies the name of the sound card to be used. The system default card can be used by specifying default as a card name. Other card names can be defined in ApexConfig.
- <channels> 2 channels are specified here, because the signal is stereo.
- <samplerate> the sampling frequency of the wave files. Not all sampling rates are supported by all devices, and some drivers automatically convert the sampling rate. Check your sound card documentation. The sample rate of the sound card should correspond to the sampling rates of all datablocks used with it.

```
1 <stimuli>
2   <fixed_parameters>
3     <parameter id="gap"/>
4   </fixed_parameters>
5
6   <stimulus id="stimulus1" >
```

```
7       <description>noisewithgap1</description>
8       <datablocks>
9         <datablock id="g5ms" />
10      </datablocks>
11      <fixedParameters>
12        <parameter id="gap">5</parameter>
13      </fixedParameters>
14    </stimulus>
15
16    <stimulus id="stimulus2" >
17      <description>noisewithgap2</description>
18      <datablocks>
19        <datablock id="g4ms" />
20      </datablocks>
21        <fixedParameters>
22         <parameter id="gap">4</parameter>
23      </fixedParameters>
24    </stimulus>
25
26    <stimulus id="stimulus3" >
27      <description>noisewithgap3</description>
28      <datablocks>
29        <datablock id="g3ms" />
30      </datablocks>
31       <fixedParameters>
32        <parameter id="gap">3</parameter>
33       </fixedParameters>
34    </stimulus>
35
36    <stimulus id="stimulus4" >
37      <description>noisewithgap4</description>
38      <datablocks>
39        <datablock id="g2ms" />
40      </datablocks>
41        <fixedParameters>
42         <parameter id="gap">2</parameter>
43        </fixedParameters>
44    </stimulus>
45
46    <stimulus id="stimulus5" >
47      <description>noisewithgap5</description>
48      <datablocks>
49        <datablock id="g1ms" />
```

```
50      </datablocks>
51        <fixedParameters>
52          <parameter id="gap">1</parameter>
53        </fixedParameters>
54    </stimulus>
55
56    <stimulus id="standard">
57      <datablocks>
58       <datablock id="datablockref"/>
59      </datablocks>
60        <fixedParameters>
61         <parameter id="gap">0</parameter>
62      </fixedParameters>
63    </stimulus>
64  </stimuli>
```

<stimuli> defines the auditory event, e.g. noise with gap and noise without gap.

- <fixedParameters> the gap is a fixed parameter that is identified by an ID
  - <parameter> the gap is a fixed parameter that is identified by an ID
- <stimulus> this element includes a description of the (variable) stimulus
  - <description>
  - <datablocks> the ID refers to the
  - <fixedParameter>
    * <parameter> includes information on the size of the variable gap

```
1   <connections>
2     <connection>
3       <from>
4         <id>_ALL_</id>
5         <channel>0</channel>
6       </from>
7       <to>
8         <id>wavdevice</id>
9         <channel>1</channel>
10      </to>
11    </connection>
12
13   </connections>
```

- <connections> defines how the different datablocks and filters are routed to the output device. The ID _ALL_ stands for all the datablocks. In this example they are routed to 1 channel of the wavdevice.

  A visual representation of connections can be obtained by choosing "Show stimulus connections" under "Experiment"in the main APEX 3 menu (top left menu bar).

outdated

```
1 <results>
2   <xsltscript>mao.xsl</xsltscript>
3 </results>
```

<Results> transforms the results using an XSLT script. Even if <results> is not specified in the experiment file APEX 3 will deliver a results file in XML. The results file will display the correct answers, the reversals, the entire sequence of responses, and the average threshold based on the number of reversals and the magnitude of the corresponding gap parameter specified in the experiment file.

- <xsltscript> a script can transform the xml data to an easier readable format; this script can be stored in folder scripts under APEX 3. For more information please read section(7.3.4)

```
1  <general>
2    <exitafter>true</exitafter>
3    <showresults>true</showresults>
4    <saveprocessedresults>true</saveprocessedresults>
5   </general>
```

outdated

<general> defines some general parameters. Saving **processed** results in a results file is only possible if a XSLT-script is defined. See section(7.3.4).

