Guinea Pig: System Manual

The goal of the **Guinea Pig** project is to build a system that automates the running of psychoacoustical listening tests. A variety of different test types will be implemented including such as *AB-test*, *ABC-test*, ...

For further information, see System Manual: Overview

Manual Index

Last modified: Sun Aug 1 18:57:56 EEST 1999

Guinea Pig - Overview

guinea pig n

1: a small stout-bodied short-eared nearly tailless domesticated rodent (*Cavia cobaya*) often kept as a pet and widely used in biological research **2:** a subject of scientific research, experimentation, or testing

- *Webster*

The goal of the **GuineaPig** project is to build a system that makes easier to create psychoacoustical listening test by taking care of some tedious routine tasks like creation of playlists. System will also automate the running of these tests.

Traditional procedure when making listening tests is such that the tester edits and records the whole test on a DAT-tape that is then played to the test subject. The test subjects listen to the test and mark their answers on paper. The tester then has to enter the answers manually to a computer to analyze them. **GuineaPig** eliminates the need to edit the whole test into a tape beforehand and removes the manual entering of the results. Also the test doesn’t need to be the same for all subjects. The order, in which the individual test items consisting of sound samples are presented to the subject, can be very easily arranged to be different for each session.

The first phase is the *test creation* where the *tester* selects the test type, used sound samples and other test parameters. As the result *configuration files* are generated that contain the parameters needed in the test.

The test is then *run* for each test subject and results are saved. After the tests have been performed the results can be studied, *processed* and analyzed with some statistical programs.

Test types

A variety of test types have been implemented including:

- **A/B**: Test in which the test subject chooses one out of two samples played to him/her.
- **A/B/X**: Test in which the sample X is the same sample as A or B. Subject chooses which one sample X is.
- **A/B/C**: Test in which A is the reference sample and the subject chooses how much samples B and C (one of which is the same sample as A) resemble the reference sample A. (Actually this test is referred as a Ref/A/B-test).
- **A/B Scale**: An A/B test in which subject gives an answer for both samples on a scale specified by the test creator.
- **A/B Scale, Fixed Reference**: A test in which subject gives an answer how sample compares to a reference.
- **A/B Scale, Hidden Reference**: A test in which subject gives an answer for both samples on a scale specified by the test creator. One of the samples A or B is the hidden reference.
- **Single Stimulus**: A Test in which a single stimulus signal is played and then graded.
- **TAFC** (Two alternatives forced choice): Test in which two samples are played altering some parameter, until subject can no longer hear difference between the samples.

A flexible **Generic Test** class is used as base for all tests. Most tests listed above can be implemented with the generic test only.

Testing features

Many options and parameters allow customizing the tests. Most features are available for all tests.

- **Multiple questions**: All tests can include any number of questions. GP contains ready-made question components for giving *grades*, *multiple-choices*, and *rank-order*. Also additional question components can be added.
- **Free or fixed sequence**: The sample sequence can be fixed (freely configurable with multiple samples and pauses between samples) or free where the subject selects in which order samples are played.
- **Time limit for answering**: A timeout can be set to limit the time the subject has for giving the answers for a test item. Without a timeout the subject can have as much time as he needs to decide the answers. With a timeout, the test goes to the next item when the time ends.
- **Parallel switching**: Normally when the subject switches to another sample, the currently playing sample stops and the another starts from beginning. In parallel mode the switching to another sample is done by cross-fading to the other sample and does not just jump to the start to the beginning to the sample. Useful in test where long samples are compared. Parallel switching is only usable for free sequence tests.
- **Playlists** are used to define the order of the test items that are presented. Different playlists may be defined for different sessions.

Audio output features

The **sound player** handles the audio output of the **GuineaPig** system. It reads the sample files, mixes them together and sends the audio data to output devices. The player is written in C. A higher level **Java** module is provided for using the player.

The Sound Player uses the *SGI Audio Library* and the *SGI Audio File Library*, therefore all supported devices and sound file formats supported by them are available in GP. The audio devices includes the normal analog stereo outputs on most systems as well as the default or additional digital outputs up to eight digital channels on one device (ADAT). The supported audio file formats include: AIFF/AIFF-C, Next/Sun, Wave, MPEG1, raw data, and many more.

Some features of the sound player:

- **Multichannel**: Number of output channels is limited only by output devices of the machine. The sound player can use multiple audio devices in parallel, emulating wider output port than a single device. For example, when three ADAT cards are used, 24-channel output is possible. Multiple devices are syncronized automatically.
- **Virtual players** allow partitioning the output of GP into smaller sections. For example, single ADAT output could be used to implement four stereo players.
- **Many audio file formats**: All audio file formats supported by *SGI's audio file library* including: AIFF/AIFC, WAV, AU, MPEG-1 layers I/II and others with multiple sample rates and sample widths.
- **Sampling rates**: depending on the audio device, at least the most common rates from 8kHz to 48kHz are supported (device may limit the possible choices. For example, digital outputs generally support only 32/44.1/48 kHz).

- **24 bit audio**: SGI audio libraries can support up to 24 bits per sample. GP's sound player uses floating point calculation for rendering audio output.
- **Analog and digital outputs**: Most SGI workstations have analog outputs. Some also have digital outputs by default (Octane, some Origin and Onyx2 servers) or with an digital audio option (on O2 with the *PCI Professional Audio Option*). Digital output interfaces include ADAT/SPDIF/AES3, optical/RCA/BNC connectors. Analog output generally are 16bit, digital upto 24 bit (ADAT/AES3).
- **Sample mixing**: multiple samples can be mixed to the output with each with its own volume levels and faders.
- **Sample synchronization**: samples can be synchronized with sample frame accuracy.
- **Faders**: each sample has its own fader that does real fades using linear or dB linear fading. Also supports cross-fading between samples.
- **Volumes**: each sample sample has its on volume controls (with fader, see above). Also there is a calibration level for each sample that can be used to calibrate a set of samples used in a test. Player's output also have a calibration or master volume level control than can be used for setting the MCL level of a test session of general master output level.
- **Java module for controlling sound player**: the sound player is a separate program. A easy object-oriented java module is provided for controlling the player.

See also: SGI Audio Features.

Requirements

The system runs on and is designed only for Silicon Graphics' workstations with IRIX 6.3 or greater. The GP's sound player engine uses the newer SGI's *Audio Library* for IRIX 6.3 and up and POSIX threads (pthreads). In future versions, IRIX 6.5 will be required.

Most of the system is written in Java. Java 1.1 is required and it is freely available for download from SGI. In future, Java 1.2 (Java2) will be used.

Additional information

The **GuineaPig web-page** is located at:

www.acoustics.hut.fi/~hynde/GuineaPig2/index.html

The page contains the **AES 106th GuineaPig paper** and presentation slides presented at the *AES 106th Convention*, May 8-11 1999, Munich, Germany:

- **GuineaPig paper** in AES 106th: [gzipped PS (674K) / PDF (1103K)]
- **AES Presentation slides** on 25/May/1999: [gzipped PS (256K) / PDF (352K)]
  Slides shown at the paper presentation at 106th AES Convention. Includes extra slides that were excluded from the presentation to fit to the time allowed.

Last modified: Fri Jul 30 18:58:43 EEST 1999

Installation Instructions and Setup

1 · Requirements

The requirements of the system are:

- A silicon graphics workstation or server
- IRIX 6.3 (O2) or 6.4 (Octane, Origin-series). Should work on IRIX 6.5 too, but not tested.
- Audio outputs, analog or digital.
- Java 1.1

The packages can be installed either with *swmgr* (Software Manager) or *inst*.

2 · Download and Install Java

If Java is not installed on your system or you want to upgrade it, download and install the **Java Execution Environment** and optionally the **Java Development Environment** for SGI from SGI's java page. Follow the instructions on the page to install java.

To see whether you have java installed or its version, type

```
  versions java_eoe java_dev
```

in a shell window. It will show a subsystem list installed to your system. If *java_eoe* is not displayed in the list, you will need to download and install the Java Execution Environment. I suggest **Java Development Environment 3.1.1 (Sun JDK 1.1.6) & Java Execution Environment 3.1.1 (Sun JRE 1.1.6)** or later are installed.

3 · Install GuineaPig System

Install the GuineaPig package with *inst* or *swmgr*.

4 · Configuring GuineaPig (personal settings)

You must add the GuineaPig binaries directory to your shell's path. For bourne-type shells you could use (for example, in file .profile):

```
  PATH="$PATH:/usr/GuineaPig/bin"
  export PATH
```

or for csh and derivates (tcsh, etc.) (for example, in file .cshrc):

```
  setenv PATH "$PATH:/usr/GuineaPig/bin"
```

That way the shell can find the GuineaPig commands and the sound player.

Last modified: Fri Jul 30 19:01:10 EEST 1999

Directory structure and files

Each test set should have it's own directory where the information about the test is stored. This information contains the parameters of the test, playlists, the samples and the results. The structure of the test directory is shown below.

1 · Test configuration

The file *test.config* contains the information needed to perform the test. It tells the type of the test, samples used, number of testees and so on. This file can be created with Test creation module. The test config file is usually named '*test.properties*' but the name can be anything you want.

More detailed information: Format of the test.config file

2 · Test items

Test items file contains information on how samples are used together in a test. In it the tester can generate pairs or triples of samples that form the test items. The test items file's name is usually '*items.properties*'.

More detailed information: Format of the test items

3 · Playlist

The playlist tells the order the individual *test items* are presented to the subject. A single test item can be, for example, a certain sample pair in the case of the AB-test. As the result of performing one test item an entry is generated in the results file.

More detailed information: Format of the playlist

4 · Sample list

The sample list relates the sample IDs the test items use to the actual audio samples. The information about a sample contains at least the sample ID and the file name of the audio file. Depending on the type of sample, additional information may be also stored.

More detailed information: Format of the samplelist file

5 · Audio samples

The actual audio sample files are usually kept in its own directory. The available sound sample formats are listed in the Sound Player documentation.

6 · Results

The results from a test session are stored in a file using Java's serialization system. One test session produces one session log or results file. Some information is stored about the session in addition to subjects' answers.

The file name of the session result file is automatically generated from the session ID. It is the form of '*results_sessionID*' where *sessionID* is the session ID of session. Session IDs are generally like '*S14864464*' (automatically generated session ID, the letter 'S' followed by a number of minutes after January 1st, 1970) or '*ses03*' (explicitly set session ID by tester). A name for a session result file could be, for example: '*results_S14864464.ser*' (.ser-suffix is added to show that it is a serialized file).

More detailed information: Format of the results, result processing

---

· **Document index** ·

Last modified: Fri Apr 17 16:38:18 EET DST 1998

Test config file

The test config file is the main configuration file that the Test creation module creates. It contains the test specific configuration information needed by the GP system when actually running the test.

The test config is stored in file *test.properties* in the test's main directory. The name is only a suggestion, the file can have any name.

See what parameters are needed in test parameter file by different test types.

· **Directory Structure** · **Document index** ·

Last modified: Fri Apr 17 16:45:36 EET DST 1998

Playlist

The playlist is a file that tells the order the individual *test items* are presented to the subject. A single *test item* can be, for example, a certain sample pair in the case of the AB-test.

The playlist is stored in a file in the test's main directory. The playlist can be specific to a certain test session or it can be global for all sessions. If the playlist's file name has not been explicitly specified, a session specific playlist is searched for from file named '*playlist_sessionID*' where *sessionID* is the session ID of the current session. For example, if the session's ID is '*ses3*', the file named '*playlist_ses3*' is looked for. If no session specific play list is not found, a file named '*playlist*' is looked for (the global playlist). If it is not found, a playlist will not be used and items will be presented to the subject in the order the system reads them from the items list (but the presentation order may not be the same as the order of items in the items file).

The playlist is a text files that contains the list of item IDs to be played to a subject. Each line is contains one item ID. Comments can be added, the '#'-character marks the start of a comment which continues to the end of line. Empty lines are also allowed.

The following example explains the format of the file. There are four items listed in the playlist and they are to be played to the subject in the order *item1*, *item3*, *item4*, *item2*.

```
#
# Order of items and which items are to be included in this test.
# Not all items in items file have to be used.
#
item1
  #and another comment
item3# this is also a comment
#yet another comment
item4
item2
```

Sample list

The sample list relates the sample IDs the test items use to the actual audio samples. The information about a sample contains at least the sample ID and the file name of the audio file. Depending on the type of sample, additional information may be also stored.

The list is currently a text file that contains the information needed to construct the java-objects used for playing samples. Here is an example of a sample list:

```
# Samples file
#

pirr44.class=guinea.player.SoundSample
pirr44.name=pirr44
pirr44.filename=samples/short/pirr44.aiff

pirr32.class=guinea.player.SoundSample
pirr32.name=pirr32
pirr32.filename=samples/short/pirr32.aiff

pirr22.class=guinea.player.SoundSample
pirr22.name=pirr22
pirr22.filename=samples/short/pirr22.aiff

pirr16.class=guinea.player.SoundSample
pirr16.name=pirr16
pirr16.filename=samples/short/pirr16.aiff

pirr11.class=guinea.player.SoundSample
pirr11.name=pirr11
pirr11.filename=samples/short/pirr11.aiff

pirr8.class=guinea.player.SoundSample
pirr8.name=pirr8
pirr8.filename=samples/short/pirr8.aiff

mcll.class=guinea.player.SoundSample
mcll.name=mcll
mcll.filename=samples/pink.aiff
```

Lets take a closer look at one sample’s information:

```
pirr44.class=guinea.player.SoundSample
pirr44.name=pirr44
pirr44.filename=samples/short/pirr44.aiff
```

First in the start of every line is a *sample ID*. In this case it is *pirr44*. After the ID, there is a property name and it value if the form of ’*name=value*’ or ’*name: value*’. For example, the line

```
pirr44.filename=samples/short/pirr44.aiff
```

sets the property ’*filename*’ of the sample *pirr44* to value ’*samples/short/pirr44.aiff*’.

The properties for sound samples are:

class=*java class name* (required)
    The java class name of the sample. For now only ’*guinea.player.SoundSample*’ makes sense.

name=*sample ID* (required)
: The sample ID of the sample, should be same as the ID in the beginning of line.

filename=*file name of audio sample* (required)
: The file name of the real audio file.

correctionVolume=*correction* (optional)
: A correction volume or calibration for sample’s volume level. The correction is relative to the sample’s normal level. It can be a level multiplier or it can be a dB level. Example: a value of ’-6dB’ is equivalent to about ’0.5’ multiplier.

· **Directory Structure** · **Document index** ·

Last modified: Wed Mar 25 14:02:53 EET 1998

Results

The results from a test session are stored in a file using Java’s serialization system. A session log of a session contains information about the session:

- Session ID.
- Starting time of session.
- Ending time of session.
- Session’s MCL level.
- If the test was aborted because some error, the exception that caused the test to abort is stored.
- Answers for each test item.

The answers are copies of the test items used in the session with the subject’s answers added. Each subject has its own copy of the test item and his/hers answers are added to that copy. Information for each test item for each subject in addition to test item parameters are:

- The subject ID of the subject that gave these answers.
- The session ID.
- Starting time of this item.
- The duration of the item for this subject or timeout-indicator that time for that item ran out.
- Number of samples played or number of sample switches the subject made before deciding the answers.
- Answers for this item. It may not contain all answers if time limit was enforced and time ran out for this item.

Serialized result files can be converted to human/computer readable text files with the result processing tools for analysis with statistical packages.

· **Directory Structure** · **Document index** ·

Last modified: Wed Mar 25 11:59:17 EET 1998

Tests

· **Document index** ·

Last modified: Tue Jul 27 12:44:06 EEST 1999

Test types

A variety of test types has been implemented including:

- **A/B**: Test in which the test subject chooses one out of two samples played to him/her.
- **A/B/X**: Test in which the sample X is the same sample as A or B. Subject chooses which one sample X is.
- **A/B/C**: Test in which A is the reference sample and the subject chooses how much samples B and C (one of which is the same sample as A) resemble the reference sample A. (Actually this test is referred as a Ref/A/B-test).
- **A/B Scale**: An A/B test in which subject gives an answer for both samples on a scale specified by the test creator.
- **A/B Scale, Fixed Reference**: A test in which subject gives an answer how sample compares to a reference.
- **A/B Scale, Hidden Reference**: A test in which subject gives an answer for both samples on a scale specified by the test creator. One of the samples A or B is the hidden reference.
- **Single Stimulus**: A Test in which a single stimulus signal is played and then graded.
- **TAFC** (Two alternatives forced choice): Test in which two samples are played altering some parameter until the subject can no longer hear the difference between the samples.
- **SSMS** (Single Stimulus Mixed Source)
- **Generic test**: A generic test type.

Additional options and features for most tests:

- **Multiple answers**: All tests can include additional questions. Questions (answers) can be choices and grades (configurable).
- **Free or fixed sequence**: The sample sequence can be fixed (freely configurable with multiple samples and pauses between samples) or free where the subject selects in which order samples are played.
- **Time limit for answering**: A timeout can be set to limit the time the subject has for giving the answers for a test item. Without a timeout the subject can have as much time as he needs to decide the answers. With a timeout, the test goes to the next item when the time ends.
- **Parallel switching**: Normally when the subject switches to another sample, the currently playing sample stops and the another starts from beginning. In parallel mode the switching to another sample is done by cross-fading to the other sample and does not just jump to the start to the beginning to the sample. Useful in test where long samples are compared. Parallel switching is only usable for free sequence tests.
- **MCLL setting**: The most comfortable listening level can be fixed by the tester or is set by the subject within the volume range defined by the tester.

· **Tests index** · **Document index** ·

Last modified: Wed Jul 28 16:20:01 EEST 1999

A/B Test

In an **A/B Test** two samples are compared. The test subject chooses one out of two samples played to him/her as the answer. The question can be for example: '*Which of the samples sounds better?*'.

Here is a list of parameters needed by the test. This table lists only the parameters that have some special information about parameters or that override or adds new parameters to the common test parameters list.

Parameter	Value	Description	R/O
class	AB	A class alias (*AB*) or a fully qualified java class name (*guinea.logic.ABTest*) of the class that handles the A/B test.	Req.

Test item parameters

Test items for A/B test have two parameters **A** and **B** which are sample IDs of the samples which are compared.

Parameter	Value	Description	R/O
A	*sample ID*	Sample ID of sample A.	Req.
B	*sample ID*	Sample ID of sample B.	Req.

Here is an example of an A/B test item:

```
# which sample is the A sample
item1.A=pirr44
# which sample is the B sample
item1.B=pirr32
```

The item's itemID is '*item1*'.

Results

As a result the subject's answer is recorded. It is either 'A' or 'B' (which was better, etc.).

