

GNU Solfege 3.9.4 User Manual

Tom Cato Amundsen

[<tca@gnu.org>](mailto:tca@gnu.org)

Copyright © 2005 Tom Eykens

Copyright © 2000, 2001, 2002, 2003, 2004, 2005, 2006, 2007, 2008 Tom Cato Amundsen

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version. The full text of the License is available in [Appendix A, GNU General Public License](#).

Table of Contents

[1. Introduction](#)

[About the manual](#)

[Bugs](#)

[Online resources](#)

[Download Solfege](#)

[Mailing lists](#)

[Preferences window](#)

[Midi stuff](#)

[User](#)

[External programs](#)

[Gui](#)

[Practise](#)

[Sound setup](#)

[Training set editor](#)

[Ear training test print-out editor](#)

[2. Help sections for the exercises](#)

[Harmonic interval](#)

[Configuration](#)

[Key bindings](#)

[Melodic interval](#)

[Configuration](#)

[Key bindings](#)

[Sing interval](#)

[Config](#)

[Key bindings](#)

[Identify the chord](#)

[Key bindings](#)

[Identify the chord](#)

[Key bindings](#)

[Multiple choice answers to music](#)

[Key bindings](#)

[Sing chord](#)

[Key bindings](#)

[Rhythm](#)

[Key bindings](#)

[Tap the rhythm](#)

[Dictation](#)

[Key bindings](#)

[Scales](#)

[Key bindings](#)

[Intonation](#)

[Key bindings](#)

[Identify tone](#)

[Manual configuration](#)

[Beats per minute](#)

[Key bindings](#)

[Sing 12 random notes](#)

[Key bindings](#)

[Name intervals](#)

[Harmonic progression dictation](#)

[3. Music theory](#)

[Scales](#)

[Intervals](#)

[Seconds](#)

[Thirds](#)

[Fourth](#)

[Fifth](#)

[Sixths](#)

[Sevenths](#)

[Inverting intervals](#)

[4. Extending GNU Solfege](#)

[Introduction](#)

[Lesson files](#)

[File encoding](#)

[Comments](#)

[Types](#)

[Global variables](#)

[Lesson file contents](#)

[Header block](#)

[Question block](#)

[music objects](#)

[Functions](#)

[Operators](#)

[The chord module](#)

[The compareintervals module](#)

[The dictation module](#)

[The elembuilder module](#)

[The element block](#)

[The header block](#)

[The question block](#)

[The harmonicinterval module](#)

[The idbyname module](#)

[The idproperty module](#)

[The idtone module](#)

[The melodicinterval module](#)

[The nameinterval module](#)

[The rhythm module](#)

[The rhythmtapping module](#)
[The rhythmtapping2 module](#)
[The singanswer module](#)
[The singchord module](#)
[The singinterval module](#)
[Midi instrument names](#)

[Percussion instrument names](#)

[A. GNU General Public License](#)

[Preamble](#)

[TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION](#)

[Section 0](#)

[Section 1](#)

[Section 2](#)

[Section 3](#)

[Section 4](#)

[Section 5](#)

[Section 6](#)

[Section 7](#)

[Section 8](#)

[Section 9](#)

[Section 10](#)

[NO WARRANTY Section 11](#)

[Section 12](#)

[How to Apply These Terms to Your New Programs](#)

[B. Not really documentation...](#)

[Not really documentation...](#)

[Welcome to GNU Solfege](#)

List of Figures

- 3.1.
- 3.2.
- 3.3.
- 3.4.
- 3.5.
- 3.6.
- 3.7.
- 3.8.
- 3.9.
- 3.10.
- 3.11.
- 3.12.

Chapter 1. Introduction

Table of Contents

[About the manual](#)

[Bugs](#)

[Online resources](#)

[Download Solfege](#)

[Mailing lists](#)

[Preferences window](#)

[Midi stuff](#)

[User](#)
[External programs](#)
[Gui](#)
[Practise](#)
[Sound setup](#)
[Training set editor](#)
[Ear training test print-out editor](#)

About the manual

This user guide is available from inside GNU Solfege on the **Help** menu. The html viewer included in the program is limited in several ways, for example when it comes to CSS style sheets, so you might want to view it in a real web browser. You can find the install directory of the user manual from **File locations** on the **Help** menu.

Bugs

There are two ways to report bugs or request an enhancement regarding the GNU Solfege application or the user manual: send an email to <bug-solfege@gnu.org> or open a new bug on [SITS \(Solfege Issue Tracking System\)](#). General questions and patches should be sent to <solfege-devel@lists.sourceforge.net>.

Please make your bug reports detailed. "I get an error message in a window when I try to start the program." is not usable to me. When reporting bugs:

- Tell me what version of Solfege you run. Please check if a newer release is available. If you only want to run stable releases, then you don't have to test newer development releases.
- What operating system are you running? Version?
- Describe exactly what you are doing when the error happens.
- Send an exact copy of the error messages. They make sense to the Solfege author even if you think they look cryptic to you.

Online resources

The homepage for Solfege is <http://www.solfege.org>. There is also a smaller page with more static info at <http://www.gnu.org/software/solfege/>.

Download Solfege

The source code is available from <http://ftp.gnu.org/gnu/solfege>. If you are adventurous, you can try the unstable (buggy, but might contain new stuff) releases from <http://alpha.gnu.org/gnu/solfege>. These releases might have more bugs, but then you get the chance to try new stuff and find and report bugs.

Source code and some precompiled binaries are available from http://sourceforge.net/project/showfiles.php?group_id=1465.

If you run Debian you can **apt-get install solfege** to download and install the program.

Mailing lists

<solfege-announce@lists.sourceforge.net>

Very low traffic, moderated and will be used to announce stable releases of Solfege. ([Subscription](#) | [Archive](#))

<solfege-devel@lists.sourceforge.net>

If you want to report problems installing or running Solfege, or have questions, comments or ideas on how to improve Solfege, please post to this list instead of using the message forum at Sourceforge or the author directly. You can post to solfege-devel without subscribing. ([Subscription](#) | [Archive](#))

<bug-solfege@gnu.org>

The standard GNU address to send bug reports. This list is at the moment forwarded to <solfege-devel@lists.sourceforge.net>

Preferences window

Midi stuff

The screenshot shows the MIDI preferences window with the following settings:

- Tempo:** Default bpm: 60, Arpeggio bpm: 180
- Preferred instrument:** Instrument: acoustic grand, Velocity: 127
- Chord instruments:** Use different instruments for chords and harmonic intervals. Highest instrument: cello (Velocity: 64), Middle instrument: pizzicato strings (Velocity: 64), Lowest instrument: electric bass (finger) (Velocity: 64)
- Preferred percussion instruments:** Count in: Mute Triangle, Rhythm: Side Stick

A Close button is located at the bottom right of the window.

Tempo

Set the tempo (beats per minute) for music and arpeggios.

Preferred instrument

Set the midi instrument and velocity (volume of each note).

Chord instruments

Solfege can use three different instruments when playing chords. One for the highest tone, one for the tones in the middle and one for the bass tone. This can be helpful if you find it difficult to hear individual tones in chords.