- <show results> if true a window will appear after completion of the experiment asking whether the results should be viewed on screen.

- <saveprocessedresults> boolean. If true the experimenter will be asked whether the results must be saved in a results file.

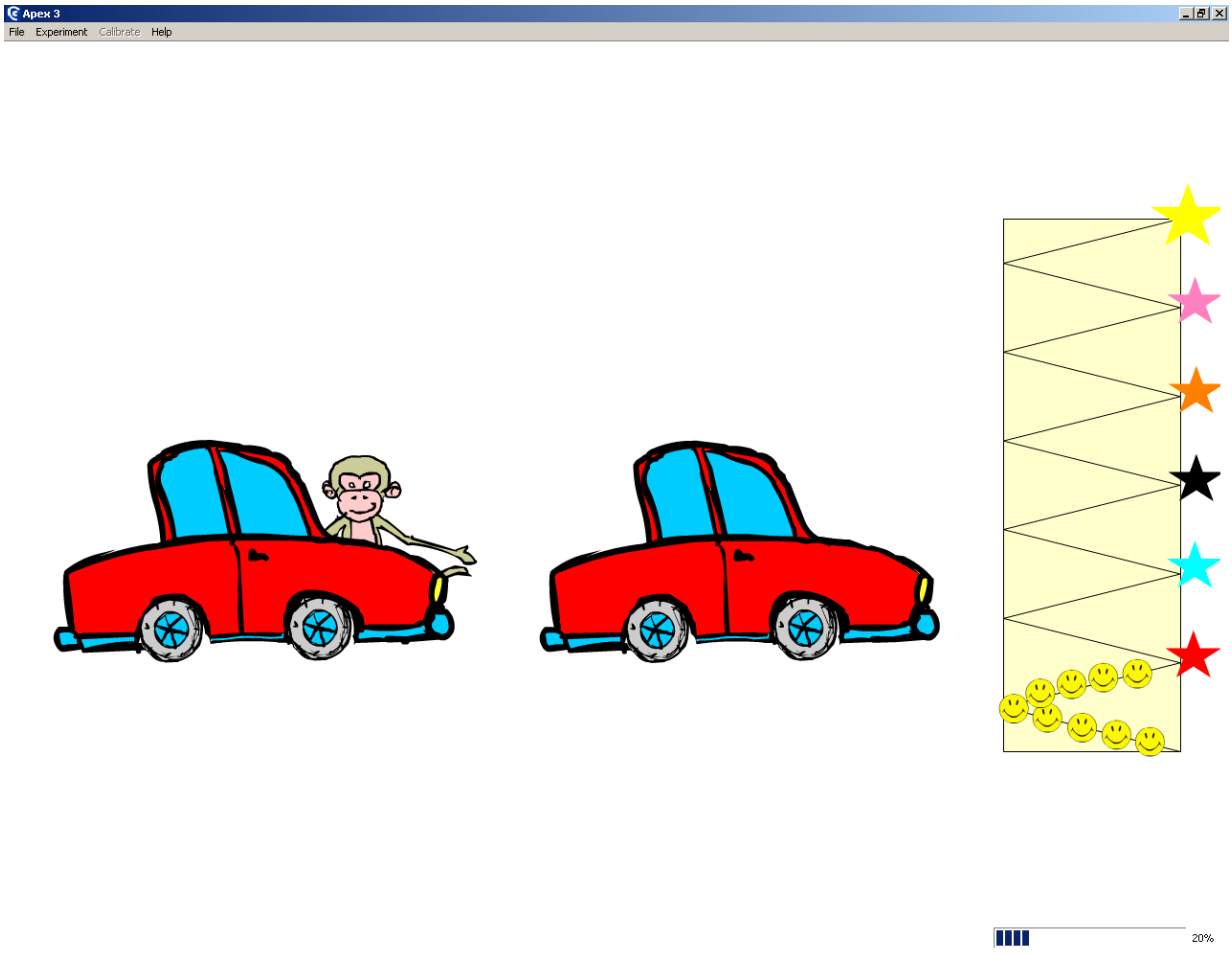

Figure 5.5: Example of gap detection in child mode

5.5 Example 4: Gap detection in child mode

5.5.1 General description of the experiment

See `examples/manual/gapdetectionchild.apx`. This is an example of a gap detection task adapted to the interest of children: two stimuli are presented to the listener in random order, a stationary noise and an interrupted noise. It is the same experiment as example 3, but pictures (movies) of cars replace buttons on the screen, and a smiley panel is shown(figure 5.5). During the presentation of stimuli the cars are animated. The child's task is to indicate the stimulus with the interruption. Feedback is provided by smileys and the minimal detectable gap is determined by means of an adaptive procedure (here 2-down, 1-up). Only those elements that are specific to the child mode are described in this example.

5.5.2 Conceptual

The main concepts illustrated in this example are **childmode**, and **reinforcement**. Child mode involves a different panel, and intro/outro movies.

5.5.3 Detailed description of various elements

```
<screens>
   <uri_prefix>car/</uri_prefix>
   <general>
     <showpanel>true</showpanel>
   </general>
   <reinforcement>
     <progressbar>true</progressbar>
     <feedback length="300">true</feedback>
     <showcurrent>true</showcurrent>
   </reinforcement>
   <childmode>
     <panel>reinforcement.swf</panel>
   </childmode>
   <screen id="screen1">
     <gridLayout height="1" width="2" id="main_layout">
       <flash row="1" col="1" id="button1">
         <path>stillcar.swf</path>
         <feedback>
           <highlight>rijden.swf</highlight>
           <positive>goodcar.swf</positive>
           <negative>badcar.swf</negative>
         </feedback>
       </flash>

       <flash row="1" col="2" id="button2">
         <path>stillcar.swf</path>
         <feedback>
           <highlight>rijden.swf</highlight>
           <positive>goodcar.swf</positive>
           <negative>badcar.swf</negative>
         </feedback>
       </flash>
     </gridLayout>

     <buttongroup id="buttongroup1">
       <button id="button1"/>
```

```
37        <button id="button2"/>
38      </buttongroup>
39      <default_answer_element>buttongroup1</default_answer_element>
40    </screen>
41  </screens>
```