· **Test types** · **Tests index** · **Document index** ·

Last modified: Wed Jan 20 11:16:08 EET 1999

A/B/X Test

In an **A/B/X** test the sample X is the same sample as A or B. Subject chooses which one sample X is.

Here is a list of parameters needed by the test. This table lists only the parameters that have some special information about parameters or that override or adds new parameters to the common test parameters list.

Parameter	Value	Description	R/O
class	ABX	A class alias (*ABX*) or a fully qualified java class name (*guinea.logic.ABRefTest*) of the class that handles the A/B/X test.	Req.

Test item parameters

The items used in A/B/X test have three parameters A, B, and Ref (reference) which are sample IDs of the samples which are compared. The Ref sample is the X in the case of A/B/X test.

Parameter	Value	Description	R/O
A	*sample ID*	Sample ID of sample A.	Req.
B	*sample ID*	Sample ID of sample B.	Req.
Ref	*sample ID*	Sample ID of the reference (X) sample Ref.	Req.

Here is an example of an A/B/Ref test’s item:

```
item2.A=pirr22
item2.B=pirr32
item2.Ref=pirr32
```

Results

As a result the subject’s answer is recorded. It is either ’A’ or ’B’ (which is the same as the reference X.

Last modified: Wed Jan 20 11:32:47 EET 1999

A/B/C Test

In an **A/B/C** (or **A/B/Ref**) test the subject chooses how much samples A and B (one of which is the same sample as Ref) resemble the reference sample Ref. Grades are given for A and B.

Here is a list of parameters needed by the test. This table lists only the parameters that have some special information about parameters or that override or adds new parameters to the common test parameters list.

Parameter	Value	Description	R/O
class	ABC	A class alias (*ABC*) or a fully qualified java class name (*guinea.logic.ABRefTest*) of the class that handles the A/B/C test.	Req.

Test item parameters

The test items used in A/B/C tests contain three parameters A, B, and Ref (reference) which are sample IDs of the samples which are compared.

Parameter	Value	Description	R/O
A	*sample ID*	Sample ID of sample A.	Req.
B	*sample ID*	Sample ID of sample B.	Req.
Ref	*sample ID*	Sample ID of the reference sample Ref.	Req.

Here is an example of an A/B/Ref test’s item:

```
item2.A=pirr22
item2.B=pirr32
item2.Ref=pirr32
```

The items itemID is ’*item2*’.

Results

As a result the subject’s answers are recorded. Grades are given for samples A and B on how they compare to the reference.

Last modified: Wed Jan 20 11:29:34 EET 1999

A/B Scale Test

In an **A/B Scale Test** two samples are compared. The subject gives a grade for both samples from a scale defined by the tester. The question can be for example: '*How good is A?*'.

Here is a list of parameters needed by the test. This table lists only the parameters that have some special information about parameters or that override or adds new parameters to the common test parameters list.

Parameter	Value	Description	R/O
class	ABscale	A class alias (*ABscale*) or a fully qualified java class name (*guinea.logic.ABTest*) of the class that handles the A/B scale test.	Req.

Test item parameters

Test items for A/B Scale test have two parameters **A** and **B** which are sample IDs of the samples which are compared.

Parameter	Value	Description	R/O
A	*sample ID*	Sample ID of sample A.	Req.
B	*sample ID*	Sample ID of sample B.	Req.

Here is an example of an A/B test item:

```
# which sample is the A sample
item1.A=pirr44
# which sample is the B sample
item1.B=pirr32
```

The item's itemID is '*item1*'.

Results

As a result the subject's answers are recorded. A grade is given for both A and B.

Last modified: Wed Jan 20 11:21:14 EET 1999

A/B Scale (Fixed reference) Test

In an **A/B Scale (Fixed reference)** test the subject gives a grade for a sample on how the sample compares to the reference.

Here is a list of parameters needed by the test. This table lists only the parameters that have some special information about parameters or that override or adds new parameters to the common test parameters list.

Parameter	Value	Description	R/O
class	ABscaleFixedReference	A class alias (*ABscaleFixedReference* or *ABscaleFR*) or a fully qualified java class name (*guinea.logic.ABscaleFTest*) of the class that handles the A/B Scale (fixed ref.) test.	Req.

Test item parameters

The test items used in this test have two parameters A and Ref (reference) which are sample IDs of the samples which are compared.

Parameter	Value	Description	R/O
A	*sample ID*	Sample ID of sample A.	Req.
Ref	*sample ID*	Sample ID of the reference sample Ref.	Req.

Here is an example of an ARef item:

```
# which sample is the A sample
item3.A=pirr32
# which sample is the B sample
item3.Ref=pirr44
```

The items itemID is '*item3*'.

Results

As a result the subject's answer is recorded. A grade is given for sample A on how it compares to the reference.

Last modified: Wed Jan 20 11:37:07 EET 1999

A/B Scale (Hidden reference) Test

In an **A/B Scale (Hidden reference)** test the subject gives an answer for both samples on a scale specified by the test creator. One of the samples A or B is the hidden reference.

Here is a list of parameters needed by the test. This table lists only the parameters that have some special information about parameters or that override or adds new parameters to the common test parameters list.

Parameter	Value	Description	R/O
class	ABscaleHiddenReference	A class alias (*ABscaleHiddenReference* or *ABscaleHR*) or a fully qualified java class name (*guinea.logic.ABRefTest*) of the class that handles the A/B Scale (hidden ref.) test.	Req.
hiddenReference	true	This test is a hidden reference version of this test.	Req.

Test item parameters

The test items used this test have three parameters A, B, and Ref (reference) which are sample IDs of the samples which are compared.

Parameter	Value	Description	R/O
A	*sample ID*	Sample ID of sample A.	Req.
B	*sample ID*	Sample ID of sample B.	Req.
Ref	*sample ID*	Sample ID of the reference sample Ref.	Req.

Here is an example of an A/B/Ref test's item:

```
item2.A=pirr22
item2.B=pirr32
item2.Ref=pirr32
```

The items itemID is '*item2*'.

Results

As a result the subject's answers are recorded. Grades are given for samples A and B on how they compare to the reference.

Last modified: Wed Jan 20 11:38:39 EET 1999

Single Stimulus Test

In a **Single Stimulus Test** Test a single stimulus signal is played and then graded.

Here is a list of parameters needed by the test. This table lists only the parameters that have some special information about parameters or that override or adds new parameters to the common test parameters list.

Parameter	Value	Description	R/O
class	SingleStimulus	A class alias (*SingleStimulus* or *SS*) or a fully qualified java class name (*guinea.logic.SSTest*) of the class that handles the Single Stimulus test.	Req.

Test item parameters

The test items used in Single Stimulus test have a single parameter A which is the sample ID of the sample which is graded.

Parameter	Value	Description	R/O
A	*sample ID*	Sample ID of sample A.	Req.

Here is an example of an Single Stimulus item:

```
# which sample is the A sample
item1.A=pirr44
```

The items itemID is '*item1*'.

Results

As a result the subject's answer is recorded. A grade is given for sample A.

· **Test types** · **Tests index** · **Document index** ·

Last modified: Wed Jan 20 11:41:05 EET 1999

TAFC Test

In an **TAFC (Two alternatives forced choice) Test** two samples are compared. Two samples are played altering some parameter until the subject can no longer hear the difference between the samples.

The test uses the procedures and terms described in

Levitt H. (1971) *Transformed Up-Down Methods in Psychoacoustics*

Here is a list of parameters needed by the test. This table lists only the parameters that have some special information about parameters or that override or adds new parameters to the common test parameters list.

Parameter	Value	Description	R/O
class	TAFC	A class alias (*TAFC*) or a fully qualified java class name (*guinea.logic.TAFCTest*) of the class that handles the TAFC test.	Req.

The TAFC has some special parameters. Prepend parameter names with '*TAFC.*' string, for example:

```
# pause length is two seconds
TAFC.pauseLength=2.0
```

Here is the list:

Parameter	Value	Description	R/O
ruleType	*integer*	The number of the rule that decides when to turn the direction. Currently rule number 1 is implemented. Default is 1.	Opt.
answerName	*questionID*	The question ID to use for storing the answer. The default is ’*level*’.	Opt.
pauseLength	*seconds*	The time to answering when sample pair has played. The answer decides whether to go to the same direction or change direction (depends on the rule). Default is 2 seconds.	Opt.
initialLevel	*volume*	The initial volume level when this item starts. Both linear scale and dB scale can be used. Default is +0.0dB (1.0 linear).	Opt.
stepSize	*volume offset*	The level step size. Both linear scale and dB scale can be used. The default is 3.0dB.	Opt.
maxRuns	*integer*	Number of runs that is required. Default is 6.	Opt.
logTrial	*true* or *false*	Whether to log each trial and answer for this item in addition to the final level. Default is *false*.	Opt.
trialAnswerName	*questionID*	The question ID to use for the trial list of rach item if trials are logged. Default is ’*trial*’.	Opt.

Test item parameters

The test items used in TAFC test have two parameters B and Ref which are sample IDs of the samples which are compared.

Parameter	Value	Description	R/O
B	*sample ID*	Sample ID of sample B.	Req.
Ref	*sample ID*	Sample ID of the reference sample Ref.	Req.

Here is an example of an B/Ref item:

```
# which sample is the B sample
item3.A=pirr32
# which sample is the Ref sample
item3.Ref=pirr44
```

The items itemID is ’*item3*’.

Results

As a result a volume level is stored at the end of test procedure.

· **Test types** · **Tests index** · **Document index** ·

Last modified: Wed Jan 20 11:46:39 EET 1999

SSMS Test

In an **SSMS (Single Stimulus Mixed Source) Test** three samples are played and the level of one sample is altered. The sample S1 is stereo background noise. Sample S2 is mono noise and sample S3 is a mono speech sample. Sample S3's level is controlled by the subject and its output is mixed together with the right output channel. The test can also be configured so that the variable gain controls both S2 and S3 or only S2.

Here is a list of parameters needed by the test. This table lists only the parameters that have some special information about parameters or that override or adds new parameters to the common test parameters list.

Parameter	**Value**	**Description**	**R/O**
class	SSMS	A class alias (*SSMS*) or a fully qualified java class name (*guinea.logic.SSMSTest*) of the class that handles the SSMS test.	Req.

The SSMS has some special parameters. Prepend parameter names with '*SSMS.*' string, for example:

```
# set level of sample S1
SSMS.S1Volume=-10dB
```

Here is the list:

Parameter	Value	Description	R/O
S1Volume	*Volume level*	Se the volume of the sample S1. The volume level can be given in linear, decibel or percent scale. The default is to use the sample’s inherent level (level is 0dB (decibel) or 1.0 (linear)).	Opt.
S2Volume	*Volume level*	Se the volume of the sample S2. It is used to set the constrant gain for S2 or it is it’s initial gain if variable gain is used for this signal.	Opt.
S3Volume	*Volume level*	Se the volume of the sample S3. It is used to set the constrant gain for S3 or it is it’s initial gain if variable gain is used for this signal.	Opt.
monochannel	*channel index*	To which output channel to direct the output of the mono samples S1 and S2. Default is 1 (right), 0 would be left.	Opt.
ensureNoOverflows	*true* or *false*	Limit output volume so that no overflows should occur when many samples are played together.	Opt.
scaleMax	*Volume level*	The maximum on the scale that the subject uses for setting the level of samples S2 and S3. This is used to avoid overflows. The default is that the maximum of the scale is 0dB (1.0 linear). This parameter has no effect when ensureNoOverflows is false.	Opt.
levelControlName	*questionID*	The question ID that is also used to control the level of the S2 and S3 samples. Default is ’level’.	Opt.
levelControlsSignals	*list of signals*	Specify which signals the level (gain) controller controls. By default it is ’S3’ (variable gain for S3, constrant gain for S2). Multiple signals are specified with a comma-separated list of signal names, for example, the list ’S2,S3’ tells the gain controller to set the gain for both S2 and S3.	Opt.

Overflow avoidance

This test can try to avoid any overflows when many samples are mixed together. In SSMS test case, three samples S1, S2 and S3 are playing at the same time and they are all added together. If all samples have full range, overflows are bound to occur. The SSMS test tries to avoid overflows so that it limits the MCL level that can be set. If all samples S1-3 have max volumes set to 1.0 (0dB), to maximum level they together can be 3.0 (+9.5dB). The output level scale is then limited to 1/3 (-9.5dB) so that the digital output level never goes to more than 1.0 (0dB).

Test item parameters

The test items used in SSMS test have three parameters S1, S2 and S3 which are sample IDs of samples.

Parameter	**Value**	**Description**	**R/O**
S1	*sample ID*	Sample ID of the S1 sample. It is a stereo background noise sample.	Req.
S2	*sample ID*	Sample ID of the S2 sample. It is a mono noise sample.	Req.
S3	*sample ID*	Sample ID of the S3 sample. It is a mono speech sample.	Req.

Here is an example of an SSMS item:

```
# which sample is the S1 sample
item1.S1=tausta1
# which sample is the S2 sample
item1.S2=noise1
# which sample is the S3 sample
item1.S3=crapachi
```

The items itemID is '*item1*'.

Results

As a result a volume level is stored.

Last modified: Wed Jan 20 11:43:52 EET 1999

Generic Test

The **Generic test** allows for various test in addition to the predefined test types in GP. The generic test is also used as a base for other tests in GP.

Here is a list of parameters needed by the test. This table lists only the parameters that have some special information about parameters or that override or adds new parameters to the common test parameters list.

Parameter	Value	Description	R/O
class	GenericABC	A class alias (*GenericABC*) or a fully qualified java class name of the class that handles the test.	Req.
generic.sampleParams	*list of names*	A comma-separated list of names of parameters that are samples.	Req.

Note: All other test classes in GP use the generic test as their base and they predefine the *sampleParams* parameter automatically. The parameter is only required if using the generic test directly or using special item features.

Test item parameters

The simplest way of defining information for special items is to use the *sampleParams* parameter above. For example, if four samples are compared in the test, the configuration file would contain a line like this:

```
# Sample parameters that are sample IDs
generic.sampleParams:   A,B,C,D
```

The list of item parameters would be like this (like in other tests):

Parameter	Value	Description	R/O
A	*sample ID*	Sample ID of sample A.	Req.
B	*sample ID*	Sample ID of sample B.	Req.
C	*sample ID*	Sample ID of sample C.	Req.
D	*sample ID*	Sample ID of sample D.	Req.

Here is an example of such an item:

```
item2.A=pirr22
item2.B=pirr32
item2.C=pirr16
item2.D=pirr11
```

The item’s itemID is ’*item2*’.

More specialized items

If additional parameters that are not samples or they are optional, the simple *sampleParams* is not enough. A *default template* needs to be created. The default item template is always used as base when items are loaded from items file. It defines the list of names of parameters that are used in the item, and which of them are required.

For example, defining a template for a test where three sets of speaker systems are tested using different virtual players and samples: The needed parameters would perhaps look like this:

Parameter	Value	Description	R/O
A	*sample ID*	Sample ID of sample A.	Req.
B	*sample ID*	Sample ID of sample B.	Req.
C	*sample ID*	Sample ID of sample C.	Req.
A_player	*player ID*	The ID of the player that is used as player for sample A.	Opt.
B_player	*player ID*	The ID of the player that is used as player for sample B.	Opt.
C_player	*player ID*	The ID of the player that is used as player for sample C.	Opt.

The **_player* parameters are not sample ID, so they cannot be given in *sampleParams*. They are also optional, they do not have to be defined.

The configuration definitions needed for such test would then be:

```
# Default item template:
#  Define which parameters are used
itemTemplates.default.parameters:         A,B,C,A_player,B_player,C_player
#  Which parameters are required (others are optional)
itemTemplates.default.requiredParameters:  A,B,C
#
# Which parameters are samples:
generic.sampleParams:   A,B,C
```

A test item would then look like this in the items file:

```
item6.A:        samp34_1ch
item6.A_player: set1ch
item6.B:        samp34_5ch
item6.B_player: set5ch
item6.C:        samp34_2ch
item6.C_player: set2ch
```

Special parameters

The generic test currently recognizes one special parameter: *player* to use for a sample. In the future more features like these may be added when needed.

Player for sample

The generic test can assign a specific player that plays that sample during that item. It looks for a player ID for a sample from item parameter whose name is in the form '**<sampleParam>_player**', where *<sampleParam>* is a parameter in the *sampleParams* list. If a player ID is found, the corresponding player is assigned as the player for that sample. If no player ID is found, default player is used.

For this to work, the <sampleParam>_player parameters must be defined as item parameters. See examples above in *special items*.

See also sound player configuration for information on how to define virtual players.

Results

As a result the subject's answers are recorded. The testers may define the questions in any way they want.

Last modified: Wed Jul 28 16:11:00 EEST 1999

Test Items

The **test items** contain the parameters of individual items or cases in a test. An item contains usually the names of the samples played in this item. For example, parameter ’A’ contains the sample ID ’ping3_64kbs’ and parameter ’B’ contains the sample ID ’ping4_64kbs’. Test items are also used to store the answers given by a subject for this item. For each subject a copy of the item is created and the answers and other information is added. See result file format for details.

The previous versions of GP (pre 2.03) there were different test item classes that were used for different tests. They all have now been replaced with a single generic item class. Also defining, extending and configuring test items have been made easier and more powerful. For most part the test items file format is the same as before, except that the **class** parameter (which told the test item class to use) should be omitted.

However, the old files should still work for some time but it is recommeded that the files are converted to new format for better compatibility to future versions.

Test items file format

The **test items file** contains a list of test items. The parameter names are prepended with the **item ID**. Here is an example of a test items file from an A/B test:

```
# Item ’item1’
# which sample is the A sample
item1.A=pirr44
# which sample is the B sample
item1.B=pirr32

# Item ’item2’
# which sample is the A sample
item2.A=pirr16
# which sample is the B sample
item2.B=pirr12
```

The file contains two test items whose item IDs are ’*item1*’ and ’*item2*’. Each have two parameters A and B.

See each test type’s documentation for the needed item parameters for that test.

Moving to newer format from older files

Converting to new format is very easy, just remove or comment out all the item class name definitions (’*itemXXX.class=classname*’) from the old file.

Last modified: Wed Jan 20 11:06:36 EET 1999

Test creation

At this point there wasn’t enough time to make flashy graphical test creation tools. The test is created by writing the required test configuration files with a text editor.