Preferred percussion instruments

Set the percussion instrument used for count-in before rhythm questions, and the instrument used to play the question.

User

Midi **User** External programs Gui Practise Sound setup

User's voice

Highest note user can sing: d#''

Lowest note user can sing: g#,,

Sex

Male

Female or child

X Close

Solfege uses this info in some exercises where the user is supposed to sing.

Lowest/highest tone the user can sing

These spin buttons tell Solfege the highest and lowest tone the user can sing. These values are only considered advisory by the program. If for example the values are set to c to c' and you have configured the program to ask you to sing small and large decims, you will have to sing tones outside this range.

Sex

Solfege need to know if the user is male or female when creating some of the questions where the user will sing the answer. This because the male voice sound one octave lower than the female voice.

External programs

Midi User External programs Gui Practise Sound setup

Convertors

MIDI to WAV convertor: /usr/bin/timidity -Ow %(in)s

WAV to MP3 convertor: /usr/bin/lame %(in)s %(out)s

WAV to OGG convertor: /usr/bin/oggenc %(in)s

Audio players

WAV file player /usr/bin/aplay %s Test

MIDI file player /usr/bin/timidity -idqq %s Test

MP3 file player /usr/bin/mpg123 Test

OGG file player /usr/bin/ogg123 Test

Misc

Mail program: xterm -e mutt

X Close

Convertors

Give command lines that can convert between different audio formats. `%(in)s` will be replaced with the name of the file we convert from, and `%(out)s` with the name we convert to. It is not necessary to enter `%(out)s` if the program automatically saved to a new file with the correct file extension.

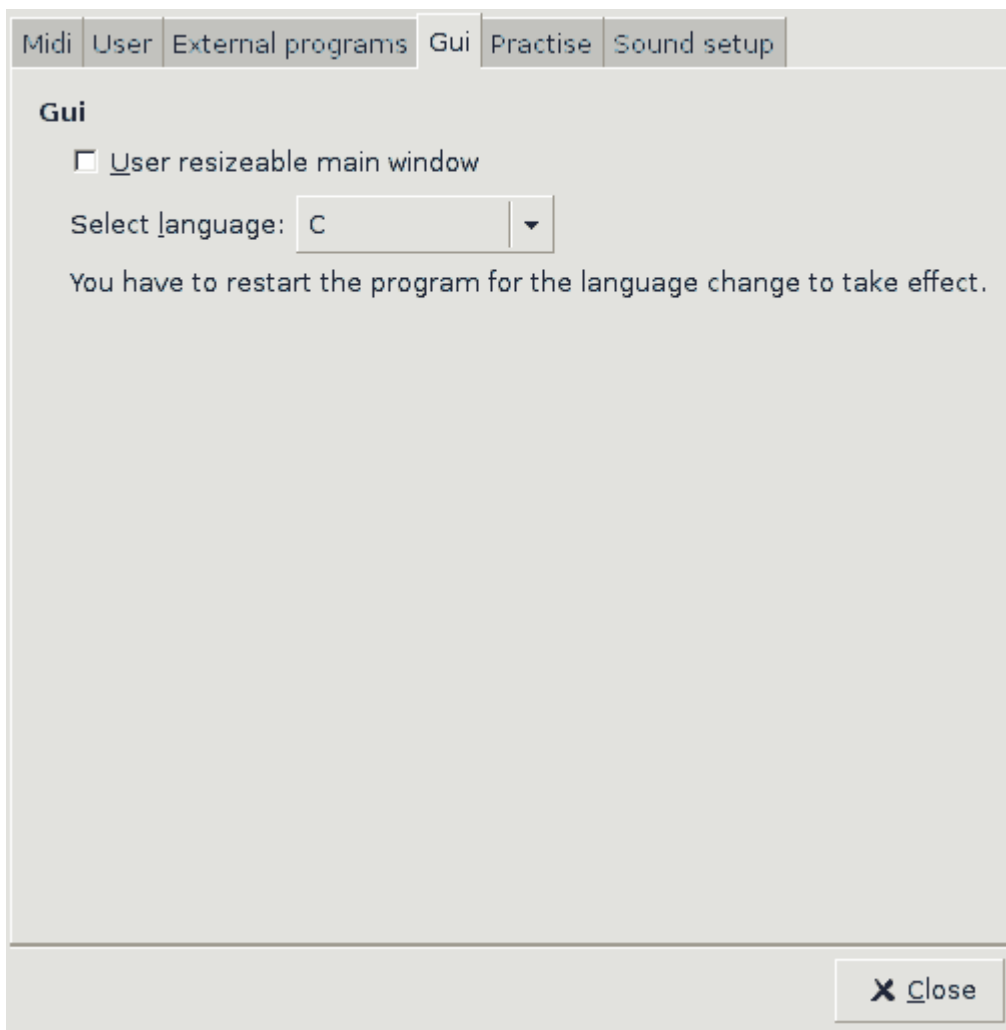
Audio players

Command lines that can play different audio formats. `%s` will be replaced with the name of the file to be played. The file name will be appended to the end of the string if you do not include a `%s`.

Misc

"Mail program" defines the command that starts the mail program when you click on email addresses in the user manual.