<screens>

- <uri_prefix> a relative path is specified here (relative with respect to the experiment file). This only applies to the movies in <screen>. There are 3 ways to specify a prefix: by directly specifying an absolute path, by directly specifying a path relative to the experiment file or by referring to a prefix stored in apexconfig.xml. Please refer to section 3.3 for more information.

- <general>

  - <show panel> if true a panel will be shown with smileys (see figure **presented above**)

- <reinforcement>

  - <progressbar> If true a progress bar will be displayed on the right hand side with smileys. The progress bar shows when a reversal occurs in an adaptive procedure (while the number of trials varies).
  - <feedback length> duration of the time after response (in msec) that APEX 3 waits before presenting the next trial. During this interval feedback can be displayed
  - <showcurrent> the interval is highlighted while a signal is presented

- <childmode> replaces the "standard" panel, sets some defaults, enables intro/outro movies and changes the background colour

  - <panel> the name of the movie file, per frame (smileys and panel are a collection of frames)

<screen> Each screen has an ID by which it can be referred to elsewhere in the experiment. In this example the screen displays two movies, each containing a car.

- <gridlayout> places elements in an irregular grid. The screen is divided into sections according to the values. In this example there are equal number of rows and columns.

  - <flash> replaces <button> of the standard version
    * <path> the name of the flash movie file on disk, when there is no animation.
    * <feedback>
    * <highlight>the name of the file while the car moves

    * <positive> the name of the movie file following a correct response (here a monkey waving)
    * <negative> the name of the movie file following an incorrect response (the trunk of an elephant)

- <buttongroup> defines a group of screen elements (those that are displayed on the screen). As many elements can be defined in a screen, APEX 3 has no way to know which element contains the subject's response. Therefore, in <default_answer_element> the element is designated that contains the subject's response. In the case of screen elements that are clicked in order to respond, the example is further complicated by the fact that we cannot specify just one of the elements (buttons, pictures), but that the response rather comes from a group of elements. This is when a <buttongroup> can be used to group together different screen elements.