The prodecure for creating a test usually like this:

- **Select test type** - What kind of test to run and.
- **Create test directory** - Test directory is used to store the information about the test.
- **Select sound samples -** Decide which sound samples to use in the test and possibly create and/or edit the files and write the sample list configuration file.
- **Create test items -** Define test item parameters, usually which samples are played to the subject. Write the test items file.
- **Create playlist(s) -** Make and write global playlist for all test sessions or session specific playlists.
- **Select subject’s UI -** Select which answer and control components to use. Write UI parameters file.
- **Set test parameters -** Write the test configuration file. Set needed parameters such as file names, sequences, timeouts, modes and so on.

Select test type

The first thing is to decide which type of test to use. Select a test from the **list of test types**.

Also see:

- **Time limits and warnings -** use/don’t use limited time for answering, length of time
- **Sample sequence -** sequence type (free/fixed) and fixed sequence
- **Sample switching -** normal or parallel switching

Test directory

A **test directory** is a directory where the files of a test are kept. It contains the test parameters files and samples (usually in their own subdirectory). The results from test sessions are also written in the current directory (which should be the test directory). Actually using one directory for all files of a test is more of a guideline than a requirement.

When running a test you should first go to the test directory of the test and then run the test (with the RunTest-program).

Sound samples

Decide which **sound samples** to use for testing. Usually sound samples are kept in a subdirectory in the test directory, for example, make a directory named *samples* for keeping the samples.

You probably have to edit or create some samples. Here are some requirements (must) or recommendations (should) for samples that are played together during one item:

- Samples should be of same length. This is more important if parallel switching is used. Samples are not required to be of same length but would be better if they are. The system will happily play samples with different lengths at the same time.
- The phase of the samples should be the same, that is, there is no extra delay in the beginning of one sample that is not in the second sample. This is more important of parallel switching is used.

These apply to all samples in a test:

- All samples in test’s sample set must have the same sample rate and must match the sample rate of the player.
- All samples must have the same number of channels.
- Samples should be calibrated, for example, the level of all samples should be the same. The sample list below provides a way to calibrate the level of samples without editing the actual sample file.
- Samples’ format must be one of valid audio file formats accepted by the player. It would be better if they are uncompressed. AIFF/AIFF-C would be a good choice.

Tools for audio files include: *dmconvert*, *mediaconvert*, *sfconvert*, *dminfo*, *sfinfo*, *soundeditor*, *soundtrack*. See the manual pages for description and/or the *Digital Media Tools Guide* (IRIS InSight book).

The **sample list** relates the sample IDs the test items use to the actual audio samples. The information about a sample contains at least the sample ID and the file name of the audio file. Depending on the type of sample, additional information may be also stored.

More detailed information: Format of the samplelist file

Test items

Write the test items. See test types and test items for needed parameters.

Playlists

Playlists are files that tell which test items (possibly a subset of all test items) to play to the test subject and in which order. A playlist can be ’*global playlist*’ which means that the same order of items is used in all sessions and for all subjects unless a ’*session specific playlist*’ is used to set a specific playlist for a specific session. The playlist(s) are stored in the test’s directory.

More detailed information: Format of the playlist

Subject’s user interface

No complete documents for subject UI components are not yet ready. Copy and modify the demo example files. They are quite self-explatory.

Test parameters

Write the test parameters file. Add parameters mentioned earlier:

- **Time limits and warnings -** use/don’t use limited time for answering, length of time
- **Sample sequence -** sequence type (free/fixed) and fixed sequence
- **Sample switching -** normal or parallel switching
- **MCLL setting -** parameters for setting the most comfortable listening level, fixed or subject sets it.
- See also **common properties -** other common, additional, required parameters for a test.

· **Tests index** · **Document index** ·

Last modified: Tue Jul 21 10:01:09 EEST 1998

Common test properties

class=*java class name* (required)
The java class name of the class that handles the test. See test specific properties for the correct class.

uifile=*name of the subject UI property file* (required)
The name of the file that contains the subject UI properties (question and control component specifications).

itemsfile=*name of the items file* (required)
The name of the file that contains the test items.

playlistfile=*name of the playlist file* (optional)
The name of the playlist file. If it omitted, the playlist will be searched from the default places.

samplefile=*name of the sample list file* (required)
The name of the file that contains the sample list.

sequenceType=*fixed* | *free* (required)
Type of sequence, *free* or *fixed*. If sequence type is fixed, the sequence is also required.

sequence=*sample sequence* (required if sequenceType is fixed)
The sample sequence for fixed sequence tests. If Sequence type of test is *fixed*, the sequence is required.

itemTimeout=*seconds* (optional)
The timeout in seconds. It is how many seconds the subject has time to give answers. If time runs out, the item has ’timeouted’ and the test goes on to the next item.

itemWarningTimeout=*seconds* (optional)
The indicator warns that time is about to run out when time left is less than ’seconds’ seconds.

Sound player properties:

player.class=*java class name* (required)
The java class name of the SoundPlayer class. Should be *guinea.player.SoundPlayer*.

player.rate=*sampling rate* (required)
The sample rate of output in Hz.

player.channels=*number of channels* (required)
Number of output channels.

player.device=*audio device* (recommended)
The device name of audio output device (*"Analog Out"*, *"ADAT Out"*, etc). If not set, the default output will be used but it would be a good idea to specifically set what device to use.

player.volume=*master volume correction* (optional)
The output master level correction. Similar to volume correction of samples except that it is for master output channels (affects all samples).

· **Test creation index** · **Tests index** · **Document index** ·

Last modified: Wed Apr 8 10:29:20 EET DST 1998

Common test parameters

Here is a table of most common test parameters used in most or all tests.

Parameter	Value	Description	R/O
class	*class name*	A fully qualified Java class name of a class that handles the test. A class alias name (a shorter name for the class) can be used also.	Req.
testDirectory	*directoryname*	The name of the test directory. All other config file names are relative to the test directory (unless they have absolute paths as filenames). If test directory is not specified, the system uses the directory where the test config file is loaded from or current directory if test config file's directory couldn't be determined.	Opt.
uifile	*filename*	Name of the file that contains information needed to construct the subject UI window. The filename is relative to the *test directory* unless absolute.	Req.
itemsfile	*filename*	Name of the file that contains test items for this test. The filename is relative to the *test directory* unless absolute.	Req.
samplefile	*filename*	Name of the sample list file. The filename is relative to the *test directory* unless absolute.	Req.
playlistfile	*filename*	Name of the playlist file. This is used mainly for a fixed playlist for all sessions. If you do not set this parameter, the playlists will be searched automatically from the default places. The filename is relative to the *test directory* unless absolute.	Opt.
resultFile	*filename*	Name of the session result file where to store answers from a session. It is better you don't set this parameter, the names for result files are generated automatically. The filename is relative to the *test directory* unless absolute.	Opt.
sequenceType	*sequence type*	The type of sample sequence played to the subject. It is either *free* (default) or *fixed*	Opt.
sequence	*sample sequence*	The sample sequence played to the subject in a fixed sequence test. * This is required if sequence type is *fixed*.	Req.*
itemTimeout	*time in secs.*	Set timeout time and enable or disable time limits. Default: no time limits.	Opt.
itemWarningTimeout	*time in secs.*	Set warning time. Default: zero.	Opt.

sampleSwitching	*switch type*	Set switching type. It is either *normal* or *parallel* (default).	Opt.
showPlayingSample	*true* or *false*	Whether to show to the subject which sample is playing currently. (default: true).	Opt.
autoPlaySample	*param. name*	Which sample to automatically start playing when item starts. Used only in free sequence tests.	Opt.

Soundplayer parameters

parameters for sound player. Prepend with the string ’player.’ in the test config file, for example, set sample rate to 44100Hz:

```
player.rate=44100
```

See sound player configuration for more detailed information.

Parameter	Value	Description	R/O
class	guinea.player.SoundPlayer	The soundplayer’s java class name. For normal use, the class name of the player should be omitted. It is used only if a special player is needed.	Opt.
rate	*sample rate*	Sample rate of the output.	Req.
channels	*num. of channels*	Number of output channels.	Req.
device	*audio device name*	The name of the audio device, for example: ’*Analog Out*’ or ’*ADAT Out*’. Device ’*default*’ will use the default device set with apanel.	Opt.
buflen	*buffer length*	The length of the mixing buffer. It can be used to adjust the length of the delay in output.	Opt.

Last modified: Wed Jul 28 16:07:55 EEST 1999

Running a test

A test is run using the **gpRunTest** command. It is started with options that tell it where to read the test configuration information and some other bits. See examples from command's manual page for how to use the command.

Usually a test goes like this:

1. Go to the test directory.
2. Run the test with the **gpRunTest** command.
   1. The test reads the parameters needed by the test (test items, samples, UI parameters), creates the subject UI window, etc.
   2. Test items are played to the subject and answers from the subject are saved.
   3. When all items have been played the results are written to a file. The file contains java objects containing copies of test items and corresponding answers from the subject. The results can be converted to a text file with result processing tools.

If an error occurs during the test, the system tries to save as much as it can (the items finished in the session).

For more complete instructions, see the manual page of gpRunTest.

Last modified: Mon Oct 5 11:00:39 EEST 1998

Sequences

A **sequence** tells which samples to play for the subject in which order and possibly with pauses between samples.

Free sequence

A **free sequence** means that a fixed sample sequence is not used or enforced. The subject selects which samples to play and in which order.

A free sequence is selected by setting the **sequenceType-**parameter in the test parameter file to ’*free*’:

```
# use free sequence
sequenceType=free
```

The **sequence-**parameter is not needed (and will be ignored if used) in a free sequence test.

Fixed sequence

A **fixed sequence** means that the sequence of samples presented to the subject is fixed and enforced so that the subject can’t affect the playing of samples. It is possible to use pauses between samples.

A fixed sequence is selected by setting the **sequenceType-**parameter in the test parameter file to ’*fixed*’ and setting the **sequence-**parameter to the sequence definition:

```
# use fixed sequence
sequenceType=fixed
sequence=<sequence>
```

The <*sequence*> is the sample sequence definition, see below for the format.

Defining sample sequence

A sequence is defined with a string that defines the order of samples and pauses between samples. A sequence consists of **sequence items** each of which define the pause length in the beginning of the sample and the sample name. The length of pause is given in seconds or in sample frames and may be omitted. The name of sample is required. The pause length and sample name are separated by a comma.

```
<sequenceitem> = [<pause>,]<sample>
<pause>        = <seconds>s | <frames>
```

The <*seconds*> (float) is the number of seconds. The letter ’s’ is used to indicate that the number is in seconds instead of sample frames. The <*frames*> (int) is the number of sample frames. Some examples:

```
0.1s,A
22050,B
A
```

the first plays sample A after a pause of 0.1 seconds. Second plays sample B after a pause of 22050 sample frames. If sample rate is 44100, 22050 is equivalent to 0.5s. The third plays sample A without a pause between A and the previous sample.

The whole **sequence** is a list of sequence items separated with a semicolon.

```
<sequence> = [<sequence>;]<sequenceitem>
```

Here is an example:

```
A;0.1s,B;0.5s,A,0.1s,B
```

The example plays samples A and B with a pause of 0.1 secods between the samples and a pause of 0.5 seconds between the sample pairs. Here’s an example of using this sequence in the test config file:

```
# use fixed sequence
sequenceType=fixed
sequence=A;0.1s,B;0.5s,A,0.1s,B
```

Automatic sample play when item starts

In free sequence tests you can start playing a sample automatically when the item starts. To do this, define the **autoPlaySample** parameter with a sample parameter name. For example:

```
# Start automatically playing sample A when item starts.
autoPlaySample=A
```

This will start playing the sample given with the item parameter A. This has the same effect as playing the ’A’ button on the subject UI’s sample-play controller (if there is one).

If this parameter is not defined, a sample is not automatically started.

Notes

Some notes about using sequences:

- There should not be any extra spaces in the sequence.
- The test system currently ignores the pause of the first sequence item in a sequence.
- The sample names used don’t refer to the sample IDs in sample lists. They refer to the parameter names of test items. For example, a test item for a A/B test contains the parameter names A and B whose values are the sample IDs of the samples that correspond to the names A and B.
- If fixed sequence is used, there should not be a sample playing controller is the subject’s UI. If it is there, it will be ignored. If free sequence is used, there must be a sample playing controller in the subject’s UI.

Last modified: Wed Jan 13 14:42:26 EET 1999

Timeout

A **timeout** can be set to limit the time the subject has for giving the answers for a test item. Without a timeout the subject can have as much time as he needs to decide the answers. With a timeout, the test goes to the next item when the time ends.

In addition a **timeout warning** can be set to alert the subject that the time is about to end: the warning indicator changes from green to yellow.

When a timeout occurs and the subject hasn't signalled that he is 'done', the indicator changes to red for a second and then goes to the next item. The results will show that a timeout occured for this item for this subject.

The timeout can be used with both free and fixed sequences. The behaviour of time limits differ slightly in fixes and free sequences, see notes.

Setting timeouts

By default the timeout time limit and timeout warning are not used (the timeout times are zero). The timeouts are enabled by setting the **itemTimeout** and **itemWarningTimeout** parameters in the test config file. Here is an example:

```
# item timeout in seconds
itemTimeout=10.0
# warning time before timeout
itemWarningTimeout=4.0
```

In this example the subject has 10 seconds to give answers for this item. When 4 seconds are left (6 seconds after the start of giving answers), the indicator changes from green to yellow to show that time is about to end. The **itemWarningTimeout** parameter may be left out (equivalent to setting it to zero) and then the indicator goes directly from green to red when time ends.

Notes

Some notes about using timeouts:

- With a **fixed sequence** the time starts when the whole fixed sequence has been played to the subject and answering is enabled.
- With a **free sequence** the time starts when the item starts. The time limit should be long enough that the subject has enough time to compare samples and switching between them multiple times. The time limit includes the time used to play samples and giving the answers.

Last modified: Thu Apr 16 10:40:56 EET DST 1998

Switching

Normally when the subject switches to another sample, the currently playing sample stops and the another starts from beginning. In **parallel switching** the switching to another sample is done by cross-fading to the other sample and does not just jump to the start to the beginning of the sample. Useful in test where long samples are compared. Parallel switching can be used only with free sequence tests.

Normal switching

In **normal switching** when the subjects switches to another sample, the currently playing sample is stopped and the another starts from beginning.

Normal switching is selected by setting the **sampleSwitching** parameter in the test config file to ’*normal*’:

```
# set switching to ’normal’
sampleSwitching=normal
```

Parallel switching

In **parallel switching** the switching to another sample is done by cross-fading to the other sample. Both samples are played at the same time and the sample position is the same in both samples.

Parallel switching is selected by setting the **sampleSwitching** parameter in the test config file to ’*parallel*’:

```
# set switching to ’parallel’
sampleSwitching=parallel
```

Sample switch cross-fade type

The type of the cross-fade used in parallel swithing can be selected. Available types are *linear* fade and *exponential (decibel linear)* fade.

Cross-fade type is selected by setting the **sampleSwitching.fadeType** parameter in the test config file. Use value ’*linear*’ or ’*exp*’ to select linear or exponential fade. The default is ’*linear*’. Example:

```
# set sample switching cross-fade type
sampleSwitching.fadeType=linear
```

Sample switch cross-fade length

The length of the cross-fade in parallel swithing is selected with the **sampleSwitching.fadeLength** parameter in the test config file. The length of fade is in seconds. The default is 0.04 seconds (40ms). Example:

```
# set sample switching cross-fade length in seconds
sampleSwitching.fadeLength=0.04
```

The fade length was set to 40ms (0.04s).

Notes

Some notes about switching:

- Parallel switching is useful in tests where long samples are compared.
- Parallel switching is only usable for free sequence tests. If fixed sequence is used, parallel switching is not used (even if set to parallel) and normal switching is used.

· **Tests index** · **Document index** ·

Last modified: Mon Jul 20 13:43:27 EEST 1998

Most comfortable listening level

The **most comfortable listening level** (**MCLL** or **MCL level**) is the volume level of the player output that is most corfortable for the subject. The output level can be fixed or the subject can set the level within the limits set by the tester.

Fixed or subject settable listening level

The **MCLL.subjectSetsLevel**-parameter in the parameter file tells can the subject set the level or is it fixed. If set to *true*, the subject sets the output listening level. If *false*, the output level is fixed. Here is an example:

```
# Subject sets MCL level
MCLL.subjectSetsLevel=true
```

Default or fixed listening level

The default of fixed listening level is set with the **MCLL.default**-parameter. If subject can’t set the output level, the level is used as the output level. If subject can set the level, this parameter tells the default or initial volume level that is set as the output level when level setting procedure starts.

An example to set fixed/default output level:

```
# Default output level is -25dB
MCLL.default=-25dB
```

MCL level limits

The available MCL level range for the subject can be limited with the **MCLL.min** and **MCLL.max** parameters. The limits also apply to the default/fixed output level. Here’s an example:

```
# set the available output level range to [-60, 0] dB
MCLL.min=-60dB
MCLL.max=0dB
```

Notes

Some notes about MCLL setting.

- The maximum level that can be set generally shouldn’t be greater than 1.0 (100% or 0dB), overflows may occur (depends on the nature of the samples used).
- Some tests may change the MCLL limits or defaults set in the test parameter file.
- In earlier version of GP there had to be a controller component named ’mcll’ in the subject UI to be able to set the listening level. The component doesn’t have to (should not) any more be added to the controls list. Level setting works now for all tests. The ’mcll’-controller’s parameters can still be used to customize the look of the listening level setting dialog.

· **Tests index** · **Document index** ·

Last modified: Mon Jul 20 16:37:14 EEST 1998

Sound Player Configuration

The examples show examples of how to set player parameters in the test config file.

Configuring players is now slighty different from previous versions. New configuration allows defining multiple sound players and virtual players. If parameter *players* is defined, the new format is used. If not, the old format used in previous versions is used. In the future, the new format will be default, and support for the old format may disappear completely in future GP versions.

The *players* parameter gives the names of players as a comma-separated list. In basic form for basic tests, it sound contain the default player '*player*':

```
# List of players
players=player
```

An example about multiple players can been seen in the virtual players section.

Audio devices

On some systems there are multiple independent audio ports. On a O2 with the digital audio option card there are four output devices: *Analog Out*, *Analog Out 2*, *ADAT Out* and *AES Out*. Different devices may have different choices of sample rates, number of channels, number of bits or output connections. Use '*apanel*' or '*apanel -print*' to list the choices of devices on your system. Also read the manuals.

Choices of devices for O2, Octane and Onyx2 systems include:

Analog Out and **Analog Out 2**
: Analog stereo outputs, 16-bit on O2, 18-bit on Octane an Onyx2. Nearly arbitrary sample rates from 4kHz to 48kHz.

ADAT Out
: 8-channel, 24-bit ADAT Optical output. Sample rates: 32kHz, 44.1kHz, 48kHz.