Gui

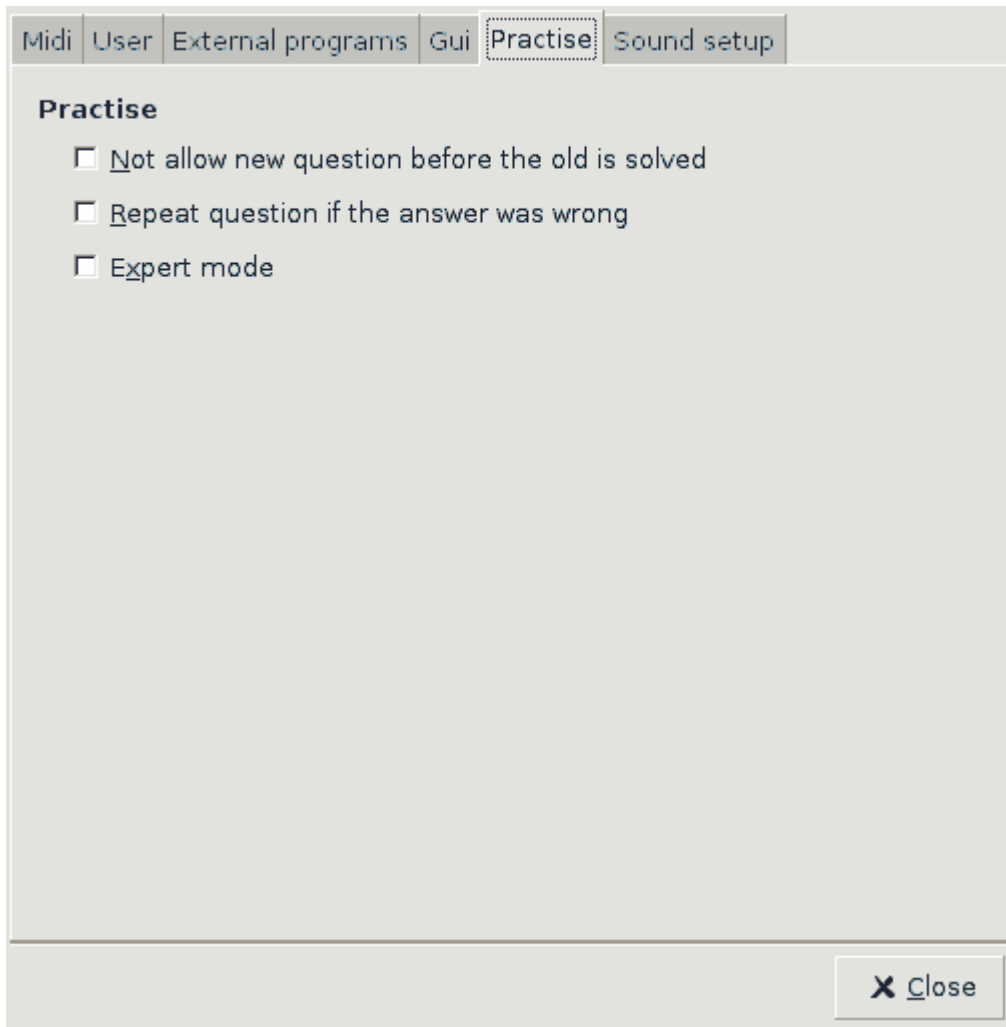


User resizable main window: Allows the user to resize the main solfege window.

Expert mode: Let the user select what questions from the lesson files to practise. No statistics is stored in "expert mode".

Select language: You can manually select the language you want if Solfege does not detect this correctly, or if you want to run Solfege with a different language than your operating system.

Practise

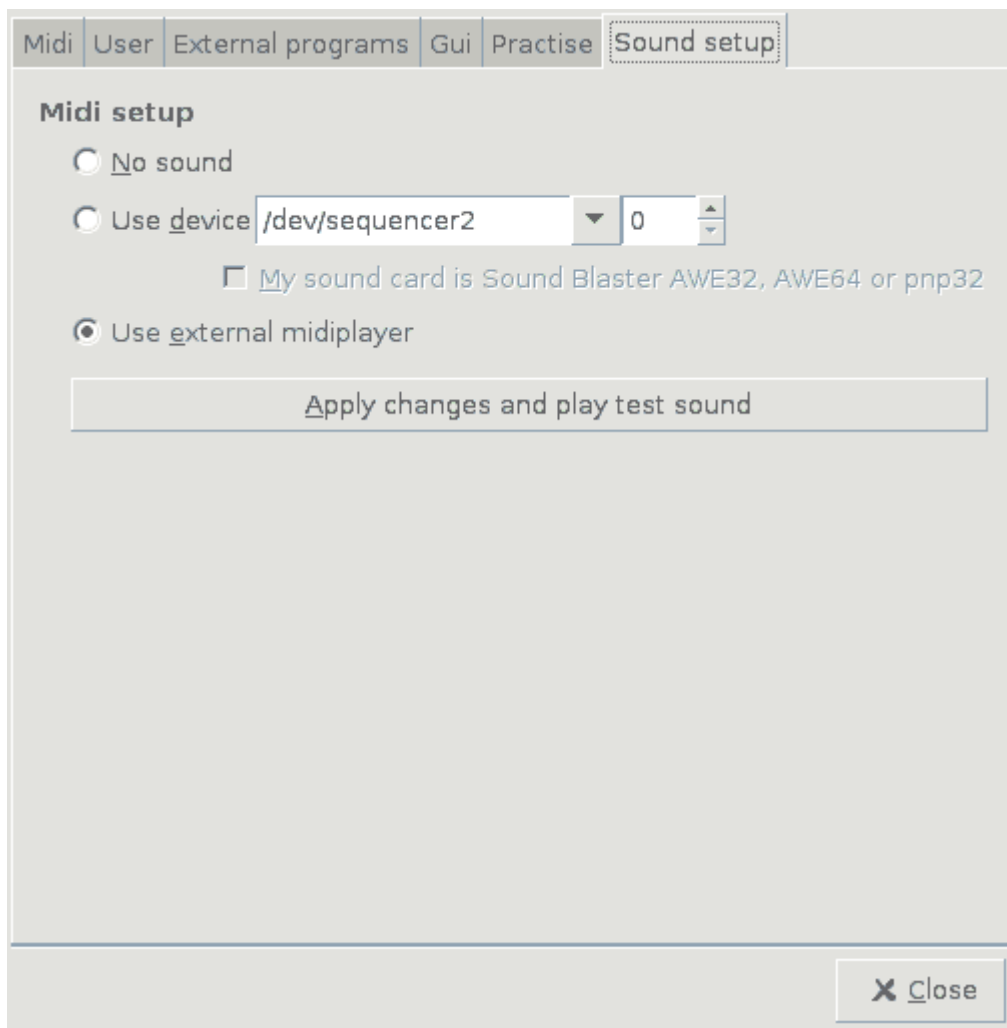


Not allow new question before the old is solved: Disable the 'new' button until the question is answered correctly or the user clicks "give up".

Repeat question if the answer was wrong: Play the sound again when the user gives an incorrect answer.

Expert mode: On many exercises, you will be able to select to practise only a subset of the questions in the lesson file.

Sound setup



Midi setup

There are three ways to play sound:

No sound:

Use this for debugging or when you are porting Solfege. No sounds are played, the midi events are printed to stdout.

Use device:

The best choice here is usually `/dev/music` because it has the best support for percussion instruments. `/dev/sequencer2` is usually a symbolic link to `/dev/music`. If your system don't have `/dev/music`, you can create it with this command as root (if you run the linux kernel version 2.2 or later):

```
cd /dev mknod music u 14 8
```

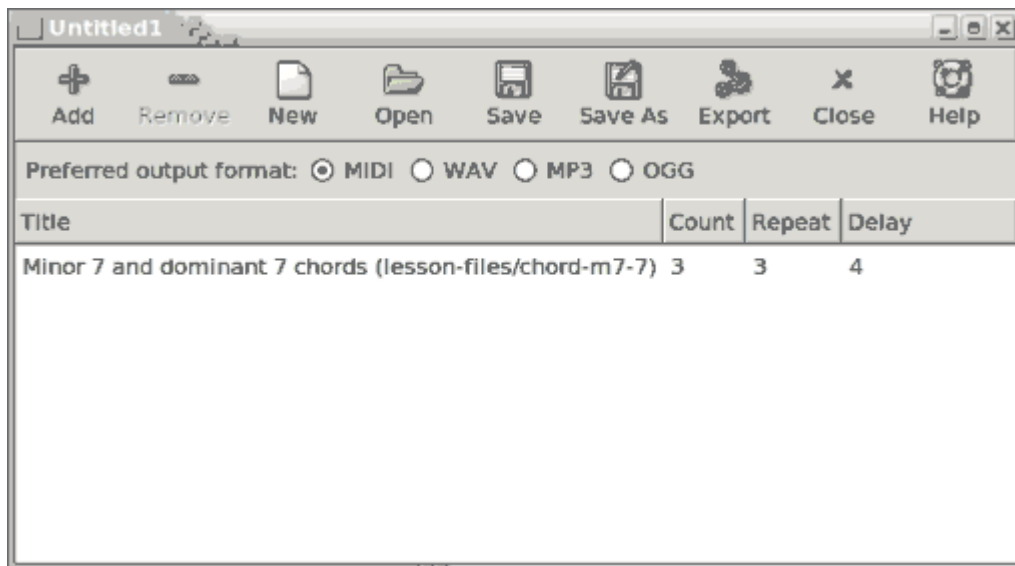
Use external midiplayer:

This can be useful when porting to systems that don't use OSS, or if you have a bad midi synth on your soundcard and want to use timidity.