Chapter 6

Example strategies

6.1 N-AFC procedures

<choices>

6.2 Roving

randomgenerator

6.3 Roving another stimulus dimension

6.4 Identification by typing a sentence

See examples/manual/opensetidentification.apx. This is an example of an open set identification task: the subject is required to type (or repeat) the word or sentence that has been presented(figure 6.1). The results are written to a text file.

In order to achieve this it is necessary to adapt <trial>, <answer>, and <screens>.

```
1  <trial id="trial_maan">
2       <answer>moon</answer>
3       <screen id="screen"/>
4       <stimulus id="stimulus_star"/>
5     </trial>
```

```
1    <screens>
2    <reinforcement>
```

Figure 6.1: Example of an open set identification experiment

```
3       <progressbar>true</progressbar>
4       <feedback length="600">false</feedback>
5     </reinforcement>
6     <screen id="screen" >
7       <gridLayout height="2" width="1" id="main_layout">
8         <label x="1" y="1" id="helplabel">
9           <text> Type the word you hear</text>
10        </label>
11        <textEdit x="1" y="2" id="text" >
12        </textEdit>
13      </gridLayout>
14      <default_answer_element>word</default_answer_element>
15    </screen>
16  </screens>
```

6.5 Practice

See `examples/manual/trainprocedure.apx`. Here the subject clicks on an interval and a stimulus is presented. This can be done as often as he/she wishes.

To achieve this `<procedure xsi:type="apex:trainingProcedureType">` should be used.

6.6 Higlighting

example threshold procedure, 2 interval, 1 tone

alle soorten van highlighting (theoretisch deel)

6.6.1 Interleaving procedures

multiprocedure

cmd line parameter

- procedure
- corrector
- ...

Chapter 7

Displaying and analysing results

7.1 The results XML file

After completion of the experiment a file containing results is always given. The default results file is in the XML format and it contains all the information about the course of the completed experiment. In the following sections, we see how to display the results on the screen in a user-friendly format and how to import them into other software for further analysis.

APEX 3 automatically assigns a default name to the results file, namely it appends "-apr" to the name of the experiment file (e.g. closedsetword-apr). It will never overwrite an existing results file, but will append a number to results (e.g. closedsetword-apr-1) in the case of an existing results file.

The results are stored in the element

```
1 <apex:results>
2
3 </apex:results>
```

The XML file contains <general> information, such as

- the testing date
- the testing duration
- the name of the XSLT script file (see 7.3.4).
- information on the procedure (eg: the adaptive parameter)

In addition, for each completed <trial> presented to the subject it includes

- details of the procedure (stimulus that was presented and the values of the variable parameters in the specific trial).
- the response that was chosen by the subject

- the outcome of the corrector
- possible errors of the output system/device (eg. underruns)
- the response time (time between the moment the buttons are enabled and the moment an answer is given)
- in case a random generator was used: the value of the random generator in the specific trial

Remark: with an adaptive procedure the number of <reversals> is given

7.2 Displaying results

APEX has infrastructure to show experiment results on the screen, directly after the experiment, or by opening the results file afterwards with APEX. The system is based on Javascript and HTML, which makes it easy to modify for the end user.

explain what HTML is

In the <results> element of the experiment file, you can define the HTML file to be used to display results, e.g.,

```
1  <results>
2     <page>apex:resultsviewer.html</page>
3     <showduringexperiment>true</showduringexperiment>
4     <showafterexperiment>true</showafterexperiment>
5  </results>
```

<page> can either refer to a file in the same folder as the experiment file, e.g., <page>myresults.htm
to a folder somewhere on disk, e.g., <page>/C/users/tom/myresults.html</page>,
or to a file in the APEX resultsviewer folder, e.g., <page>apex:specialresultsviewer.html</pag

If <showduringexperiment> is true, the resultsviewer will be shown while the experiment is running, and will be updated after each trial. If <showafterexperiment> is true, it will be shown after the experiment has finished.