AES Out
: Stereo AES3 24-bit digital output. Sample rates: 32kHz, 44.1kHz, 48kHz.

in addition the device name **default** can be used that selects the default output device selected with apanel. However, you should not use the defaults device.

Set the device of the player by setting the **device** parameter in the test config file. Example:

```
# set output device
players.player.device=ADAT Out
```

You should use a device that is not the same as the default device. The audio output of tools and programs (includes bells from the console and shells and window manager, web browsers, etc.) goes to the default device if output device is not specially set (usually isn't).

Multiple audio device support

The GP sound player can drive multiple audio devices at the same time. GP combines several devices as a single audio port. Multiple devices are synchronized.

Multiple audio device output can be selected by writing the names of the audio devices to use separated with the slash-character ('/') in the *device* player parameter. For example, to use ADAT cards:

```
# set output device
players.player.device=ADAT Out/ADAT Out 2
```

To the GP system, the output looks like a single 16-channel output port with channels 1-8 going to *'ADAT Out'* device and channels 9-16 to *'ADAT Out 2'* device. Any output devices can be used. Currently, maximum number of audio devices is four.

The same sample rate is used for all devices, so all devices must support the sample rate used.

The number of channels should be left undefined when using multiple devices. The sound player then automatically uses the maximum number of channels for each device.

Sample rate

The sample rates available depends on the audio devices that are used. For example, the analog devices in the O2 support all sample rates from 4kHz to 48kHz with 1Hz resolution. Digital ports only support 32/44.1/48 kHz.

Set sampling rate of the player by setting the **rate** parameter in the test config file. Example:

```
# sampling rate of output
players.player.rate=44100
```

The sample rate of the player and all the samples must be the same. If the sound player can not use some desired sample rate, samples should be converted to a rate supported by the device.

When using multiple devices, the same sample rate is used for all devices.

Channels

The maximum number of channels depends on the audio device that is used. Analog outputs usually support mono or stereo output. Digital outputs support up to 8 channels (depending on which digital port to use). If number of channels is not defined, the maximum number of channels supported by the device is automatically used. Specially, when using multiple devices, the number of channels should be left undefined (or defined as zero).

Set number of channels of the player by setting the **channels** parameter in the test config file. Example:

```
# number of channels (0/1/2/4/8, etc.)
players.player.channels=2
```

If a sample has more channels than there are output channels, extra channels in the sample are ignored.

Sound buffer length

The sound player mixes the output audio data in blocks (or buffer) of fixed size. The output is also double-buffered, the next block of data is calculated as the first is being written to the audio hardware. The buffering also means that an action (for example, the subject presses a button to play a sample) that is meant to take effect immediately will necessarily have a delay that roughly corresponds to two times the length of the buffer. The default buffer length is 4096 sample frames that will cause a delay of about 200ms (sample rate = 44.1kHz). It is possible to shorten the delay by shortening the buffer length. Shortening the buffer length reduces the delay but it also increases the risk of getting dropouts in sound if the buffer calculation didn’t finish in time (other system activity and increased overhead caused by shorter blocks may cause problems, also audio files are loaded from disk on the fly).

The buffer length is set with the **buflen** parameter in the test config file. The length is in sample frames. The length of 44100 corresponds to one second with the sample rate of 44.1kHz. Example:

```
# set mixing buffer length
players.player.buflen=4096
```

Sets the buffer length to 4096 sample frames (roughly 93ms with sample rate 44.1kHz).

Note: If a command was scheduled beforehand to start at a certain point of time, no delay is observed (sub-millisecond accuracy).

Virtual players

Virtual players allow partitioning the output port the the GP into smaller sections. For example, an eight-channel ADAT output could be sectioned into four stereo-players. Virtual players behave the same way as real players except some restrictions. Currently, 16 virtual players can be defined.

Configuration is similar to the configuration shown earlier. Usually, the channels to use for the VP and the ’*parent*’ player of the VP. For example:

```
# List of players
players=player,vp1,vp2,vp3

# Main player
players.player.device=ADAT Out
players.player.rate=44100

# Mono player, a single channel: channel 0
players.vp1.channel=0
players.vp1.parent=player

# Stereo player, channel range: channels 1-2
players.vp2.channel=1-2
players.vp2.parent=player

# 5-ch player, channel range: channels 3-7
players.vp3.channel=3-7
players.vp3.parent=player
```

The *players* parameter contains the list of players used for the test. First is the default player ’*player*’.

Virtual players (vp1-3) are virtual players. Each must define the *parent* parameter that sets which real player's channels to use. The *channel* parameter defines which channels of the parent to use for this VP. A single channel player (vp1) can be defined with the index of the channel. For players with more channels, a range of channels is specified. Note that the indexing of channel numbers starts from zero.

Multiple audio devices can also be used with virtual players. For example, to add another virtual player with 8 channels (only the changes and additions to the example above are shown):

```
# List of players
players=player,vp1,vp2,vp3,vp4

# Use multiple audio devices
players.player.device=ADAT Out/ADAT Out 2

# ... parameters for VPs 1-3 ...

# 8-ch player, channel range: channels 8-15
players.vp4.channel=8-15
players.vp4.parent=player
```

Last modified: Mon Jul 26 17:52:57 EEST 1999

Logging

GuineaPig saves the diagnostics and error messages it generates when running a test. By default, the messages are saved to file '*session.log*' in the test directory. If the file already exists, new information will be appended to the file (does not overwrite an existing log file). Also the verbosity level of messages that is included can be selected.

The logging (verbosity) level is set with the **logLevel** parameter in the test configuration file. For example:

```
# Verobosity level
logLevel:   debug
```

The log levels available are: **silent**, **brief**, **normal**, **verbose**, **veryverbose**, and **debug** (in order of from least verbose (*silent*) to most verbose (*debug*)). The default is *normal*. Level *silent* only prints error messages.

It is also possible to define a different logging level for console output with the **logConsoleLevel** parameter. For example:

```
# Log level for file
logLevel:        debug
# Log level for console
logConsoleLevel:  brief
```

In this example all messages are saved to a file but only *brief* logging is printed on console. The console logging level must not be more verbose than file logging level.

The file where to save the log can be selected with the **logFile** parameter:

```
# File where to log
logFile:   session.log
```

The file name is relative to test directory unless absolute path is used. In the future, logging may be possible to automatically to use a session ID based log file name.

Last modified: Tue Jul 27 12:46:52 EEST 1999

Subject’s User Interface

- **Creating a subject user interface**
- Subject UI parameters
- **Question components**
    - **GradeBar -** a grade from a scale.
        - **FiveGrade -** a grade bar for the ITU-R five-grade impairment scale given in *Recommendation ITU-R BS.562*.
        - **TenGrade -** a grade bar for giving a grade for ’clarity’ of a sample.
        - **VolumeGrade -** a grade bar that gives volume level values instead of numbers.
    - **CheckboxChoice -** Choose an answer.
    - **RankOrder** - give rankings for a bunch of objects.
- **Control components**
    - **PlayPanel -** a controller to play samples. Also acts as a monitor that shows which sample is currently playing.

· **Document index** ·

Last modified: Thu Jan 21 16:33:05 EET 1999

UI parameters

Here is a list of basic parameters for the subject UI windows:

Parameter	Value	Description	R/O
class	guinea.ui.ABSubjectUI	Java class name of the subject UI object.	Req.
title	*text*	The title of the window.	Opt.
questions	*list*	A comma-separated list of question component names.	Opt.
controls	*list*	A comma-separated list of control component names.	Opt.
isResizable	*true* of *false*	Is the window resizable? Default is *true*.	Opt.
width	*number of pixels*	The width of the window in pixels. If this is non-zero, the window's width is fixed. Default is 0 (window's width is determined automatically).	Opt.
height	*number of pixels*	The height of the window in pixels. If this is non-zero, the window's height is fixed. Default is 0 (window's height is determined automatically).	Opt.
showTestProgress	*true* of *false*	Whether to display a test progress monitor component to the subject that shows how many test items have been done.	Opt.

Title of the UI window

The **title** parameter is used to set the title of the UI window. Example:

```
# Set window title
title=ABC Test
```

Questions

The **questions** parameter lists which question components are used. The list is a comma-separated list of question IDs, for example:

```
# List of question objects to be used
questions=gA,gB
```

Two question components with IDs '*gA*' and '*gB*' are used (grade for samples A and B). The parametars for those questions must be included in the UI file. If no questions are used, do not define the questions-parameter at all.

Controls

The **controls** parameter lists which control components are used. The list is a comma-separated list of control IDs, for example:

```
# List of control objects to be used
controls=play
```

One control component with ID ’*play*’is used (for subject to select which sample to play and to show which sample is playing). The parametars for those controls must be included in the UI file. If no controls are used, do not define the controls-parameter at all.

Size of window

Normally the window’s size is determined automatically by Java’s window toolkit and its layout managers. The size of the window depends on how many question and control components there are, the length of the text labels and their font sizes.

The window’s size can be fixed with the **width** and **height** parameters. By default they both are zero which means the size is determined automatically by the Java’s layout manager. Example:

```
# Set window’s width to 640 pixels
width=640
```

This will set the window’s width to 640 pixels. The height will be determined automatically. The window’s width will also cause all question and control components to that width.

By default the user can resize the subject UI window with the mouse. This can be prevented by setting the **isResizable** parameter to *false*. For example:

```
# Do not allow the subject to resize the window.
isResizable=false
```

the **isResizable** parameters works both when window’s size is fixed or automatic.

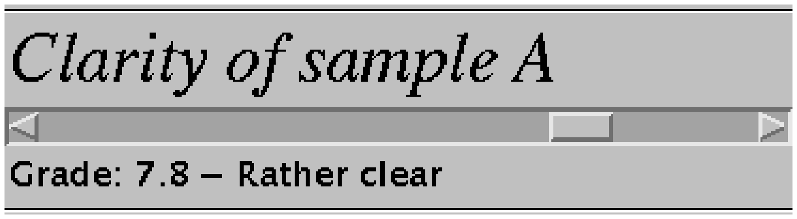
· **UI index** · **Document index** ·

Last modified: Fri Nov 13 14:26:41 EET 1998

GradeBar

The **GradeBar** is used to give a grade as an answer to a question. The scale that is used can be specified with any limits and number of decimals. Adjectives may be associated with certain ranges of values.

Here is an example of a grade bar with range from 0.0 to 10.0 with one decimal and adjectives:



Here is a list of parameters available for a grade bar:

Parameter	Value	Description	R/O
class	GradeBar	Class alias for a grade bar component.	Req.
question	*text*	The question shown to subject.	Req.
minimum	*number*	The minimum of the scale. Default is 0.0.	Opt.
maximum	*number*	The maximum of the scale. Default is 5.0.	Opt.
decimals	*number of decimals*	Number of decimals in the grade. Default is 1.	Opt.
questionfont	*font name*	The font used for question text.	Opt.
showValue	*true* or *false*	Whether to show the current value to the subject. Default is *true*.	Opt.
showLabels	*true* or *false*	Whether to show the adjectives to the subject. Default is *true*.	Opt.
choiceformat	*choice format string*	Give adjectives to certain ranges of values. Default is no adjectives, only the value is shown.	Opt.
defaultAnswer	*number*	Set the initial value (and the position) of the grade bar when the question is initialized or reset. A special value '`random`' can be used to initialize the value with a new random number every time the component is reset. If this option is left out, the default answer is the midpoint of the minimum and maximum of the scale.	Opt.

Question

The **question** parameter is used to set the question that is shown to the subject. Example:

```
# Set question text
qA.question=Clarity of sample A
```

The scale of the grade

The range of grades that the subject can give is specified with the **minimum**, **maximum** and **decimals** parameters. Both minimum and maximum can be either positive or negative but minimum should be less than the maximum. If integer answer values are wanted, set decimals to zero.

The following example sets the scale to from 0 to 5 with one decimals. The question name will be ’qA’:

```
# Set range of grade to [0.0, 10.0] (one decimal).
qA.minimum=0.0
qA.maximum=10.0
qA.decimals=1
```

Question font

The font used to display the text of question can be set with the **questionfont** parameter. It takes a valid Java font specification as a value. For example:

```
# The font of the question
qA.questionfont=Serif-italic-24
```

You can use the **FontTester** tool help you select the fonts you want. See also the API of the Java’s Font class.

Show value to subject?

By default the value of the grade bar is shown to the subject as he/she moves the scrollbar. Set the **showValue** parameter to *false* to not show the value to the subject. Example:

```
# Do not show grade to subject
qA.showValue=false
```

If showValue is set to *false*, only the adjectives (see below) will be shown if they are set.

Associate adjectives to range of values

Adjectives can be associated with ranges of values. It is done by setting the **choiceformat** parameter. More information about the choice format can be found in the Java’s ChoiceFormat API. The following example should help you to make your own choiceformats:

```
# Set adjectives
qA.choiceformat=0.0#Very unclear|2.0#Rather unclear|4.0#Midway|6.0#Rather clear|8.0#Very Clear
```

This string tells that the adjective ’Very unclear’ will be shown if the value X is between 0.0 and 2.0 (0.0 <= X < 2.0), ’Rather Unclear’ will be shown if 2.0 <= X < 4.0, and so on. If X < 0.0 the first adjective will be shown and if X > 8.0 the last adjective is shown.

For adjectives to be shown, the **showLabels** parameter must be set to `true` (it is true by default).

Default/initial answer

By default the grade bar’s knob is set to the middle of the minimum and maximum of the range when the component is reset. Also a *fixed* or *random* value can be used. To use a fixed value as the initial answer or the position of the knob, set the **defaultAnswer** parameter to desired value. For example, to use value 1.5 as initial value:

```
# Use initial value 1.5
qA.defaultAnswer=1.5
```

The position of the knob is set to the position that matches that value on the range.

To automatically generate a random initial value when the component is reset, set **defaultAnswer** to string ’*random*’:

```
# Use random values for initial value
qA.defaultAnswer=random
```

It generates a new random value between the minimum and maximum of the grade bar’s range.

It is also possible to limit the range of the initial random value to a smaller range than minimum and maximum of the component. Use the **defaultAnswer.randomMin** and **defaultAnswer.randomMax** parameters to set different limits. For example (lets assume the component’s answer value range is [0.0, 10.0]):

```
# Use random initial values with values between [3.0, 7.0]
qA.defaultAnswer=random
qA.defaultAnswer.randomMin=3.0
qA.defaultAnswer.randomMax=7.0
```

· **UI index** · **Document index** ·

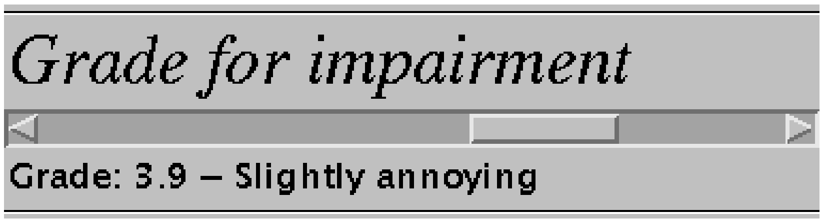
Last modified: Thu Jan 21 17:21:12 EET 1999

FiveGrade

The **FiveGrade** UI component implements a continuous grading scale with "anchors" derived from the ITU-R five-grade impairment scale given in *Recommendation ITU-R BS.562* as shown below.

```
Impairment                    Grade
Imperceptible                 5.0
Perceptible, but not annoying 4.0
Slightly annoying             3.0
Annoying                      2.0
Very annoying                 1.0
```

One decimal place is used. Here is an example of a five-grade bar with range from 1.0 to 5.0 with one decimal and adjectives:



The FiveGrade component is derived from the **GradeBar** component. It simply defines the the scale from [1.0,5.0] with one decimal and the adjectives. Equivalent component can be done with a GradeBar only. Here is a list of parameters that are used for the five-grade bar:

Parameter	Value	Description	R/O
class	guinea.ui.FiveGrade	Java class name of five-grade bar.	Req.
question	*text*	The question shown to subject.	Req.
questionfont	*font name*	The font used for question text.	Opt.
showValue	*true* or *false*	Whether to show the current value to the subject. Default is *true*.	Opt.

Example of defining a FiveGrade component

Here is an example of how to implement the FiveGrade bar shown above:

```
# Example of a FiveGrade question component
q2.class=guinea.ui.FiveGrade
q2.question=Grade for impairment
q2.questionfont=Serif-italic-24
```

Implementing a FiveGrade with a GradeBar

The FiveGrade can be also implemented with the **GradeBar** only. Here is a sample how to create an equivalent component that was shown above:

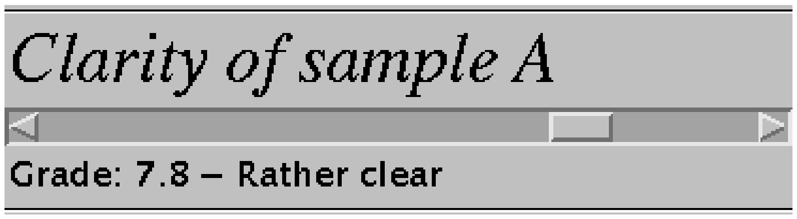
```
# Example of emulating a FiveGrade with a GradeBar
q2.class=guinea.ui.GradeBar
q2.question=Grade for impairment
q2.questionfont=Serif-italic-24
q2.minimum=1.0
q2.maximum=5.0
q2.decimals=1
q2.choiceformat=1.0#Very annoying|2.0#Annoying|3.0#Slightly annoying|4.0#Perceptible, but not annoying|5.0#Imperceptible
```

· UI index · Document index ·

Last modified: Thu Jul 23 14:36:27 EEST 1998

TenGrade

The **TenGrade** is an example of using GradeBar as a ten-grade answering component. The scale goes from 0 to 10 with one decimal and shows the grade symbolicly also ("Very unclear", "Rather unclear", "Midway", "Rather clear", "Very Clear"). Here is an example of a ten-grade component.


Clarity of sample A

Grade: 7.8 – Rather clear

The FiveGrade component is derived from the **GradeBar** component. It simply defines the the scale from [0.0,10.0] with one decimal and the adjectives. Equivalent component can be done with a GradeBar only. Here is a list of parameters that are used for the five-grade bar:

Parameter	Value	Description	R/O
class	guinea.ui.TenGrade	Java class name of ten-grade bar.	Req.
question	*text*	The question shown to subject.	Req.
questionfont	*font name*	The font used for question text.	Opt.
showValue	*true* or *false*	Whether to show the current value to the subject. Default is *true*.	Opt.