Extra options

Check the "My sound card is Sound Blaster AWE32, AWE64 or pnp32" check button if you have this kind of sound card. This will give you real percussion in the rhythm exercise. Code still has to be added for other sound cards. This option is only necessary if you use /dev/sequencer to play midi sounds.

Training set editor



The training set editor let you create MIDI/WAV/MP3/OGG files of questions so that you can upload them to your pda, cell phone or MP3 player. A solution sheet will be generated for you to print out. Then you can let the MP3 player play the tracks by random order, and you can use the solution sheet to check if you recognised the music correctly.

You use the training set editor to define which exercises to generate. You can save your definition in a file for later use. Each time you click **Export** a new set of files are generated in a directory of your choice. You have to manually upload the generated sound files to you mobile device.

The program let you generate questions from as many lesson files as you like, but the most typical usage would be to generate lots of questions from just a single, or just a few files.

The programs used to convert between the different file formats are defined in Gui page of the [preferences window](#). Please check the definitions there if you have problems converting the MIDI files to WAV, MP3 or OGG format.

Table headings explained

Count

The number of questions to generate from the lesson file.

Repeat

The number of times to repeat each question.

Delay

How long delay it will be between the questions. Measured in the length of quarter-notes.

Ear training test print-out editor

This tool is available on the **File** menu. Use it to create ear training tests to print out on paper. Solfege will generate two versions of the document: one for the students to complete, and one with the correct answer already written.

The **Add** button will pop up a menu with all exercises from the active learning tree that this tool can create exercises from. When writing this, the exercise modules [idbyname](#), [melodicinterval](#) and [harmonicinterval](#). From lesson files written for the [idbyname](#) module, only [chord](#), [rvoice](#) and [voice](#) music objects are supported.

Chapter 2. Help sections for the exercises

Table of Contents

[Harmonic interval](#)

[Configuration](#)

[Key bindings](#)

[Melodic interval](#)

[Configuration](#)

[Key bindings](#)

[Sing interval](#)

[Config](#)

[Key bindings](#)

[Identify the chord](#)

[Key bindings](#)

[Identify the chord](#)

[Key bindings](#)

[Multiple choice answers to music](#)

[Key bindings](#)

[Sing chord](#)

[Key bindings](#)

[Rhythm](#)

[Key bindings](#)

[Tap the rhythm](#)

[Dictation](#)

[Key bindings](#)

[Scales](#)

[Key bindings](#)

[Intonation](#)

[Key bindings](#)

[Identify tone](#)

[Manual configuration](#)

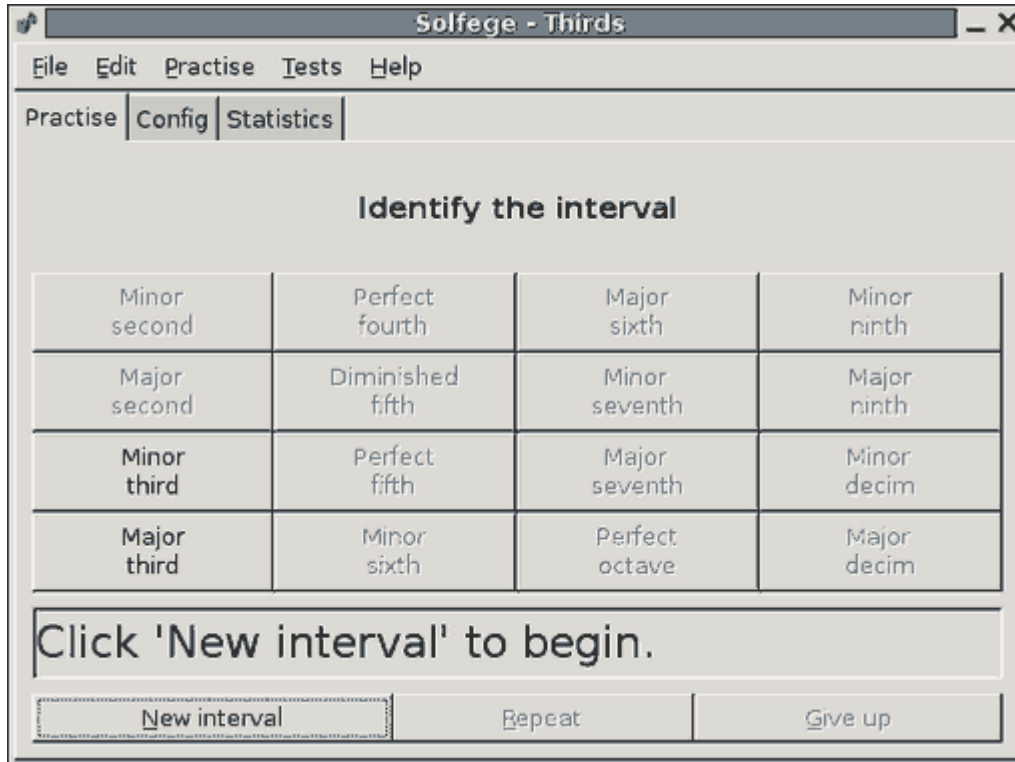
[Beats per minute](#)

[Key bindings](#)

[Sing 12 random notes](#)

[Key bindings](#)

Harmonic interval

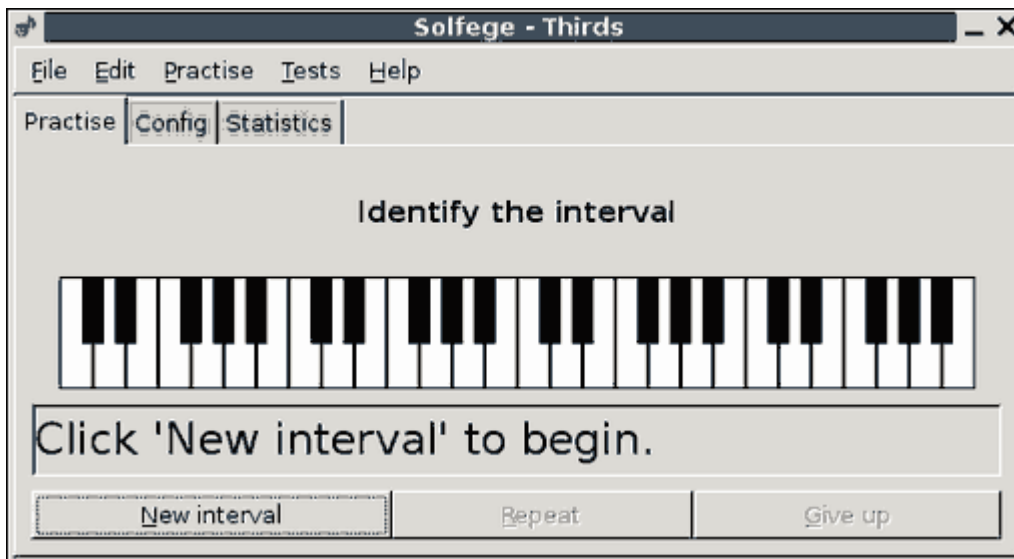


This exercise is one of the exercises you can use to practise intervals. The concept is pretty simple: You press the **New interval** button to play a random interval, and then you should tell what interval it was.