7.2.1 The results HTML file

If you want to make small changes to the way results are shown, the best way to start is to copy resultsviewer.html to the same folder where your experiment file is stored, rename it something sensible, change the reference in your experiment file, and then modify the HTML file according to the desired result.

give example

Currently the following divs are available:

```
1  <div id="procedureparameterplot" style="width:600px;height:300px;"></div
2    <br>
3    <div id="procedureparametervalues"></div>
```

```
4    <div id="procedureparameterlastvalue"></div>
5    <div id="procedureparameterreversals"></div>
6    <div id="procedureparametermeanrevs"></div>
7    <div id="procedureparametermeantrials"></div>
8    <div id="procedureparametertable"></div>
9
10
11   <div id="confusionmatrixplot"></div>
12   <div id="confusionmatrixtable"><table></table></div>
13   <div id="confusionmatrixsummary"></div>
```

document each div, automatically extract from resultviewer.l documentation?

7.2.2 The internals - APEX

If you want to change more than the basic screen layout, you need to change or add some javascript code. In what follows, the internals will be explained. First the APEX side will be explained: how the results viewer gets the actual data from APEX. Next resultsviewer.html and the associated javscript code will be explained.

When results are to be viewed (depending on `<showduringexperiment>` and `<showafterexperi` APEX will load the results HTML file in a basic web browser (called QWebView, based on WebKit). Then, every time a trial is finished, it will call a javascript function `newAnswer`, with as argument a string containing the XML that would normally be written to the results file. For example, after a trial, the following Javascript code cold be executed:

```
1  newAnswer("<trial␣id=\"trial1\">\n<procedure␣type=\"apex:adaptiveProcedu
```

What happens next, fully depends on the HTML/Javascript code.

Whenever new results should be displayed, APEX will call the `plot` function in Javascript. Note that before each plot any number of calls to newAnswer can occur.

7.2.3 The internals - `resultsviewer.html`

The resultsviewer implements the `newAnswer` and a number of plotting functions that can be used to display results. In resultsviewer.html itself, the plot() function needs to be implemented, which can refer to any of the plotting functions defined in the provided javascript libraries, e.g., the following code will show a confusion matrix if a constant procedure was used, and a staircase of the adaptive parameter otherwise:

```
1  <script type="text/javascript">
2    $(document).ready(function(){
3
4    });
5
```

```
6   function plot() {
7     if ( containsAdaptive(results.xml[0])) {
8       plot_rtprocedureparameter();
9     } else {
10      plot_rtconfusionmatrix();
11    }
12  }
13 </script>
```

`Resultsviewer.html` can include the following javascript libraries:

rtresults.js Implements `newAnswer` and collects the results in variable `results`, which has the following members (all Arrays): answers, correctanswers, parametervalues, xml, trials, stimuli, standards results . `rtresults.js` does not contain any plotting functions. If you want to do some extra processing when newAnswer is called, you can implement `extraNewAnswer`.

explain each member

rtconfusionmatrix.js implements `plot_rtconfusionmatrix`, which will calculate and plot a confusion matrix.

rtlocalisation.js implements `plot_rtlocalisation`, which adds extra functionality on top of `rtconfusionmatrix.js`, specifically for localisation experiments. It will parse the stimulus ID to extract the angle of incidence (by implementation of `extraNewAnswer`), and show extra metrics such as the RMS ans absolute localisation error.

rtprocedureparameter.js implements `plot_rtprocedureparameter` which will plot the staircase of an adaptive procedure and show a number of performance metrics, such as the mean of the last N trials and of the last N reversals. To set the value of N, you need to define the variable reversalsForMean, e.g.,

```
1 \item[rtprocedureparameter.js] implements \function{plot\_rtprocedure
  you need to define the variable reversalsForMean, e.g.,
2 \begin{lstlisting}
3 $(document).ready(function(){
4   reversalsForMean=8;}
5   );
```

rtpsignifit.js implements `extraPlot`, which will be automatically called from the plot functions above when this file is loaded. It demonstrates the use of psignifit to estimate psychometric function parameters [**??**].

Add example of how to make a small change

7.3 Exporting results

While you could simply copy-paste the relevant information from the results XML file into the desired format, this is labour intensive and error prone. Therefore several options are provided to convert the XML file in a more user friendly format.