Example of defining a TenGrade component

Here is an example of how to implement the TenGrade bar shown above:

```
# Example of a TenGrade question component
q2.class=guinea.ui.TenGrade
q2.question=Clarity of sample A
q2.questionfont=Serif-italic-24
```

Implementing a FiveGrade with a GradeBar

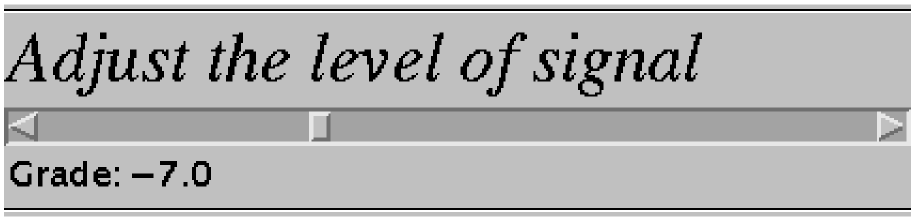
The FiveGrade can be also implemented with the **GradeBar** only. Here is a sample how to create an equivalent component that was shown above:

```
# Example of emulating a TenGrade with a GradeBar
q2.class=guinea.ui.GradeBar
q2.question=Clarity of sample A
q2.questionfont=Serif-italic-24
q2.minimum=0.0
q2.maximum=10.0
q2.decimals=1
q2.choiceformat=0.0#Very unclear|2.0#Rather unclear|4.0#Midway|6.0#Rather clear|8.0#Very Clear
```

Last modified: Thu Jul 23 14:52:30 EEST 1998

VolumeGrade

The **VolumeGrade** is an extension to the normal GradeBar which given numbers as answers. This version gives Volume objects as answers by converting GradeBar's numeric answers to Volume type in selected scale. Here is an example of a volume grade component.



The VolumeGrade component is derived from the **GradeBar** component. All parameters are the same as in the GradeBar except these differences and additions:

Parameter	Value	Description	R/O
class	VolumeGrade	Volume grade's class alias name.	Req.
scaleType	*scale's type*	The volume scale to use. Possible values are: *linear*, *decibel* and *percent*. By default decibel scale is used.	Opt.

The numeric minimum and maximum values and other values are in the selected scale.

Volume scale to use

By default the volume grade uses *decibel* scale. You can also use *linear* and *percent* scales. Select the scale to use by setting the **scaleType** parameter to either *decibel*, *linear* or *percent*. For example:

```
# Use linear scale
level.scaleType=linear
```

Other parameters

Other parameters work just like in a regular GradeBar. They just operate on the numeric values instead of volume objects. To set minimum and maximum values, use plain numeric values and set the scale type as you want. For example to get volume answers from -20dB to 10dB in decibel scale, define min/max parameters like this:

```
# Get decibel values from range [-20, 10] dB with one decimal
level.minimum=-20
level.maximum=10
level.decimals=1
```

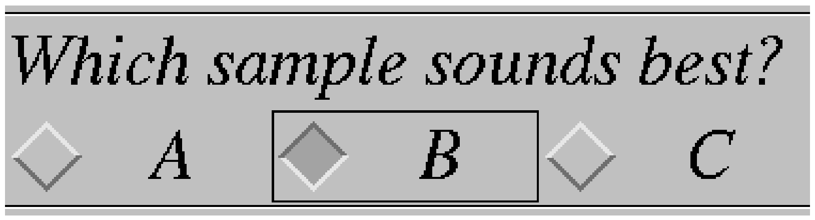
Also adjectives can be used as well as 'default answer' including random initial value.

Last modified: Thu Jan 21 16:52:13 EET 1999

CheckboxChoice

The **CheckboxChoice** is used to select one of multiple choices as an answer to a question. Answers are user-defined and the corresponding labels shown to the subject can be set.

Here is an example of a checkbox choice with three choices:



Here is a list of parameters available for a checkbox choice:

Parameter	Value	Description	R/O
class	guinea.ui.CheckboxChoice	Java class name of checkbox choice	Req.
question	*text*	The question shown to subject.	Req.
questionfont	*font name*	The font used for question text.	Opt.
answers	*list of answers*	The answers sent by the answer component when the corresponding label is selected. The *list* is a comma separated list of answers (strings only currently).	Req.
labels	*list of labels*	The choices (labels) shown to the subject. The subject selects one of the choices and the corresponding answer is sent. The *list* is a comma separated list of strings that are shown as labels.	Opt.

Question

The **question** parameter is used to set the question that is shown to the subject. Example:

```
# Set question text
q5.question=Which sample sounds best?
```

Question font

The font used to display the text of question can be set with the **questionfont** parameter. It takes a valid Java font specification as a value. For example:

```
# The font of the question
q5.questionfont=Serif-italic-24
```

You can use the **FontTester** tool help you select the fonts you want. See also the API of the Java’s Font class.

Answers and labels

The **answers** list defines the answers are sent by the answer component, the **labels** list defines the labels that are shown to the subject. Both are comma-separated lists of strings. Example:

```
# Set answers (labels are same as answers)
q5.answers=A,B,C
```

This sets the answers and labels as shown in the window example above. If no **labels** are specified, the answers will be used as labels also. In this case it acts the same as this:

```
# Set answers and corresponding labels
q5.answers=A,B,C
q5.labels=A,B,C
```

Labels are used for displaying the selections to the subject. Example:

```
# Set answers and corresponding labels
q5.answers=A,B,C
q5.labels=First,Second,Third
```

the labels shown are ’First’, ’Second’ and ’Third’ instead of ’A’, ’B’ and ’C’. Then the subjects selects ’Second’, ’B’ will sent to the test system as the answer.

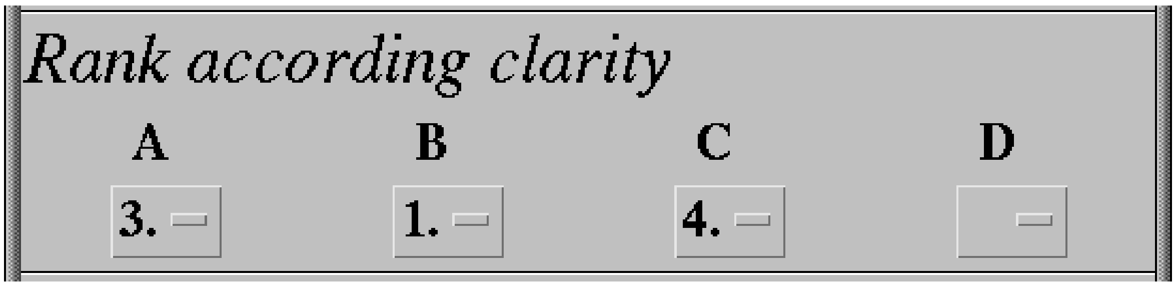
· **UI index** · **Document index** ·

Last modified: Mon Jul 27 14:32:39 EEST 1998

Rank Order

The **RankOrder** is used to select one of multiple choices as an answer to a question. Answers are user-defined and the corresponding labels shown to the subject can be set.

Here is an example of a rank order component with four labels to rank:



Here is a list of parameters available for a rankorder component:

Parameter	Value	Description	R/O
class	RankOrder	Java class name alias a rankorder component.	Req.
question	*text*	The question shown to subject.	Req.
questionfont	*font name*	The font used for question text.	Opt.
labels	*list of labels*	The list of labels that are ranked.	Req.
labelsfont	*font name*	The font used for labels and their popup-menus	Opt.
allowTies	*boolean*	Whether to allow/disallow ties in ranking. Default is to allow ties (*true*).	Opt.
allowIncompleteRanking	*boolean*	Whether to allow/disallow incomplete ranking (not all labels have been ranked) as a valid answer. Default is incomplete ranking is allowed (*true*).	Opt.

Question

The **question** parameter is used to set the question that is shown to the subject. Example:

```
# Set question text
q5.question=Rank according clarity
```

Labels that are ranked

The **labels** parameter contains a list of labels that are ranked. The list is comma-separated lists of strings. Example:

```
# Set labels to rank.
q5.labels=A,B,C,D
```

A pop-up menu is created for each label. The pop-up menu is used to select the ranking to that label. There are as many choices as there are labels. For example, if four labels *’A,B,C,D’* are used, the pop-up menu has choices *1.*, *2.*, *3.*, and *4.* and also an empty choice (rank not given yet or left empty).

Ties

By default the rank order allows ties (same rank given for multiple labels). You can turn this off by setting the **allowTies** parameter to *false*. Example:

```
# disallow ties
q5.allowTies=false
```

If ties have been disabled, when a illegal ranking is choosed to a label (there is already a label with that ranking), the pop-up reverts to its previous ranking.

Incomplete ranking

By default the rank order components sends an answer whenever a ranking choice is made for any label (unless possibly a tie is selected when ties has been disabled). The answers are generally incomplete until a rank has been given for all labels. To send an answer only when all labels have been ranked, set the **allowIncompleteRanking** to *false*. For example:

```
# send answer only when all labels have been given a rank
q5.allowIncompleteRanking=false
```

If the ranking is changed after a complete answer has been sent, the answer may become incomplete. In that case the component sends a null answer which removes the previous answer and leaves that component in an unanswered state (may disable DONE-button in the subject window because all questions haven’t been answered).

Question font

The font used to display the text of question can be set with the **questionfont** parameter. It takes a valid Java font specification as a value. For example:

```
# The font of the question
q5.questionfont=Serif-italic-24
```

You can use the **FontTester** tool help you select the fonts you want. See also the API of the Java’s Font class.

Labels font

The font used in the labels and pop-up menus used to rank labels is set with the **labelsfont** parameter. It takes a valid Java font specification as a value. For example:

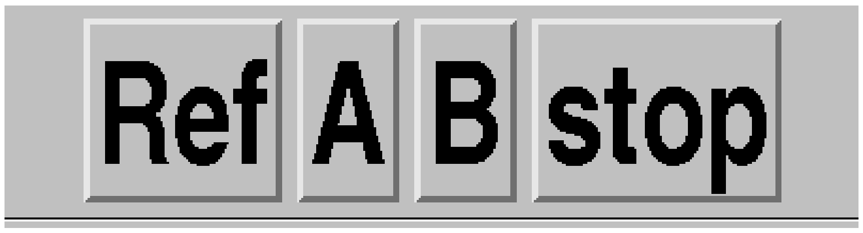
```
# Set font for labels and pop-up menus
q5.labelsfont=Serif-bold-18
```

Last modified: Wed Jan 20 16:06:31 EET 1999

PlayPanel

The **PlayPanel** component is a *control* component that is used by the subjects to play samples. It also acts as a *monitor* that can show which sample is currently playing. Possible choices are user-definable and the corresponding labels shown to the subject can be set. Also a ’stop’ button is added to stop sample playing (some tests may ignore this button).

Here is an example of a play panel choice with three choices:



Here is a list of parameters available for a play panel controller:

Parameter	Value	Description	R/O
class	guinea.ui.PlayPanel	Java class name of the PlayPanel component	Req.
font	*font name*	The font used for buttons.	Opt.
choices	*list of choices*	The choicess sent by the play component when the corresponding label is pressed. The *list* is a comma separated list of choices (strings only currently).	Req.
labels	*list of labels*	The labels shown to the subject. The subject selects one of the choices and the corresponding choice is sent. The *list* is a comma separated list of strings that are shown as labels.	Opt.

Buttons font

The font used to display the text of choices can be set with the **font** parameter. It takes a valid Java font specification as a value. For example:

```
# The font for the buttons
play.font=SansSerif-bold-48
```

You can use the **FontTester** tool help you select the fonts you want. See also the API of the Java’s Font class.

Choices and labels

The **choices** list defines the choices of commands that are sent to the test system. In all tests the choices are sample names such like "A", "B" or "Ref" that refer to the corresponding test item parameters. The **labels** list defines the labels that are shown to the subject on the buttons on the UI. Labels can be used to display a different label that corresponds to a specific command. Both are comma-separated lists of strings. Example:

```
# Set choices (and labels)
play.choices=Ref,A,B
```

This sets the choices and labels as shown in the window example above. If no **labels** are specified, the choices will be used as labels also. In this case it acts the same as this:

```
# Set choices and corresponding labels
play.choices=Ref,A,B
play.labels=Ref,A,B
```

Labels are used for displaying the selections to the subject. Example:

```
# Set choices and labels
play.choices=Ref,A,B
play.labels=Reference,System A,System B
```

the labels shown are ’Reference’, ’System A’ and ’System B’ instead of ’Ref’, ’A’ and ’B’. Then the subjects selects ’System A’, ’A’ will sent to the test system as the command.

Last modified: Thu Jan 21 16:11:41 EET 1999

Results Processing

The role of the result processing GuineaPig system is to gather the data from individual test sessions (parameters of the test, answers given by the subjects) and output the data in a way that analytical tools can read them. The actual analysis of results is not a part of the GuineaPig system.

Internally the results are stored using Java’s serialization system. The results contains the test item parameters, the answers the subjects have given and additional information about the particular session.

- About **result processing** in GuineaPig system.
- **ResPrint -** tool to convert the results from the internal system format to a text file.
- **Customization** of the output of results printing with config files.
- Formatting different kind of answer types (*details will be added later*):
  - **Numbers**
  - **Dates**
  - **Rank orders**
  - **Volume levels**

· **Document index** ·

Last modified: Sun Aug 1 19:07:28 EEST 1999

Results Processing in GuineaPig System

Results created by the GuineaPig system are in a format different from what the usual statistical sofware packages expect. A small converter program is provided to convert result data into a text file.

The **ResPrint** tool will convert result files from the GuineaPig system's internal format to a tabulated text file. It will read the session results files given as arguments, scans their contents and prints out a text file to standard output. For each entry (test item), it will print (by default):

- **Item ID -** the test item ID.
- **Subject ID -** the subject ID of the person who gave the answers.
- **Session ID -** the session ID of the session.
- **Duration of ihem -** how much time was spent for this item from starting of this item to the time the subject was done. The time is shows as seconds. For tests that enforce a time limit for answering this field may read TIMEOUT which tells that the subject exceeded the time limit.
- **Number of switches -** the number of times the subject has switched samples during an item (free sequence test) or number of samples played to the subject (fixed sequence tests).
- **Item parameters -** the test item's parameters, usually the names of the samples played (sample IDs of the samples, actually).
- **Answers -** answers to the questions. If an answer for a particular question is missing, it's field will be left blank.

In the beginning of the printout, a comment will be printed that shows which column is which. For result files that contain session information (serialized result files from tests), comments are printed that contain the session ID, starting and ending time of the session and the MCL level of the session.

Running the converter

First go to the test directory where the result files are kept. Then run the converter program and give the file names of the session result files as agruments, for example:

```
    gpResPrint results_*
```

if your result file names begin with '*results_*'. The converter then scans in the files given as arguments and prints out the results in a tabulated table.

For more details about ResPrint tool, see **ResPrint manual page**.

Program output

Program output format depends slighty on the test type (different parameters and questions). For example output of the ResPrint program processing a A/B/C-test (actually a Ref/A/B) could look like this:

```
#item   subject session time    switch  A       B       Ref     qB      qA
item4   ville   zappa   8.7     7       pirr11  pirr16  pirr16  7.0     2.0
item3   ville   zappa   14.2    12      pirr8   pirr11  pirr8   2.0     8.0
item2   ville   zappa   8.5     9       pirr22  pirr32  pirr32  2.6     7.6
item1   ville   zappa   10.0    7       pirr44  pirr32  pirr44  7.8     9.4
```

Here is another example of a A/B scale test and session information. Also shows an example of automatically generated session and subject IDs if IDs were not set.

```
#session id: S14837061
#session start time: Wed Mar 18 14:21:55 GMT+02:00 1998
#session end time: Wed Mar 18 14:25:12 GMT+02:00 1998
#session MCLL: -3.9999999999999999dB
#item   subject         session         time    switch  A       B       q2      q1
item4   S14837061a      S14837061       130.2   25      pirr11  pirr16  5.3     5.2
item3   S14837061a      S14837061       27.6    13      pirr8   pirr11  6.0     3.0
item2   S14837061a      S14837061       4.9     3       pirr22  pirr32  7.0     3.0
item1   S14837061a      S14837061       24.8    11      pirr44  pirr32  6.0     4.0
```

Output can be directed to a file or piped to another program with normal unix syntax.

Comment lines start with the ’#’ character. Each field will be separated with a single ASCII TAB-character. White space other than TAB-characters are not considered field separators.

The output of the tool can be customized with result options file(s) and tools options.

Note!

You might have noticed that the order of the answers (question IDs) in the result file may not be the same as they appear on the test configuration files. Also, the order of the answers (for example: [qA, qB]) in the result file produced from a session1 may be different from the result file produced by another session2 ([qB, qA]). For example:

```
gpResPrint session1.ser >res
gpResPrint session2.ser >>res
```

The order of answers may be different in session1’s result from session2’s results. This is due to internals of the system where results are stored in an item as a hash table which is not ordered.

The problem occurs only when you haven’t defined manually the order of fields to be printed by the tool (either with command line option or result options file). To avoid this problem, define the fields to be printed and their order in the results config file with the *fields* property or with the -fields option in ResPrint.

· **Results Processing Index** · **Manual index** ·

Last modified: Sun Aug 1 19:09:42 EEST 1999

Results processing configuration

The output of the **ResPrint** tool can be customized with command line options and config files. When ResPrint starts, it loads the default results processing options from file '*/usr/GuineaPig/lib/results.properties*'. Then it loads the file '*results.properties*' from the current directory if it finds one. Then an additional config file is loaded if one in provided with the ResPrint's -f option.

The config files allow you to configure which fields are printed and which order. The format used to print the values of a field can be customized with some built-in variations or with additional plug-in modules. Also simple filtering of item results is possible.

Empty lines and comment lines (lines beginning with a hash ("#") character) are ignored.

Fields to print and their order

The **fields** parameter lists the field IDs of the information about an item and their order. The selection of fields is provided as a comma-separated list of field IDs, for example:

```
# Print these fields
fields:   ITEMID,SUBJECTID,SESSIONID,TIME,SWITCH,A,B,q1
```

This would print the *item ID*, *subject ID*, *session ID*, *item duration*, *number of sample switches*, parameters *A* and *B* and the answer to question *q1*.

The fields are either **built-in fields**, *parameter fields* or *answer fields*. Built-in fields include such fields as *item ID*, *item duration*, etc. Parameter fields are the parameters of an item, such as 'A' or 'B' or 'Ref'. Answer fields are the answers given for the questions specified in the UI config file.

Built-in fields

Here is the list of built-in fields:

ITEMID
The item's item ID (a string).

SUBJECTID
The subject ID of the subject that gave the answers (a string).

SESSIONID
The session ID of the session during which this item was presented (a string).

ITEMSTART
The date and time when item was presented to the subject (a java's Date object).

PLAINTIME
The duration of the item (in seconds), how much time the subject used to grade the samples.