If you are using the buttons interface, then you can right-click on the buttons to hear the interval they represent.

Configuration

On the config page of the exercise, there is a combo box where you can select different ways to answer the question. Currently there is a piano, guitar, bass and a few types of accordion in addition to the default buttons interface. Below is a screenshot showing the piano interface.

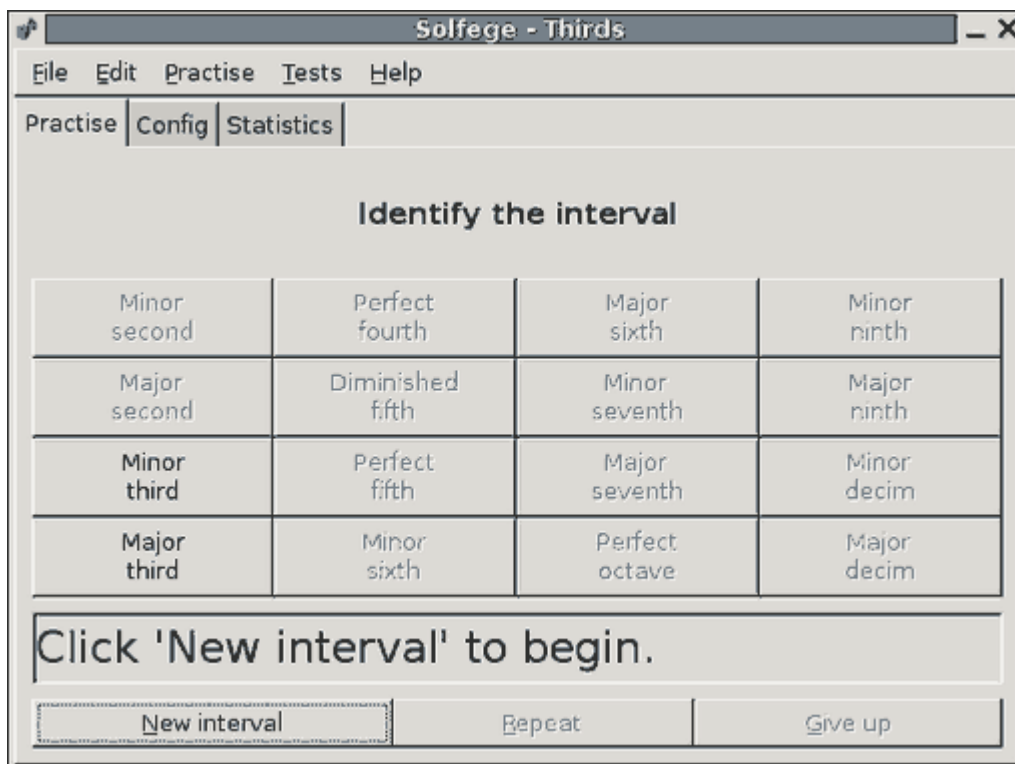


Key bindings

- New interval: **Alt-n**
- Repeat: **Alt-r**
- Repeat melodic: **Alt-m**
- Give up: **Alt-g**

Minor second: 1	Perfect fourth: 2	Major sixth: 3	Minor ninth: 4
Major second: q	Tritone: w	Minor seventh: e	Major ninth: r
Minor third: a	Perfect fifth: s	Major seventh: d	Minor decim: f
Major third: z	Minor sixth: x	Perfect octave: c	Major decim: v

Melodic interval

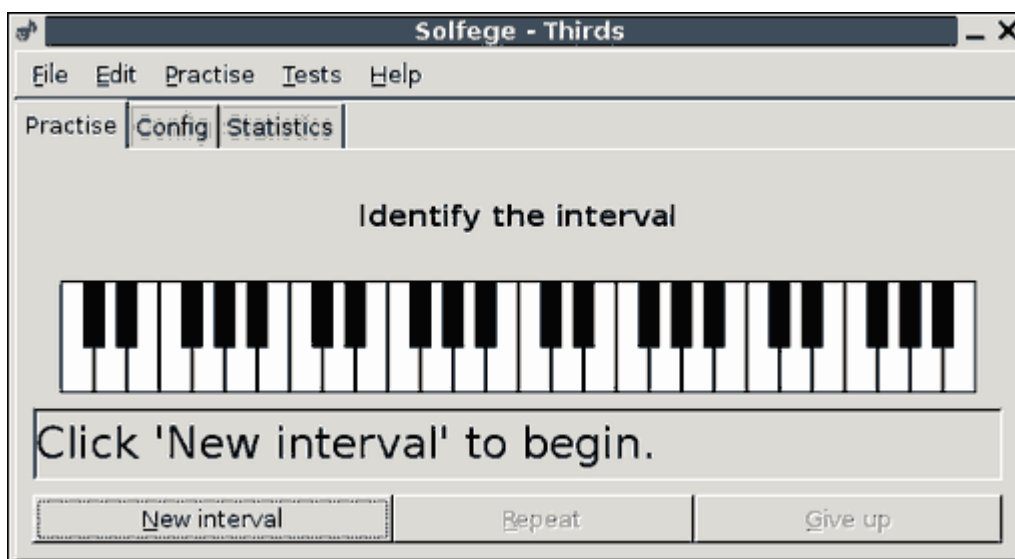


This exercise creates random intervals and you should try to identify them.

If you are using the buttons interface, then you can right-click on the buttons to hear the interval they represent.

Configuration

On the config page of the exercise, there is a combo box where you can select different ways to answer the question. Currently there is a piano, guitar, bass and a few types of accordion in addition to the default buttons interface. Below is a screenshot showing the piano interface.