The first option, `<saveprocessedresults>`, will append a section to the results file containing the essential data in comma separated values (CSV) format, which can be easily copied to another program.

A next option, XSLT transforms, uses the XSLT programming language to convert the results XML file into another format, which could be text, XML, HTML, etc.

If you conduct your analysis in Matlab or R, the most efficient option will be to use the APEX Matlab or R toolbox, and import your result XML file directly. In the R toolbox, a function is also provided to convert a series of results XML files into one main CSV file.

7.3.1 saveprocessedresults

If `<saveprocessedresults>` is `true` in the experiment file the processed data will be appended to the XML file under `<processed>`

The results file contains a lot of information. A summary of the relevant results can be obtained through an XSLT transform (see next paragraph).

add example of output

7.3.2 Using the APEX Matlab Toolbox

7.3.3 Using the APEX R Toolbox

refer to AMT section, add teaser here

7.3.4 XSLT transforms

An XSLT script transforms the results XML file to a summary of the results. An XSLT script is provided with APEX (`apexresult.xsl`), which can do the most common transformations. You can also create your own scripts, provided you have some XSLT knowledge. Please note that we are phasing out the use of XSLT in APEX, in favour of HTML resultsviewer and the APEX Matlab and R toolbox. Therefore the `apexresult.xsl` is no longer maintained, and we advise against using it in new projects.

To execute an XSLT transform, a transformation scenario has to be configured once. This is done by clicking on the “Configure transformation scenario” button in Oxygen. If you then click on “new”, you can search for the appropriate XSLT

script in `xslt/` and give it a name in the upper field of the "edit scenario" box. If you then click on "transform now", the processed results are shown immediately.

The processed results can be saved in the results XML file and/or shown immediately after a test (in the last case the program asks whether you want to see them or not).

Appendix A

Using cochlear implants from Cochlear

To directly stimulate subjects using cochlear implants from Cochlear, APEX 3 uses the NICv2 interface, provided by Cochlear. This interface is abstracted as the L34Device (to be used under `<devices>`). It is always called L34, irrespective of the device that is used with NICv2 (which can be an L34 but also an SP12 or other devices).

A.1 L34 setup

To use APEX 3 with the L34 device, you need the L34 plugin, which is provided on request, provided you sign the Cochlear NIC agreement.

After obtaining the L34 plugin (in the form of the file `l34plugin.dll`), copy it to the `plugins` directory under the main APEX directory. You also need a properly setup Cochlear NICv2 environment. We refer to the NICv2 documentation on how to do this. For APEX 3 it boils down to having the right version of NICv2 (the same APEX 3 was compiled against) and having the Nucleus NIC binaries directory in your path. If this part is setup correctly, you should be able to use the L34 device with device number 0 (the simulated device).

Before starting an APEX 3 experiment that uses the L34device, make sure the appropriate devices are connected to the computer and the NIC environment is setup correctly.

A.2 Electrical stimulation files

When using the L34 device, you need prefined stimulation patterns. Electrical stimulation patterns are not stored in the wave format, such as for the wavdevice, but as one of the following file types:

replace qic by aseq

qic In a qic file, 2 parameters are specified per stimulus: the channel to be used and the magnitude (in %). Before a qic file can be sent to the NICv2 interface, it thus first has to be mapped by apex.

qicext electrode, magnitude (CU), period, phasewidth, phasegap

xml defined by cochlear (ref?)

Either of these files can be generated using the Nucleus Matlab Toolbox, provided by cochlear. The most important difference between xml stimulation files and other simulation files, is that for the latter APEX 3 will do channel mapping using a user map defined in the experiment file.

A.2.1 Generating .qic files

In `.qic` files, pulses are defined by a certain magnitude (between 0 and 1) and a channel. They can be generated by the Matlab function `genstimulus_elec.m` and written to disk using the `save_sequence.m` function. A .qic file can be read back into matlab using the `read_qic.m` function.

As the period is constant for an entire qic file, powerup frames have to be inserted in the file if no stimulation is desired at a certain instant.

A.3 Parameters of the L34 device

To use the L34 device, define a `<device>` under `<devices>`, use the `xsi:type="apex:L34DeviceTy` attribute.