STATUS
The status of testing this item. Possible values are:

DONE
Test item was tested succesfully.
TIMEOUT
Item grading time limit expired when testing this item.
ABORTED
The subject aborted the test session during testing this item.
ERROR
A testing system error occured during this item.
NOTDONE
This item has not been tested. (Generally, this should not be possible when printing results.)

TIME
A combination of PLAINTIME and STATUS fields. If the item was succesfully tested, this field shows the duration of the item (how much time the subject used to grade the samples). By default, the time is shown in seconds with one decimal. If something special happened during testing this item, the information from STATUS field is shown instead of time (usually TIMEOUT indicating that item grading limit expired).

SWITCH
Number of sample switches during testing this item (an integer).

SESSIONSTART
The time when this session started (a java's Date object).

SESSIONEND
The time when this session ended (a java's Date object).

MCLL
The MCL level of this item's session (a GP's Volume object).

Test item filtering

With the filtering parameters or the ResPrint's filtering options you can select only a subset of all item results to be printed. If several filters are specified, the intersection of the sets defined by different filters is used (filters are 'and'ed together).

Select items: the **items** property is a comma-separated list of *item IDs* to print.

```
# print only items item01, item03 and item05
items:   item01,item03,item05
```

Only answers with given item IDs are printed.

Select subjects: the **subjects** property is a comma-separated list of *subject IDs* to print.

```
# print only answers given by subjects with subject IDs john, jane and doe.
subjects:   john,jane,doe
```

Only answers with subject IDs given are printed.

Select sessions: the **sessions** property is a comma-separated list of *session IDs* to print.

```
# print only answers from sessions SES04, SES05, SES06.
sessions:   SES04,SES05,SES06
```

Only answers with session IDs given are printed.

If a filter is not defined for items, all items are included in printing. The same goes for subjects and sessions filters.

You can use several types of filters, for example:

```
# print only answers for item01, item03 and item05 given during
# sessions SES04, SES05 and SES06.
items:      item01,item03,item05
sessions:   SES04,SES05,SES06
```

Field customization

The format of all the fields can be customized. Properties for fields are searched from parameters that look like this:

```
fields.TIME.label:    Time/s
```

This example sets the label for the built-in TIME field. To choose a label ’System A’ for item parameter A you would do it in the same fashion:

```
fields.A.label:    System A
```

All fields have some standard properties. In the following, the FIELDID is the field ID name you want to customize.

The **field label** is the string that is shown in the header in the place of this field. Use property *label* to set it:

```
fields.FIELDID.label:     field label
```

The ’*field label*’ is shown in the header. If label is not specified, the field’s ID is used as the label also.

The **format** used to display the field’s contents can be specified. You’ll need to provide a java class name of a formatter object with the *format* property:

```
fields.FIELDID.format.class:    java_class_name
```

where *java_class_name* is the class name of the object you want to use for formatting. The formatter object must be a subclass of java’s Format class. By default GP includes support for the most common format classes: numeric values and dates.

Formatters usually (numbers, dates) has a **pattern** that can be customized to change the output of the formatter. It is usually set with the *pattern* property of the field’s format properties:

```
fields.FIELDID.format.pattern:     formatting_pattern
```

See the documentation of the formatter for info about its pattern specs. *More information about built-in fields formatting will be added later.*

Defaults for builtin fields can be found from /usr/GuineaPig/lib/results.properties .

Sub-fields

Sub-fields allows you to split the information form one field to multiple fields. For example, you can split the starting time of the item (ITEMSTART) to deparate time and date fields. The format of the field ID of a sub-field is quite simple. For example, to define a date sub-field of the ITEMSTART parameter, the ID could be:

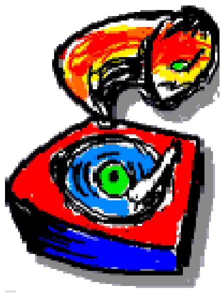
```
fields:    ITEMID,SUBJECTID,ITEMSTART/date,A,B,...
```

The slash ('/') marks that this is a sub-field ID. The value that is formatted is got from the ITEMSTART parameter, the field ID that is in the left side of the slash character. The right side is not important, you may use anything you want. To customize the format of the new sub-field, just customize it the same as with any other field, for example:

```
# Subfield ITEMSTART/date (The date part of the ITEMSTART field)
#
fields.ITEMSTART/date.format.class:   guinea.tools.resprint.GPSimpleDateFormat
fields.ITEMSTART/date.format.pattern: dd/MM/yyyy
```

This would format the date 'Mon Jan 18 14:42:59 EET 1999' as '*18/01/1999*'. With more sub-fields you could split the day, month, year to each their own fields.

Sound Player Module



The **Sound Player** handles the audio output of the system. Available documentation is listed below:

- **Audio file formats** the player accepts.
- **Parameters** of player.
- SoundPlayer API

· Document index ·

Last modified: Mon Apr 20 15:48:28 EET DST 1998

Sound File Formats

The sound player uses the SGI’s *Silicon Graphics Audio File Library (AF)* to read the audio files. Therefore all file formats supported by the AF can be used with the GuineaPig system. The list of supported formats include: AIFF-C, AIFF, SND/AU, WAVE, MPEG1, etc. See the manual page of ’*afIntro*’ for the list of supported formats and more information about the formats. The sound player currently doesn’t support headerless raw data files.

The preferred file formats are uncompressed AIFF and AIFF-C. Conversion from and to other formats can be done, for example, with *mediaconvert*, *dmconvert* and *sfconvert*.

· **Sound Player Index** · **Document index** ·

Sound Player Parameters

The examples show examples of how to set player parameters in the test config file.

Sample rate

The sample rates available depends on the audio device that is used. For example, the analog devices in the O2 support all sample rates from 4kHz to 48kHz with 1Hz resolution. Digital ports only support 32/44.1/48 kHz.

Set sampling rate of the player by setting the **rate** parameter in the test config file. Example:

```
# sampling rate of output
player.rate=44100
```

The sample rate of the player and all the samples must be the same. If the sound player can use some sample rate, samples should be converted to a supported rate.

Channels

The maximum number of channels depends on the audio device that is used. Analog outputs usually support mono or stereo output. Digital outputs support up to 8 channels (depending on which digital port to use).

Set number of channels of the player by setting the **channels** parameter in the test config file. Example:

```
# number of channels (1/2/4/8)
player.channels=2
```

If a sample has more channels than there are output channels, extra samples are ignored.

Audio device

On some systems there are multiple independent audio ports. On a O2 with the digital audio option card there are four output devices: *Analog Out*, *Analog Out 2*, *ADAT Out* and *AES Out*. Different devices may have different choices of sample rates, number of channels, number of bits or output connections. Use '*apanel*' or '*apanel -print*' to list the choices of devices on your system. Also read the manuals.

Choices of devices for O2, Octane and Onyx2 systems include:

Analog Out and **Analog Out 2**
: Analog stereo outputs, 16-bit on O2, 18-bit on Octane an Onyx2. Nearly arbitrary sample rates from 4kHz to 48kHz.

ADAT Out
: 8-channel, 24-bit ADAT Optical output. Sample rates: 32kHz, 44.1kHz, 48kHz.

AES Out
: Stereo AES3 24-bit digital output. Sample rates: 32kHz, 44.1kHz, 48kHz.

in addition the device name **default** can be used that selects the default output device selected with apanel. However, you should not use the defaults device.

Set the device of the player by setting the **device** parameter in the test config file. Example:

```
# set output device
player.device=ADAT Out
```

You should use a device that is not the same as the default device. The audio output of tools and programs (includes bells from the console and shells and window manager, web browsers, etc.) goes to the default device if output device is not specially set (usually isn't).

Output volume

The output volume level offset or correction is used to set the output level. It can be used to adjust the volume of digital outputs (there is no level control for digital outputs as there are for analog outputs in apanel).

In earlier versions of the GuineaPig system the player's volume parameter was used to set the default listening level for the test (the MCL level). In current and future versions of the system use the MCLL-parameters in the test configuration file. Do not use the player's volume level for that purpose anymore.

Set the volume of the player by setting the **volume** parameter in the test config file. The volume level is in linear scale (usually between 0 and 1) or in decibel scale. Example:

```
# set volume offset (linear scale, volume 50%)
player.volume=0.5
```

This is equivalent to about:

```
# set volume offset (linear scale, volume -6dB)
player.volume=-6dB
```

Do not use levels more than 1 (linear scale) or positive decibel values, the output level may overflow with loud samples and cause distortions to audio.

Sound buffer length

The sound player mixes the output audio data in blocks (or buffer) of fixed size. The output is also double-buffered, the next block of data is calculated as the first is being written to the audio hardware. The buffering also means that an action (for example, the subject presses a button to play a sample) that is meant to take effect immediately will necessarily have a delay that roughly corresponds to two times the length of the buffer. The default buffer length is 4096 sample frames that will cause a delay of about 200ms (sample rate = 44.1kHz). It is possible to shorten the delay by shortening the buffer length. Shortening the buffer length reduces the delay but it also increases the risk of getting dropouts in sound if the buffer calculation didn't finish in time (other system activity and increased overhead caused by shorter blocks may cause problems, also audio files are loaded from disk on the fly).

The buffer length is set with the **buflen** parameter in the test config file. The length is in sample frames. The length of 44100 corresponds to one second with the sample rate of 44.1kHz. Example:

```
  # set mixing buffer length
  player.buflen=4096
```

Sets the buffer length to 4096 sample frames (roughly 93ms with sample rate 44.1kHz).

Note: If a command was scheduled beforehand to start at a certain point of time, no delay is observed (sub-millisecond accuracy).

Last modified: Sat Jul 18 18:27:46 EEST 1998

Utilities & Tools

Some tools and utilities for running the tests and for creating them:

- **RunTest -** run a listening test.
- **ResPrint -** convert and print results as a ASCII files
- **FontTester -** show Java fonts available in the system
- **UITester -** test and debug subject UI properties
- **PlayerTester -** test sound player and sound samples.
- **UIServerTest -** test remote UI server and client connections.
- **RemoteUIClient -** request a remote subject UI client from subject UI server and start it.
- **RemoteUIApplet -** an applet to request a remote subject UI client from subject UI server and start it.

· **Document index** ·

Last modified: Sat Jan 16 14:35:23 EET 1999

Running a test

NAME

gpRunTest - run a listening test.

SYNOPSIS

```
gpRunTest [-f test_config_file] [-session sessionID] -subject subjectID
  [-timeout timeout] [-wtimeout warning_timeout]
  [-sequenceType free|fixed] [-sequence sequence]
  [-sampleSwitching normal|parallel]
```

DESCRIPTION

A test is run using the graphical **gpRunTest** command. It is usually started without any arguments, but command line options can also be used. For a bit quicker start, you could give the test configuration file, session ID and subject ID on the command line.

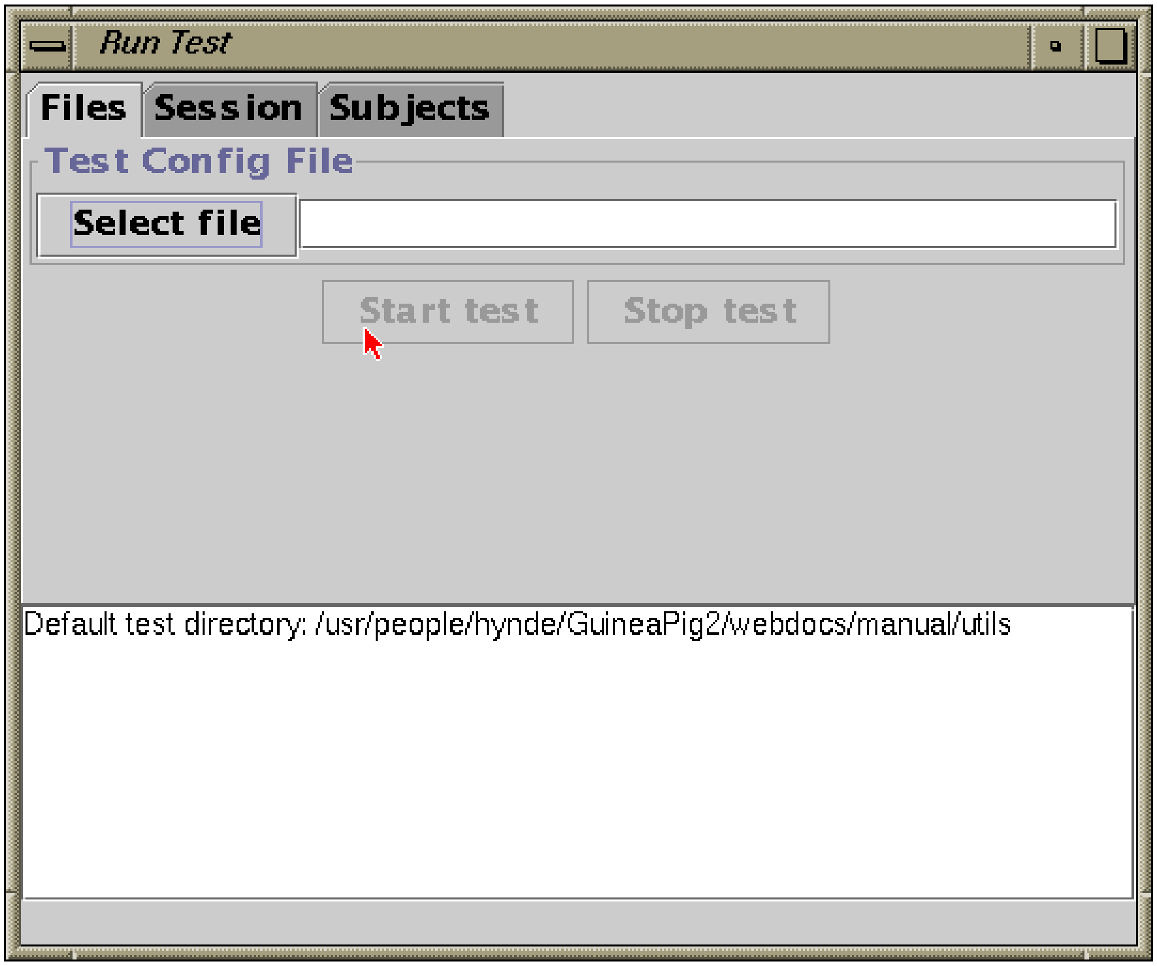
Usually a test goes like this:

1. Go to the test directory (not required but a good practice).
2. Start the **gpRunTest** tool to run the test. The GUI will appear. Set required information about session:
   1. **Select the test configuration file** either from the *Files* panel from the GUI or with the -f command line option.
   2. **Select session ID** for test session either in the *Session* panel or with the -session command line option.
   3. Optionally, **select playlist file** manually in the *Session* panel if the tool failed to find the correct playlist file automatically.
   4. **Add subjects** to test session in the **Subjects** panel. Subjects on local and remote terminals can be added. Local terminal subjects can also be added with the -subject command line option.
3. **Start the test** with the *Start test* button on **Files** panel.
4. Test items are played to the subject and answers from the subject are saved.
5. When all items have been played the results are written to a file. The file contains java objects containing copies of test items and corresponding answers from the subject. The results can be converted to a text file with result processing tools.
6. Terminate RunTest tool with the window-close button.

If an error occurs during the test, the system tries to save as much as it can (the items finished in the session).

RUNTEST GUI

The structure of the RunTest tool’s GUI panel looks like this:



```
Default test directory: /usr/people/hynde/GuineaPig2/webdocs/manual/utils
```

Figure 1: RunTest panel.

On the top there are **tabs** that lead to various settings panels. Below the tabs is the selected **panel**. Below it, there is a **message area** that displays RunTest messages. The last line is a **status line**.

FILES PANEL

The **Files** panel is used to select the *test configuration file*. Press **Select file** to pop-up a file selection dialog to select config. file:

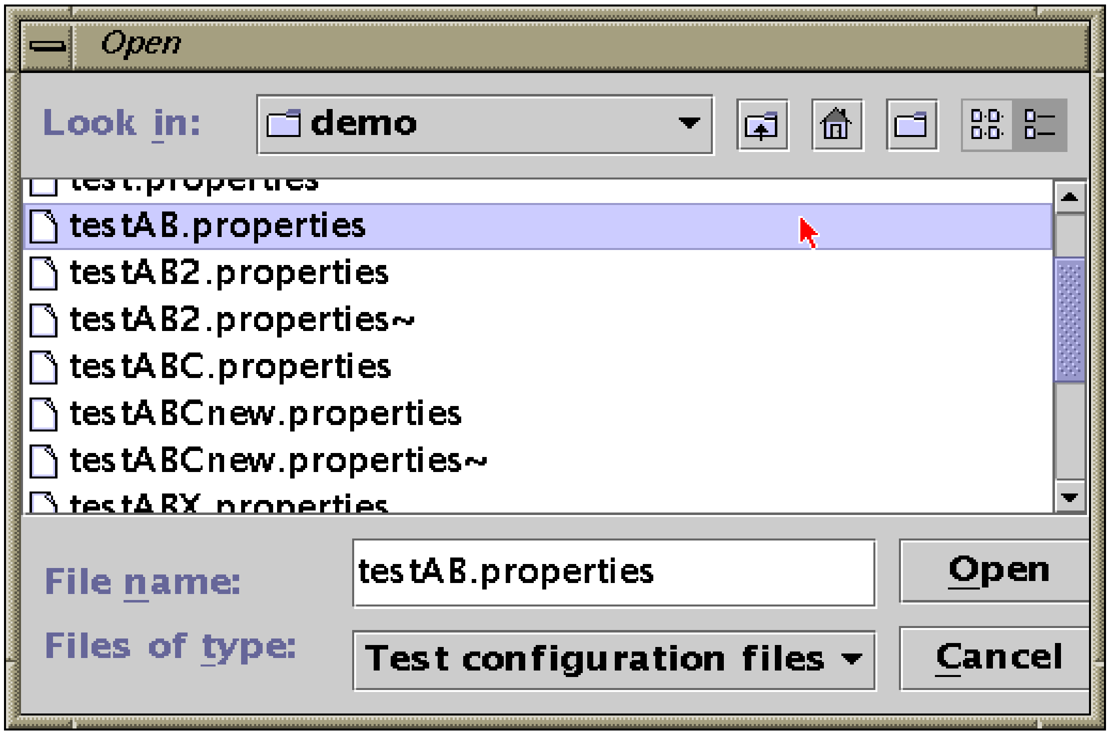


Figure 2: Test configuration file selection dialog.

The panel is also used to start the test when all settings have been done. Press **Start test** to run the test.