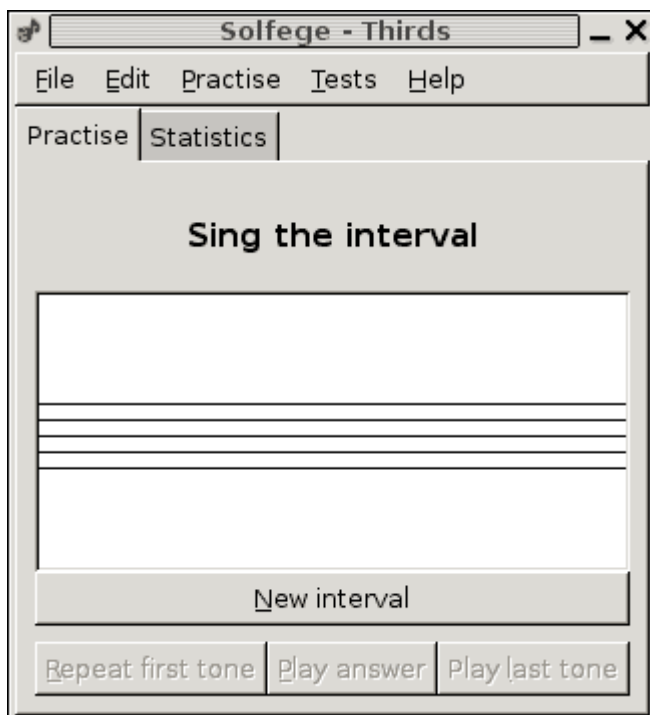


Key bindings

- New question: **Alt-n**
- Repeat: **Alt-r**
- Give up: **Alt-g**

Minor second: 1	Perfect fourth: 2	Major sixth: 3	Minor ninth: 4
Major second: q	Tritone: w	Minor seventh: e	Major ninth: r
Minor third: a	Perfect fifth: s	Major seventh: d	Minor decim: f
Major third: z	Minor sixth: x	Perfect octave: c	Major decim: v

Sing interval



In this exercise, Solfege will display one or more intervals, and you should sing them. Unfortunately, it is not yet possible to sing into a microphone and let Solfege decide if you sing correct, so you have to decide yourself if you are correct or wrong.

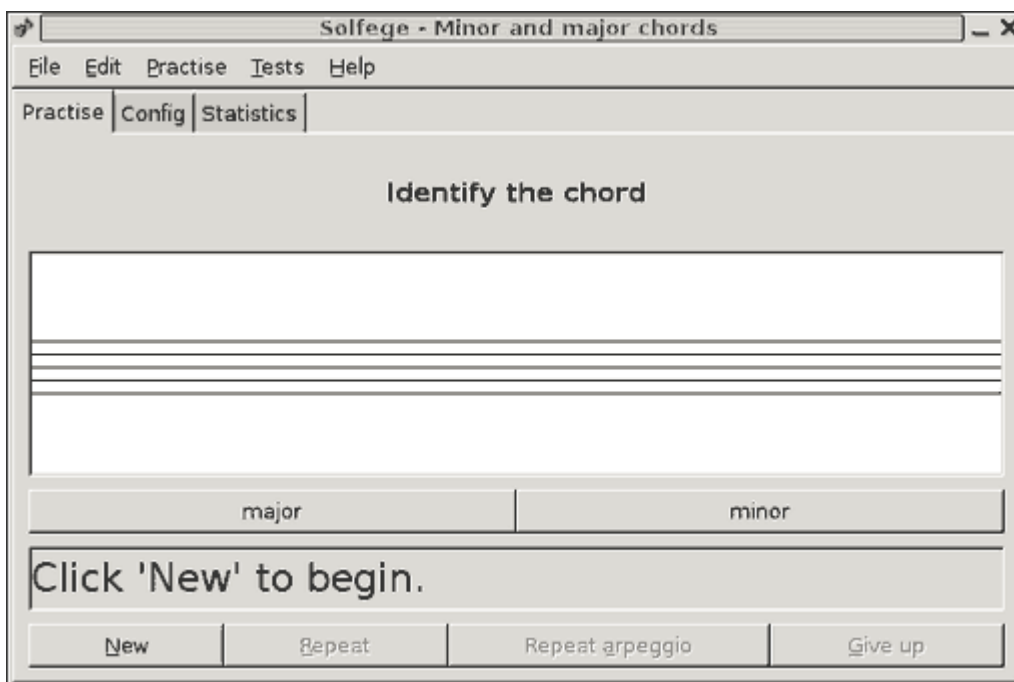
Config

The program will try to make a question where all tones are within the range the user can sing, as configured in the [preferences window](#). Sometimes, it is not possible to keep the question within this range, for example when the exercise is configured to create many intervals where all intervals go upwards.

Key bindings

- New interval: **Alt-n**
- New interval, last was correct: **Alt-n**
- New interval, last was wrong: **Alt-w**
- Repeat first tone: **Alt-r**
- Play answer: **Alt-p**
- Play last tone: **Alt-l**

Identify the chord



The purpose of this exercise is to identify the chord being played.

Start the exercise by pressing **New**. Solfege will then play a chord, and you should identify it by clicking one of the buttons below the empty staff line.

If you guess correct, the program will display the chord on the staff line and flash the message "Correct" in the status bar. Then you can click the button **New** to get a new question.

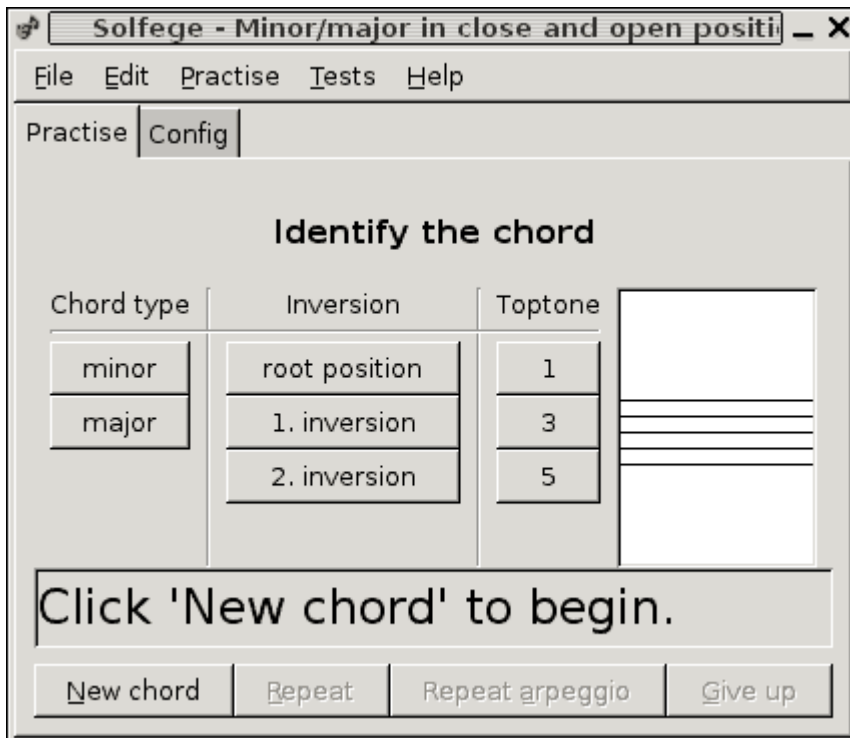
If you guess wrong the message "Wrong" will be displayed in the status bar.

Key bindings

- New chord: **Alt-n**
- Repeat: **Alt-r**

- Repeat arpeggio: **Alt-a**
- Give up: **Alt-g**

Identify the chord



This page is a generic help page for all exercises written using the chord exercise module. These exercises can ask the user for three things: chord type, inversion and top tone. The idea is that you answer this in three steps:

- Identify the chord type.
- What is the inversion?
- Which tone is the highest tone in the chord?