The L34 device has some main parameters:

device_id The device number to be used, as defined by Cochlear. E.g., the first L34 connected to the computer has id 1. There is a simulated device with id 0, which allows to debug experiments without having an L34 connected. Note that triggering does not work properly with the simulated device.

implant The implant can be either `cic3` (Esprit3G or Sprint or Freedom with old internal part) or `cic4` (Freedom with new internal part).

When using `.qic` or `.qicext` files, APEX 3 calculates the number of current units on the basis of the defined map. The map can be either entered directly in the experiment file (inline) or derived from the Nucleus fitting software R126 (fromR126). The following map parameters should be defined: number of electrodes, mode, pulsewidth, pulsegap, period (1/total rate) and the C's and T's for the different channels.

Example:

```
1 <devices>
2  <device id="l34" xsi:type="apex:L34DeviceType">
3    <device_id>1</device_id>
4    <implant>cic3</implant>
5
6    <defaultmap>
```

```
7     <inline>
8     <number_electrodes>3</number_electrodes>
9     <mode>MP1+2</mode>
10    <pulsewidth>25</pulsewidth>
11    <pulsegap>8</pulsegap>
12    <period>138.9</period>
13    <channel number="1" electrode="22" threshold="120" comfort="150"/>
14    ...
15    <channel number="22" electrode="1" threshold="130" comfort="152"/>
16    </inline>
17  </defaultmap>
18 </device>
19 <devices>
```

Note that Cochlear numbers the electrodes from 22 (apex) to 1 (base). You can use either convention in your `.qic` file, as long as the above map is specified correctly.

Note also that when using stimulation files in the XML format, the map is not used.

A.4 Bilateral stimulation

For bilateral stimulation, two devices with each a map need to be defined. To control the order of starting both devices, you can specify a master device. If a master device is specified, this will be the last device started. If not specified, the devices are started in order of appearance in the experiment file. The device that sends a trigger signal will typically be the master device.

The `<trigger>` element can be used to specify trigger signals to be sent or received. The `<trigger>out</trigger>` specifies a trigger out signal to be sent and the `<trigger>in</trigger>` tells the device to wait for a trigger signal before starting stimulation.

Example:

```
1  <master>l34-1</master>
2  <device id="l34-1" xsi:type="apex:L34DeviceType">
3    <device_id>1</device_id>
4    <trigger>out</trigger>
5    ...
6  </device>
7  <device id="l34-2" xsi:type="apex:L34DeviceType">
8    <device_id>2</device_id>
9    <trigger>in</trigger>
10   ...
11 </device>
```

Note that datablocks are attached to a device, and need to be defined twice if they are to be used with both devices.

Appendix B

Using the Matlab toolbox

Appendix C

Plugins

C.1 Introduction

Apex supports custom audio filter plugins. In this way, Apex can easily be extended without modifying Apex itself. A plugin is essentially an object that provides a processing function that accepts a block of samples and returns a processed block of samples.

We will illustrate the process of creating a plugin by implementing an amplifier plugin. Note that Apex already contains an amplifier filter, making this example less useful in practice.

Next to the aforementioned processing function, a filter can also contain parameters. Parameter values can be specified in the experiment xml file or set by another apex module.

C.2 Solving problems

using the plugin dialog

C.3 Example: amplifier

C.3.1 The experiment file

Our example amplifier looks as follows in the experiment xml file:

```
<filter id="myAmplifier" xsi:type="apex:pluginfilter">
    <device>wavdevice</device>
    <channels>1</channels>
    <continuous>false</continuous>
    <plugin>amplifierplugin</plugin>
    <parameter id="gain" name="gain">-10</parameter>
</filter>
```

plugin Is the name of the binary plugin file you provide. If no extention is specified, Apex will guess one based on the current operating system.[1]

Parameter is defined for each parameter that the plugin accepts. On initialisation the plugin will be queried for each parameter whether it is valid. The `id` attribute is the parameter id to be referenced from elsewhere in the experiment file. The name is what the plugin uses to determine which internal parameter to modify.