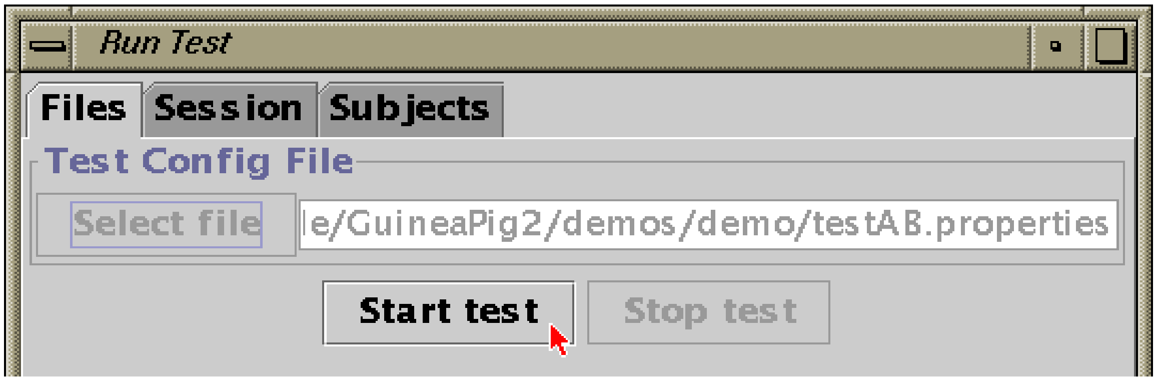


Figure 3: Starting test.

SESSION PANEL

The **Session** panel is used to set the session ID for the session. Also playlist file can be selected.

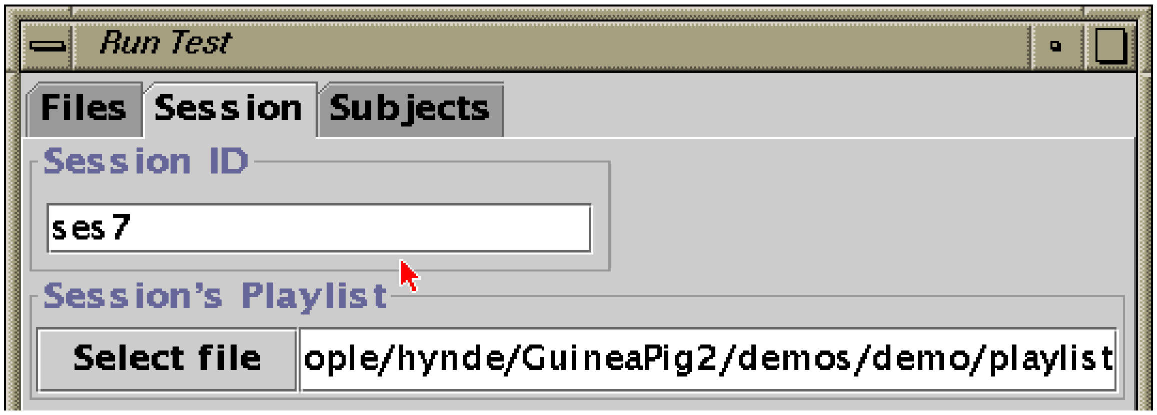
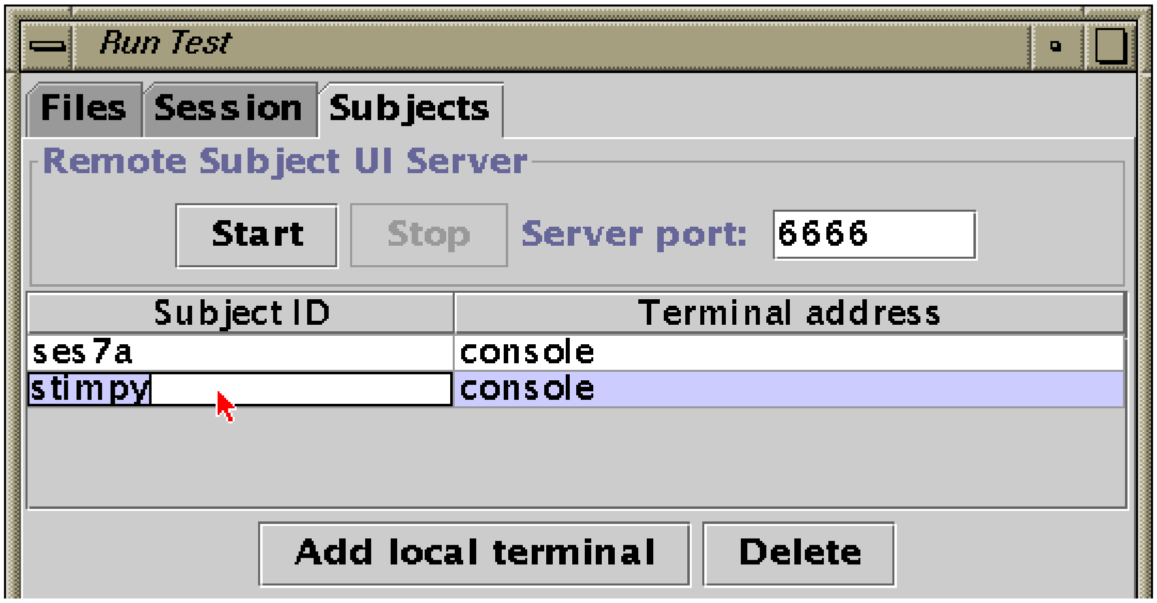


Figure 4: Session panel. Setting session ID and playlist.

If the **playlist** field is empty, a playlist is search for automatically when session ID is selected (remember to press return in session ID field). Pressing **Select file** displays a file selection dialog for playlists (similar to dialog in Fig. 2).

SUBJECTS PANEL

The **Subjects** panel is used to add local and remote subjects to the test. Subjects can also be deleted from the session.



Subject ID	Terminal address
ses7a	console
stimpy	console

Figure 5: Subjects panel. Adding and deleting local and remote terminal subjects.

In the lower part of the panel, local console subjects can be added with the **Add local terminal** button. A local or remote subject can be deleted from session with the **Delete** button.

For using remote terminals, the *remote subject UI server* must first be started with the **Start** button. The server waits for connections from remote terminals. When a connection is made, the server sends the remote terminal the subject UI panel and adds the remote subject to the subjects list. Use the **Stop** button to close the server. After closing the server, no more remote terminals cannot be added.

The subjects added to the current session are shown. In the *Subject ID* field the subject's ID is shown. The subject ID is editable by the tester. In the *Terminal address*, the subject's terminal location is shown. For local console subjects *console* is shown. For remote subjects the IP address and the hostname of the remote machine is shown.

COMMAND LINE OPTIONS

Here are the command line options that can be used. All command line options are optional, required values can be set using the GUI also.

-f *test_config_file*
: Set the test configuration file where to read test information. This should be the first option on the command line.

-session *sessionID*
: Set the session ID of this session. If no session ID is set, a new 'unique' ID will be generated. If session-specific playlist are used, the session ID must be set explicitly (the system searches session playlists based on session IDs). Session ID should be set before the subjects are added.

-subject *subjectID* (required)
: Add a subject to this session with ID '*subjectID*'. If subject ID is *NONAME*, a new subject ID is generated based on the session ID of this session. Session ID should be set before the subjects are added.

In addition there are some additional options that allow you to override some test parameters set in the configuration file. These options shouldn't be used directly, the parameters should be in the test parameter file. These option are mostly just for testing and debugging.

-timeout *timeout* (optional)
: Set the time (in seconds) the subject has to give the answers. If set to zero, time limit will not be enforced.
See also: Timeouts

-wtimeout *warning_timeout* (optional)
: Set the time in seconds before the timeout to warn that time is about to end. If set to zero, no warning is shown (goes directly to timeout when time ends).
See also: Timeouts

-sequenceType *free* | *fixed* (optional)
: Set sequence type of test. In *free* the subject can freely switch between samples, in *fixed* a fixed sequence of samples is played.
See also: Sequences

-sequence *sequence* (optional)
: Set sample sequence for test.
See also: Sequences

-sampleSwitching *normal* | *parallel* (optional)
: Set sample swithing type in free sequence tests. In *parallel* a switch from one sample to another is done with a cross-fade, in *normal* the first sample stops and the new sample starts from beginning.
See also: Switching

EXAMPLES

These examples show primarily using RunTest with command line options.

First the simplest and most commonly used ways to run the test. First:

```
cd TESTDIR
RunTest -f test.properties -subject arnold
```

and second:

```
cd TESTDIR
RunTest -f test.properties -subject NONAME
```

The *TESTDIR* is the pathname of the test directory. The file '*test.properties*' is the name of the file test configuration file. A new session ID for this session is generated because session ID hasn't been set explicitly. If the first example, a single subject is used and his subject ID is '*arnold*'. In the second example, setting the subject's ID to '*NONAME*' will generate a new subject ID for this subject based on the session ID.

In the next example the session ID of the session is set explicitly:

```
cd TESTDIR
RunTest -f test.properties -session ses02 -subject NONAME
```

This will set the session ID to '*ses02*' for this test. Also a new subject ID for the subject is generated based on the new session ID.

SEE ALSO

Tests, Timeouts, sequences, switching, **results processing**

NOTES

Last modified: Wed Jul 28 13:12:43 EEST 1999

Printing test results files

NAME

ResPrint - print test results.

SYNOPSIS

```
ResPrint [-f options_file] [-o output_file]
  [-fields list_of_fieldIDs] [-items list_of_itemIDs]
  [-sessions list_of_sessionIDs] [-subjects list_of_subjectIDs]
  file ...
```

DESCRIPTION

The **ResPrint** tool converts the serialized the result files created by the **gpRunTest** tool to easier human/computer readable text files.

ResPrint takes as arguments the file names of the serialized results files, reads them in and prints out the information in tabulated text format. When **ResPrint** starts, it first loads GP system's default result output options file. Then it looks for the '*results.properties*' output options file in the current directory and if found, loads it (see also section files).

Command line options can be used to select some of the test items to print. Also an options file can be given that can be used to configure the output.

-f *options_file* (optional)
: Set the test configuration file where to read result printing options. This should be the first option to the ResPrint program. Options included in this file override the corresponding options given in the default and/or the current directory's options file.

-fields *list_of_fieldIDs* (optional)
: Select which fields to output and the order of the fields. This option overrides the ResPrint's default list of fields or the result options file's list of fields.

-o *output_file* (optional)
: The file where to write the results. If omitted, results are printed on standard output.

file ... (required)
: Select which results files to process.

Some options can be used to filter the output of results. You can filter by *item ID, session ID* or *subject ID*:

-items *list_of_itemIDs* (optional)
: Only print results for specified items. The *list_of_itemIDs* is a comma-separated list of test item IDs to include in output. If this option is not used, results for all items will be included.

-sessions *list_of_sessionIDs* (optional)
: Only print results for specified sessions. The *list_of_sessionIDs* is a comma-separated list of test session IDs to include in output. If this option is not used, results for all sessions will be included.

-subjects *list_of_subjectIDs* (optional)
: Only print results for specified subjects. The *list_of_subjectIDs* is a comma-separated list of

subject IDs to include in output. If this option is not used, results for all subjects will be included.

If several filters are specified, the intersection of the item sets produced by the separate filters are included for output.

FILES

The **ResPrint** tool reads the **serialized results files** that are output as the result of running a test. They are by default written to files with the form of ’*results_SESSIONID.ser*’ where the *SESSIONID* is the session ID of the session.

Output files: The tool converts the serialized results files to TAB-delimeted text format and outputs it to stardard output. You can save the results to a file by redirecting the output of the tool to a file. In the output, results are output one line per test item with item’s parameters and answers as fields. Fields are separated by a ASCII TAB character. Comment lines may be output, they always begin with the hash (’#’) character.

Result printing options files: The default printing options are first automatically loaded from GP systems default results options file ’*/usr/GuineaPig/lib/results.parameters*’. Then, if a file named ’*results.properties*’ is found in the current directory, it is loaded. Options found in that file override the corresponding options in the default options. Then if an additional option file is provided with the -f option, it is loaded.

The format of the options file(s) is described here.

EXAMPLES

First go to the test directory where the result files are kept. Then run the converter program and give the file names of the session result files as agruments, for example:

```
ResPrint results_*
```

if your result file names begin with ’results_’. The converter then scans in the files given as arguments and prints out the results in a tabulated table. If you have the result printing options file ’*results.properties*’ in the same directory, it is loaded automatically.

To use an additional or alternate options file you want to use, give its name with the -f option, for example:

```
ResPrint -f results-alt.properties results_*
```

where the ’*results-alt.properties*’ is the name of the additional options file.

To print only the item ID, the parameters ’A’ and ’B’ and the answer for question ’q1’ (such as in example for A/B test in demos):

```
ResPrint -fields ITEMID,A,B,q1 results_*
```

To print only results given for items with IDs ’*item01*’, ’*item04*’ and ’*item22*’:

```
    ResPrint -items item01,item04,item22 results_*
```

To print the the same items above, but only those during sessions '*SES04*', '*SES05*' and '*SES06*':

```
    ResPrint -items item01,item04,item22 -sessions SES04,SES05,SES06 results_*
```

Note that you can put those filters in an additional options file and read them with the **-**f option.

SEE ALSO

Tests, **results processing**

NOTES

Last modified: Mon Jan 18 15:04:46 EET 1999

FontTester

The **FontTester** is a tool that shows which Java fonts are available and what they look like. It is useful for selecting which fonts to use in the subject UI blocks.

The font tester is started with the **gpFontTester** command. It will bring up the font tester window:

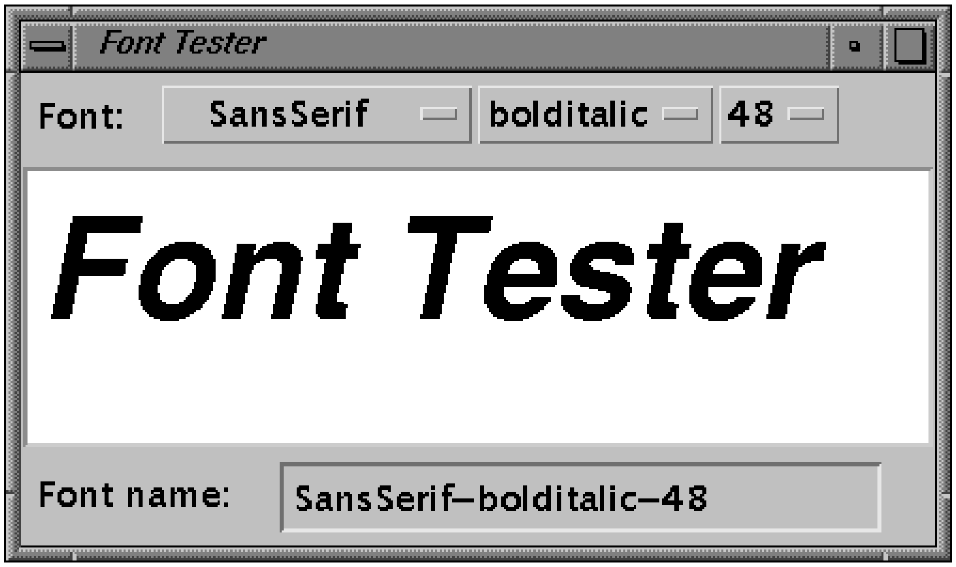


Fig. 1: The font tester window.

In the top are the choices of *font family*, *font style* and the *font size* available. Change font parameters with the pop-up menus.

In the middle there is a text field that contains example text using the selected font. You can edit the example text with mouse and keyboard. If the text field becomes too small to fit the text, resize the window.

In the bottom the Java font name of the selected font is showed. You can use the font name with Java apps. You can cut the font name and paste it elsewhere with the mouse.

To quit, close the window using the close-button in the top left.

Last modified: Wed May 6 16:56:37 EEST 1998

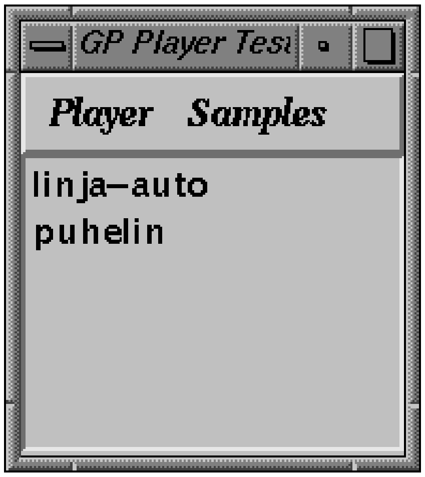
Sound Player Tester

The **PlayerTester** is a tool for testing the sound player and sound samples. You can select the player parameters (such as sample rate, channels, devices, etc.) and you can load samples and play them.

The player tester is started with the **gpPlayerTester** command with no arguments. For example:

```
gpPlayerTester
```

It will bring up the main window:

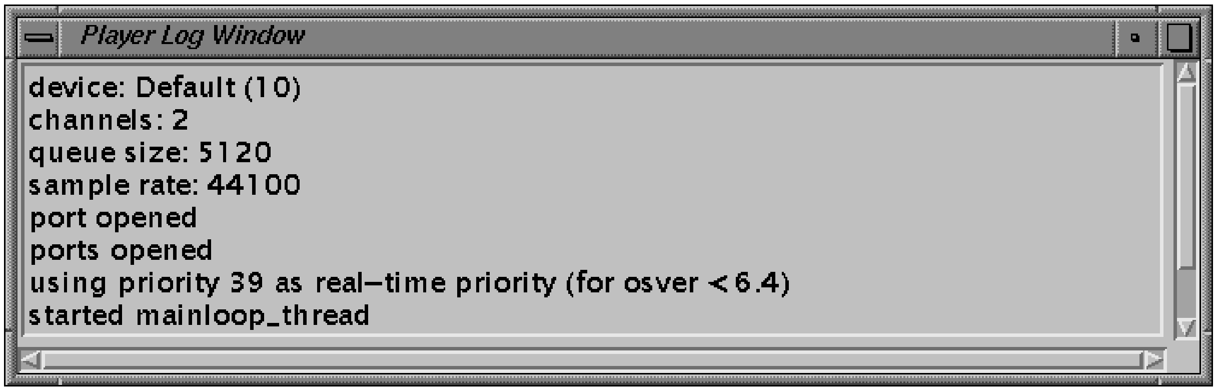


```
linja-auto
puhelin
```

Fig. 1: An example of a player tester window.

The window has two menus **Player** and **Samples**. They are used for player and sample operations. Below then there is a list that shows the samples that are currently loaded.