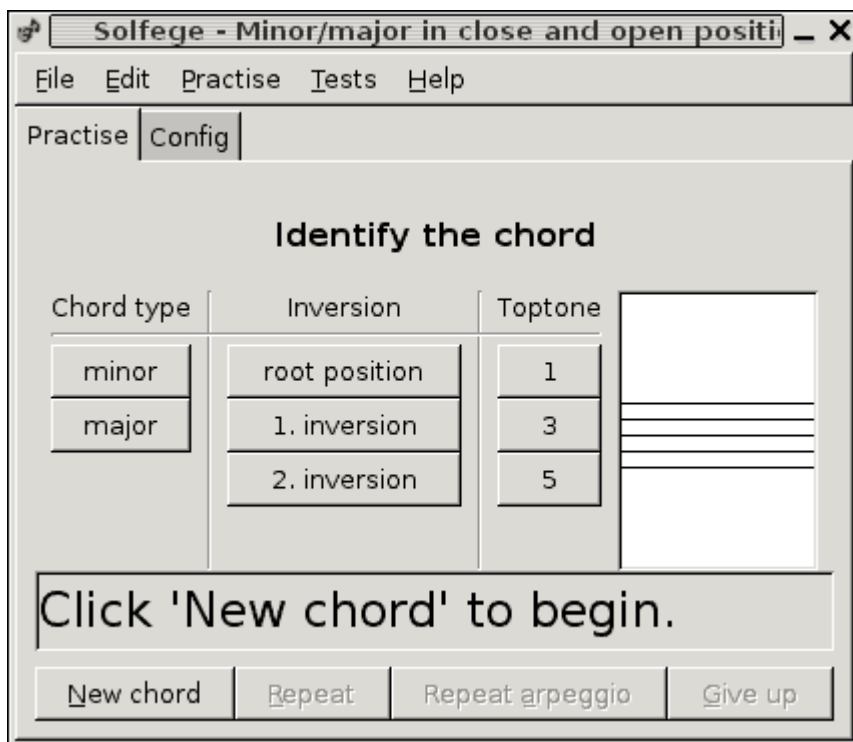
It is important that you take your time, maybe sing the chord, and identify the chord type before you try to find the inversion.

Please notice that it is also possible that an exercise will only ask for the chord type and inversion, or even only inversion and top tone.

Key bindings

- New chord: **Alt-n**
- Repeat: **Alt-r**
- Repeat arpeggio: **Alt-a**
- Give up: **Alt-g**

Multiple choice answers to music



This page is a generic help page for all exercises written using the idproperty exercise module. Exercises will typically write more specialized help text than this.

The screenshot above show one example of how an exercise can look like.

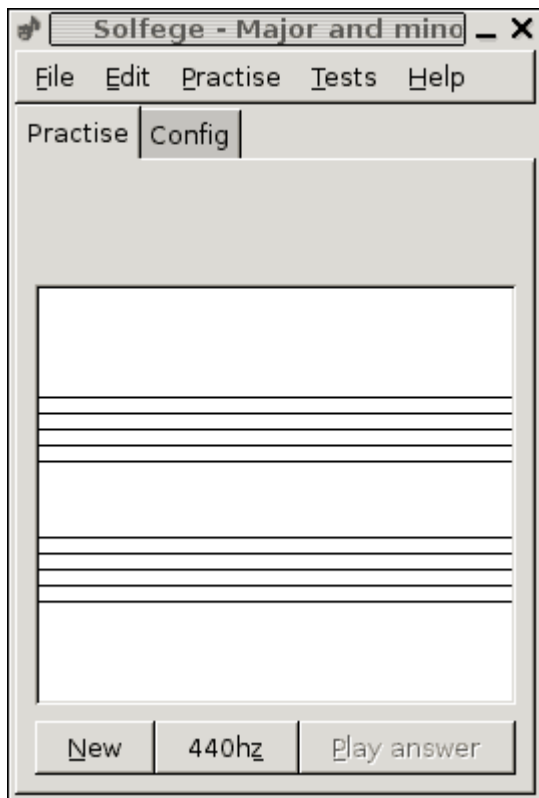
The exercise will display a question and play some music, and you have to select one answer from each column in the table of buttons. When you select the correct answer in a column, the label will turn bold, and the message "Correct" will be flashed in the status bar.

The exercise will have a **Repeat arpeggio** if one or more of the questions is of a type the program can play arpeggiated.

Key bindings

- New chord: **Alt-n**
- Repeat: **Alt-r**
- Repeat arpeggio: **Alt-a**
- Give up: **Alt-g**

Sing chord



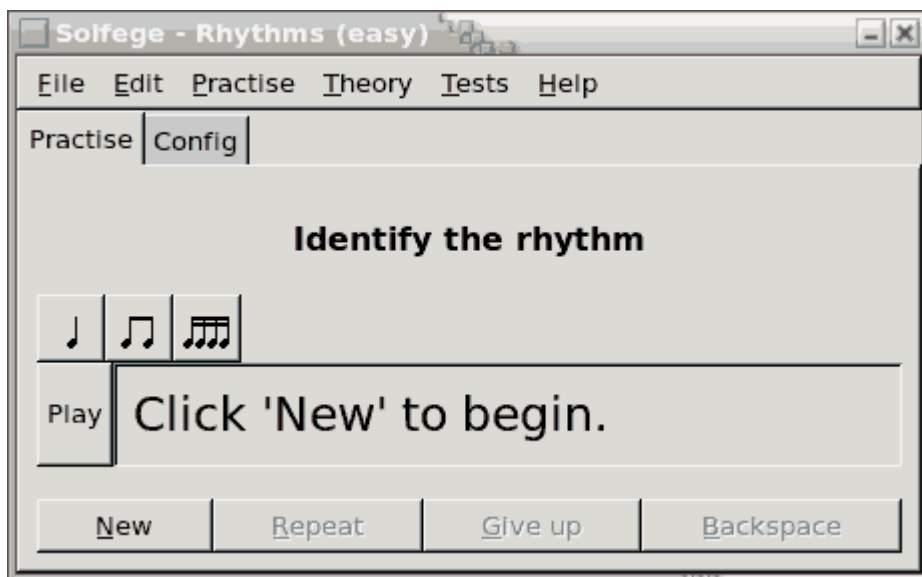
If you are conducting a choir, you have to sing the starting tones for the different voices, and if you don't have a piano near by, you have to use a tuning fork. If you are a male, you will sing the tones for the women, one octave deeper, and visa versa.

The program will play the tone A (440 hz) for you, and display a chord that you must sing. Solfege does not yet have support for microphone, so you will have to decide yourself if your answer is correct or not.

Key bindings

- New: **Alt-n**
- 440hz: **Alt-z**
- Repeat answer: **Alt-p**

Rhythm



The program play a randomly generated rhythm, and the user should reproduce the rhythm. The user enters the rhythm by clicking on buttons representing different rhythmic elements.