C.3.2 Build environment

To successfully build a plugin, you need the following things setup properly in your build environment:

- A modern compiler, preferably the same as used for compiling Apex[2]
- The Qt libraries and build tools `http://trolltech.com/developer/downloads/qt/index`
- The file pluginfilterinterface.h
- Optional: the amplifier plugin example

C.3.3 Creating the plugin

The main steps for creating and using a plugin are:

- Create a header file (.h) for your plugin, defining classes inheriting from PluginFilterInterface and PluginFilterCreator. Include pluginfilterinterface.h in this header.
- Create a source files (.cpp) implementing all methods for your plugin.
- Create a .pro file for use with qmake to compile your plugin
- Compile your plugin using qmake and make
- Copy your plugin library to the Apex plugins directory
- Create an apex experiment file that refers to your plugin library
- Run Apex with the experiment file

[1] On Linux, Apex tries to add the extention `.so` and the prefix `lib`. On Windows, Apex tries to add the extention `.dll`.

[2] On Linux we currently use gcc 4.1, on Windows we use Visual studio 2003.

The actual plugin is created by using the file `pluginfilterinterface.h` and inheriting from the PluginFilterInterface and PluginFilterCreator objects.

The only purpose of the PluginFilterCreator object is to return an instance of your filter. In this way, it is possible to have multiple filters in one library. This is currently not implemented in Apex but only provided in the interface.

The easiest way to implement your own filter, is to start from the amplifier example. We suggest that you use qmake to process a .pro file as follows:

```
TEMPLATE = lib

TARGET = amplifierplugin
CONFIG += plugin
CONFIG += debug

QT -= gui

INCLUDEPATH += /path/to/pluginfilterinterface.h

SOURCES += amplifierplugin.cpp
HEADERS += amplifierplugin.h
```

The most important part about the plugin .h file itself is to inherit from QObject and include the necessary macro's.

```
 class AmplifierPluginCreator : public QObject, public PluginFilterCreato
{
    Q_OBJECT
    Q_INTERFACES (PluginFilterCreator)

    ...
```

In the .cpp file you should include the following:

```
Q_EXPORT_PLUGIN2(amplifierplugin, AmplifierPluginCreator)
```

Then run qmake and make. Make sure that Apex can find your plugin library by putting it in the Apex plugins directory or by providing an absolute or relative path in the experiment xml file.

Note that Apex might crash if your plugin crashes (i.e., overwrites memory it should not, etc.).

If you want to know more about the general plugin mechanism, consult the Qt documentation on QPluginLoader and the Qt Plugin Howto ("The Lower-Level API: Extending Qt Applications").

Appendix D

Customizing appearance

The appearance of virtually every element on the screen can be customized. On a small scale, every element on screen can be customized by using tags such as bgcolor or fontsize per element, but on a broader scale every element can be customized by adding a style element and specifying a style sheet to be used.

Internally the Qt Stylesheet mechanism is used. For a complete overview on how this works, we refer to the Qt manual, which can also be found on the internet at http://doc.trolltech.com/4.3/stylesheet.html.

Basically, a style sheet consists of a *selector* and a series of properties that can be set.

```
1 QPushButton#button1 {
2    background-color: red;
3 }
```

Here the selector is on line 1 and the property background-color is set to red.

A selector again consists of two parts: the type of elements that will be affected and the name of the concrete element to select. Here the type is QPushButton and the name is button1. The latter name would be what you specify as an id in the experiment file.

All elements used in Apex are listed in table D.1. Several generic names of element are given in table D.2.

In Apex, stylesheets can be specified at 3 levels: inside the <**screens**> element in the <**apex_style**> element: if placed here, every element on screen will be affected, including the panel buttons and the main menu. If places in the <**style**> element, the stylesheet will affect all screens, but not the panel or menu's. If placed in a specific screen element, only that element will be affected.

The color names that can be used, are those specified in the css standard. A list can be found at http://www.w3schools.com/css/css_colornames.asp.

Element name	Screen element	Description
QPushButton	button	clickable button
QLabel	label	text label
	picture	picture

Table D.1: Screen elements for use in style sheets.

Name	Type	Description
background	QWidget	The background of the screen
startbutton	QPushButton	The start button

Table D.2: Screen element names for use in style sheets.