A **Player Log Window** will pop up also. It will display debug messages from the external player and the java module. You can close it with the close button in the top left. If you want to see it again, select show log window from the player menu.



```
device: Default (10)
channels: 2
queue size: 5120
sample rate: 44100
port opened
ports opened
using priority 39 as real-time priority (for osver < 6.4)
started mainloop_thread
```

Fig. 2: An example of a player tester log window.

Player menu

The player menu can be used to select the parameters of the player. The sample rate, channels, audio device and launch mode must be set before starting the player (launch), they cannot be changed after the player has been started.

Sample rate:
: Some preselected sample rates available are listed. Select desired sample rate from the list to use that rate. Default is 44.1kHz.

Channels:
: Select number of output channels, 1, 2, 4 or 8 are listed. The number of channels that is available depends on the audio device used. Default is 2.

Audio device:
: The audio device name to use. The default is ’default’ which uses the default audio device selected with apanel. Some devices may not be available on your system.

Master volume:
: Set player’s output level correction. Some preselected choices are listed. The level can be set any time. Default is 0dB.

Virtual players:
: Allocate new virtual player with specified number of channels within a real player. The player must be running already for this to work. Example: first start player with 8 output channels via ADAT Out interface. Then allocate four 2-channel virtual players. Now load several stereo samples and assign them to various virtual players (see later).

Launch mode:
: Select the mode that the soundplayer communicates with the external player program. The default is ’socket’.

Launch:
: Start the player. The player parameters must be before launch to take effect.

Connect:
: Connect to a player that is already running and waiting for a connection. You usually should use *launch* above.

Stop:
: Stop and quit player player.

Quit:
: Stop and quit player program and exit PlayerTester.

Show Log Window:
: Show log window if you have closed it.

Samples menu

The samples menu is used to load samples and combine several samples. The menu only works after when the player has been started.

Load Soundsample:
: Load a sound sample file. It will pop up a file selection dialog to get the file name. It will then load the sample, add the name of the sample to the sample list and starts a sample window for the sample.

Load serialized sample:
: The same as above but reads the properties from a serialized file and loads the sample.

Load property sample:

The same as above but reads the properties from a text properties file and loads the sample.

Make parallel:

Create a new parallel sample. First select the samples you want to combine from the sample list and then select ’make parallel’. A new sample window will come up.

Sample window

When sample is loaded, a new window for the sample is created:

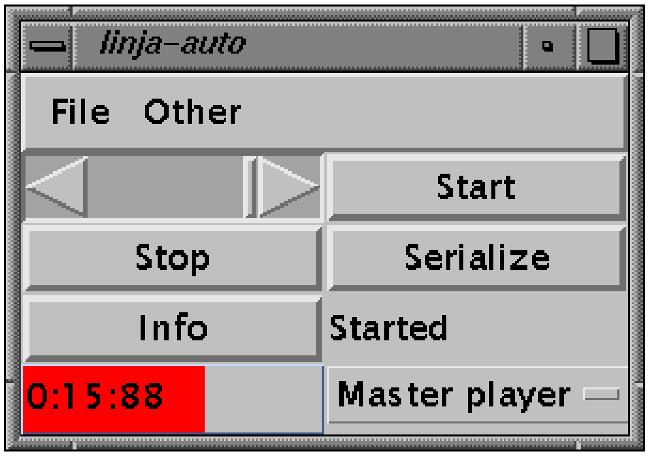


Fig. 3: An example of a player tester sample window.

The volume of the sample can be changed with the scrollbar on the top left. The *’start’* and *’stop’* button can be used to start/stop the sample (surprise). On the bottom left is a sample meter that shows the current position in the sample. You can jump to any position on the sample by clicking at a position or dragging and releasing an indicator to desired position. On bottom right is shown which virtual player is used. The default is ’master player’ which is the real player. If virtual players have been allocated with the player-menu, you can assign the sample to a virtual player. To unload the sample, close the window with the close-button on the top left.

Parallel sample

A parallel sample is a sample that is a combination of several samples. At any time only one of the samples is playing and the others are silent. When an other sample in a parallel sample is selected, the switch is done using a cross-fade. The window for parallel sample is almost the same as in regular sample window. Only the virtual player selection is replaced a selection of samples that consist this sample. Selecting one of the samples switches to the other sample using a cross-fade. Here is a figure of the parallel window:

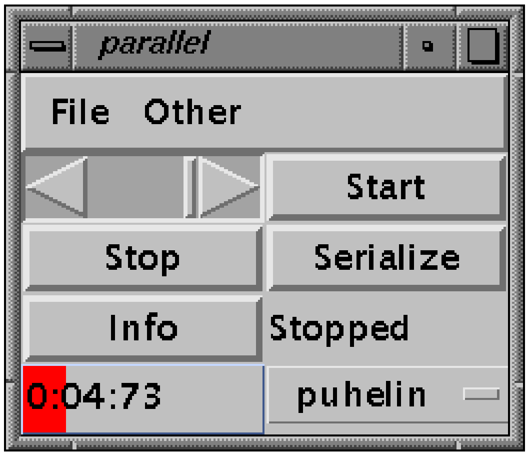


Fig. 4: An example of a player tester parallel sample window.

Last modified: Mon May 25 16:33:44 EEST 1998

Subject UI Tester

The **UITester** is a tool that shows what the subject UI window looks like before using it in a test. It is useful for experimenting with the ui parameters and adding ui components and testing them. Use it with the **FontTester** utility for selecting the fonts.

The font tester is started with the **gpUITester** command with the file name of the UI properies as an argument. For example:

```
gpUITester uiABC.properties
```

It will bring up the subject UI window (in this case an A/B/C test UI window:

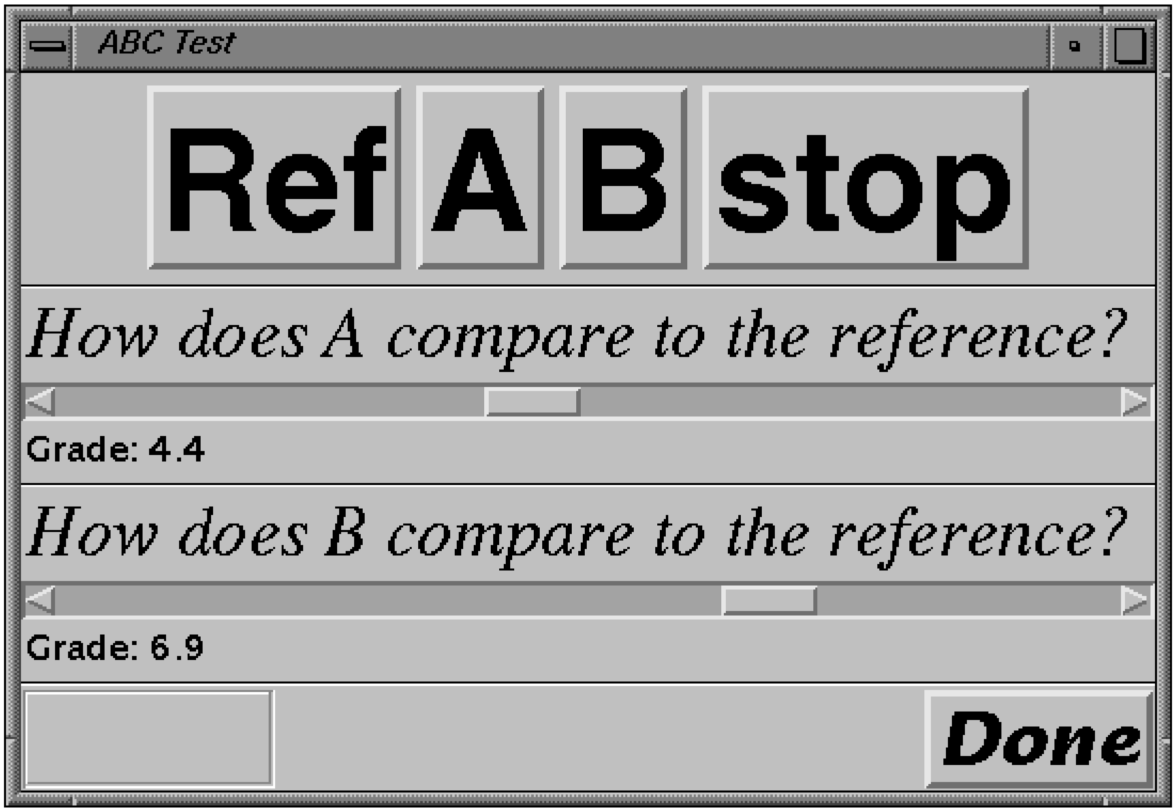

ABC Test
Ref
A
B
stop
How does A compare to the reference?
Grade: 4.4
How does B compare to the reference?
Grade: 6.9
Done

Fig. 1: An example of an UI tester window.

The window will show the control and answering panels of the subject UI. All the features of subject UIs cannot yet be tester with the UITester tool (more features will be added later).

To quit, close the window using the close-button in the top left.

Last modified: Mon May 25 14:41:47 EEST 1998

Subject UI Server Tester

NAME

gpUIServerTest - test remote UI server and client connections.

SYNOPSIS

```
gpUIServerTest [-p port] uifile
```

DESCRIPTION

The **UIServerTest** tool can be used to test that remote subject UI modules and connections work. The tester will start a remote subject UI server that will wait for connections to the server. When a connection is made, the server will send a copy of the specified subject UI module to the remote client. Also a local server version of the subject UI is made and it is passed for a UI tester program for testing the remote UI (using the gpUITester tool).

The **gpUIServerTest** has the following parameters and options:

uifile
: The file name of the subject UI module that is sent to the remote client.

-p *port*
: Specify the server's port number. If not set, default will be used.

To quit the server, press ctrl-C.

EXAMPLES

To start the server tester with the file '`uiABC.properties`' as the UI specification file:

```
gpUIServerTest uiABC.properties
```

To start the same with a specific server port number 5000:

```
gpUIServerTest -p 5000 uiABC.properties
```

SEE ALSO

gpUITester, gpRemoteUIClient, Remote UI Applet

Last modified: Thu Oct 8 15:30:38 EEST 1998

Remote Subject UI Client

NAME

gpRemoteUIClient - request a remote subject UI client from subject UI server and start it.

SYNOPSIS

`gpUIServerTest` `[-p` *`port`*`] [server]`

`java guinea.tools.remoteui.RemoteUIClient` `[-p` *`port`*`] [server]`

DESCRIPTION

The **gpRemoteUIClient** connects to a remote subject UI server and requests a subject UI module that will be shown to the test subject. When the connection has been completed, the client subject UI will start and the window will come up.

The **gpRemoteUIClient** has the following parameters:

server
: Specify the subject UI server's hostname. If no hostname is given, 'localhost' is used.

-p *port*
: Specify the server's port number. If not set, default will be used.

EXAMPLES

To start the UI client and request the subject UI from default server (localhost) and default port number:

```
gpRemoteUIClient
```

To use host 'gpserver.foo.com' as the subject UI server:

```
gpUIServerTest gpserver.foo.com
```

To use host 'gpserver.foo.com' as the subject UI server and port number 5000:

```
gpUIServerTest -p 5000 gpserver.foo.com
```

To start the client from remote terminal that doesn't have GP system installed (either SGI or non-SGI), you'll have to start the client with command:

```
java guinea.tools.remoteui.RemoteUIClient -p 5000 gpserver.foo.com
```

It will do the same as in previous example. You'll also need to have the GP java class files and their location added to the Java class path. You can copy the java class files from GP home directory, copy the file named `guinea.jar`.

SEE ALSO

gpUIServerTest, Remote UI Applet

· Utils Index · Document index ·

Last modified: Wed Nov 4 16:13:35 EET 1998

Remote Subject UI Applet

NAME

guinea.tools.remoteui.RemoteUIApplet.class - an applet to request a remote subject UI client from subject UI server and start it.

SYNOPSIS

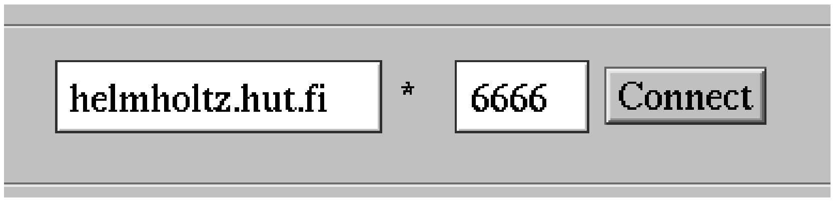
`appletviewer` *`URL`*
`netscape` *`URL`*

DESCRIPTION

The **RemoteUIApplet** is an applet that can be used to connect to a remote subject UI server the same as with the **gpRemoteUIClient** tool. Since the client is an applet, you can theoretically use any networked java-capable device to display the test user interface to the subject. At least Sun's JDK *appletviewer* and *netscape 4.06* (both on SGI) seem to work. Also a quick test with Windoze version of *netscape 4.06* seemed to work. Netscape's versions prior to 4.06 generally didn't seem to work.

The applet will show as a small panel on the web browser. The applet allows the subject to set the remote UI server's hostname and port number. A popup menu can also be used to select the UI server's hostname and port. Finally the subject presses the '*Connect*' button to initiate the connection to the remote server. From there on things go the same as with the **gpRemoteUIClient** tool.

Here is an image of the applet:



An example of RemoteUIApplet: on left there is a textfield to select the server hostname, next is a button to pop up a selection of available UI servers, next is a textfield to set the port number and finally a 'Connect' button to contact the remote UI server.

APPLET CONFIGURATION

Applet's parameters and options are set by the tester by giving parameters to the applet using the applet-tag's param-tags on the HTML-page that invokes the applet. The default server host and the list of selectable servers from the popup menu can be configured. Also the fonts used by the applet can be set.

Font parameters

Font parameters allow setting different fonts for different objects in the applet panel. Available parameters are:

menufont *fontspec*
: Set the font for menu items in the host popup menu.

textfont *fontspec*
: Set the font for textfields (hostname and port).

buttonfont *fontspec*
: Set the font for buttons (connect-button).

font *fontspec*
: Set the default font. It will be used if other specific fonts have been set.

Server host parameters

Server host parameters allow setting the default server hostname and portnumber as well as list of hosts available from the host menu. Available parameters are:

defaultHost *hostinfo*
: Set the hostname and port of the subject UI server shown initially.

defaultPort *portnumber*
: Set the default port number to use if port number has not been explicitly set (usually with menu configuration and *hostinfo*s).

hostN *hostinfo*
: Add a server into the popup menu. The **N** is an integer starting from 1 (one). The applet will scan applet parameters starting from *host1* and goes on to *host2*, *host3* and so on. It will stop when it finds no *host(n+1)* after *host(n)*. Parameter names *host3*, *host03* and *host003* are all equivalent.

The *hostinfo* parameter tells the address of the server and which port to use. The simplest form is the host name alone, for example:

```
foo.bar.com
```

It specifies the host ’*foo.bar.com*’ and uses the default port. A non-default port number can be specified:

```
foo.bar.com:9000
```

Port 9000 of host ’*foo.bar.com*’ is specified. For the menu, also a label can be set:

```
foo.bar.com:9000;GuineaPig UI server
```

This will define the same address as previous but on the menu, ’*GuineaPig UI server*’ will be shown as the label. If no label is set, the hostname will be used as the label. If default port is used, the port number can be left out:

```
foo.bar.com;GuineaPig UI server
```

In the *hostinfo* sepcifications, the hostname can be set to ’`DOCUMENTHOST`’ (all capitals). This will be automatically replaced with hostname of the host the applet was loaded from. For example:

```
DOCUMENTHOST;GuineaPig UI server
```

If the applet was loaded from host *foo.bar.com*, this hostinfo will be automatically changed to

```
foo.bar.com;GuineaPig UI server
```

The port number can be added the same way as with other examples. If the applet was loaded from local disk (using a `file:` URL), '*localhost*' will be used as the hostname.

Other options

It is possible to disable the host menu and the textfields for entering host name and port number. Also the applet can be set to automatically connect to the UI server when applet is started. Options are:

showHostMenu *true* or *false*
: Whether to show the host menu to the subject for selecting the server host. By default this option on (value is *true*) and the host menu is shown. Set to *false* to disable the menu (the menu will not be shown).

allowSetHost *true* or *false*
: Whether to allow the subject to set the hostname and port by using the textfields. By default this option is on (value is *true*). Set to *false* to not allow the subject to set the host name and port number, the textfields are used only to show the name of the selected host. The host menu can be used to select the server host (unless the host menu also has been disabled).

autoConnect *true* or *false*
: Whether to automatically connect to the server when the applet is started. By default this option is off (value is *false*). When using the autoconnect, the server host should have been set with the defaultHost parameter. Also the allowSetHost and showHostMenu should generally be set to *false*.

APPLET TAG

The HTML's <APPLET> tag is used to embed the applet onto the page and to pass parameters to the applet. Here is an example of what the applet tag would look like:

```
<applet code="guinea.tools.remoteui.RemoteUIApplet.class" archive="guinea.jar"
        width=400 height=50
        alt="Your browser understands the APPLET tag but isn't running the applet, for some reason.">

<param name="font" value="Serif-14">
<param name="menufont" value="Serif-italic-14">

<param name="defaultHost" value="localhost;Localhost">
<param name="defaultPort" value="6000">

<param name="host1" value="bird.hut.fi;Hynden kone">
<param name="host2" value="helmholtz.hut.fi">
<param name="host3" value="helmholtz.hut.fi:6001;Helmholtz, port 6001">

Your browser is completely ignoring the APPLET tag!
</applet>
```

More detailed description of some tags:

code
: The java class name of the applet. It must be as shown or the applet will not work.

archive

The name of the Java ARchive (jar) that contains GuineaPig java classes. In this example the `guinea.jar` file must be found from the same directory where this html-file is in.

param

Parameters are passed to the applet with **param** tags. They each contain a parameter name, value pair. See above for available parameters names and values.

NOTES

Usually web browsers only allow network connections to the same host where the applet was loaded from. Some browsers can be set to allow connections to other hosts also. For example, JDK's *appletviewer* allows applet to make connections to other hosts if network access properties are changed to allow unrestricted access. Running applets from local disk (using `file:` URLs) usually allow unrestricted network access.

The Java tutorial gives more detailed information about what applets can and can't do.

SEE ALSO

gpRemoteUIClient, gpUIServerTest
Java tutorial: what applets can and can't do

· **Utils Index** · **Document index** ·

Last modified: Fri Oct 30 12:30:08 EET 1998

API Documentation

API documents are generated from the documentation comments in the Java source code using the Java’s *javadoc* utility and some postprocessing. The documents provide information on all java classes used in the system.

Guinea Pig Java class API documents

- **Packages Index**
- **Class Hierarchy**
- **Index of All Fields and Methods**

· Document index ·

Last modified: Sun Apr 19 20:51:42 EET DST 1998