When you have entered enough rhythm elements, Solfege will check your answer. If everything is correct is will display a happy face, otherwise a sad face, and all wrong rhythms will be marked.

If some of your answer was wrong, everything from the first wrong element will be removed (preserving any correct rhythms in the beginning of your answer) when you click on the sad face, or when you click on the rhythm buttons on top of the page.

You can click the 'Play' button to hear your suggestion.

The questions made by this exercise are at the moment made by selecting rhythm elements randomly. This is not the best way to do it, and we hope a more clever way of generating questions will be made in a later release.

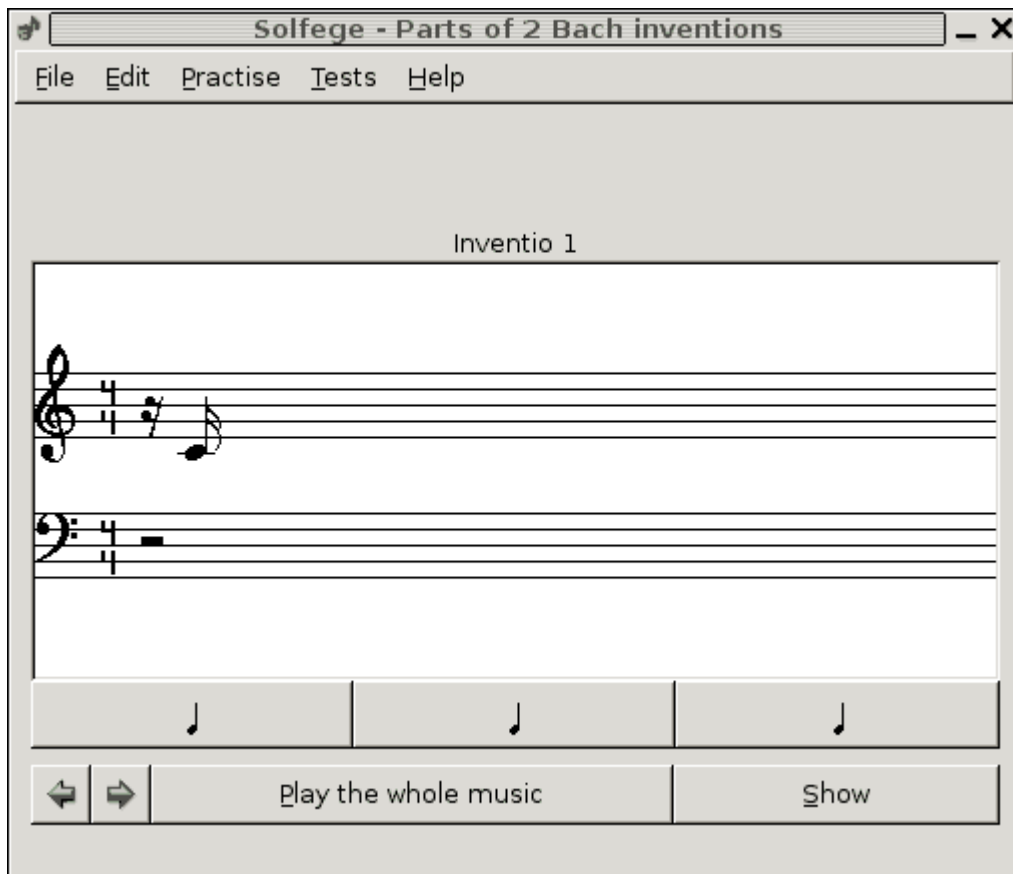
Key bindings

- New: **Alt-n**
- Repeat: **Alt-r**
- Give up: **Alt-g**
- Backspace: **Backspace**

Tap the rhythm

The program will play a randomly generated rhythm, and the user should reproduce the rhythm. The user enters the rhythm by tapping on the button labeled *Tap here*.

Dictation



This exercise is called the dictation exercise, but if the necessary lesson files are written it can be used in several ways:

- You can let Solfège play some music for you that you are supposed to write down on paper. Click on the buttons with a quarter note pixmap to repeat smaller parts of the music. You have to click on the Show button and check your notes yourself to see if you made any mistakes.
- You can use this exercise to practise sight singing: When you start the exercise, press Show and then try to sing the music. Then you can use the Play the whole music button or the quarter note buttons to let the program play the music. You have to decide yourself if you think you succeeded.

Key bindings

- Play the whole music: **Alt-p**
- Show: **Alt-s**

Scales

Scales are a complex matter. For example is the greek lydian (C-D-E-F-G-A-B-C) different from the medieval and modern lydian (C-D-E-F#-G-A-B-C). You can read about all the scales used in GNU Solfège [here](#).

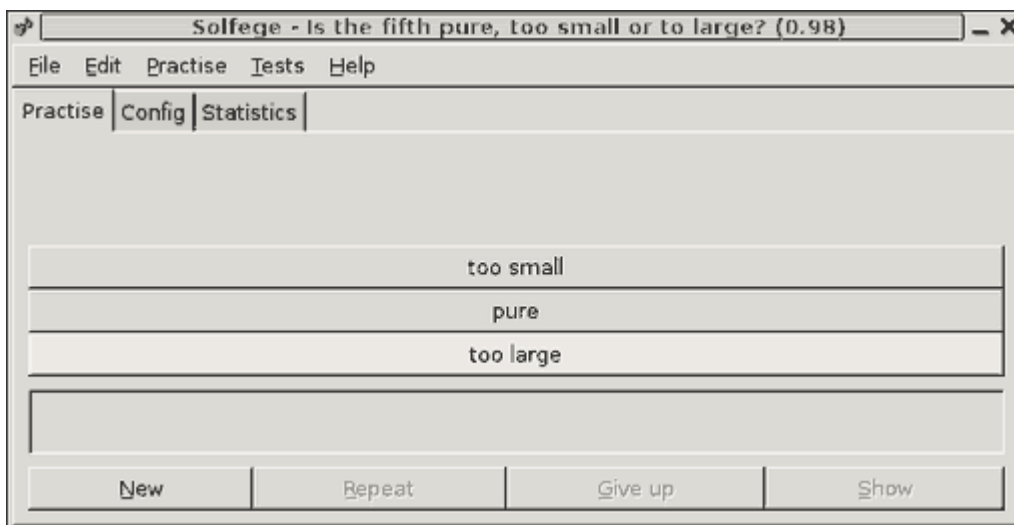
Solfège has three variants of scale exercises so far.

- Solfege will play a scale, and you should identify the scale by clicking on the button with the scale name.
- Solfege will play a scale, and you should identify the structure of the scale. You will be presented a collection of buttons labeled with a number of '1', '2' and '3'. These numbers represent the intervals minor second, major second and minor third that are between the tones of the scale.
- Solfege will play a scale, and you should identify the grade. For example may Solfege take the natural minor scale, and play it from any of the tones one the scale, and you must tell which tone it starts on.

Key bindings

- New: **Alt-n**
- Repeat: **Alt-r**
- Repeat slowly: **Alt-s**
- Give up: **Alt-g**

Intonation

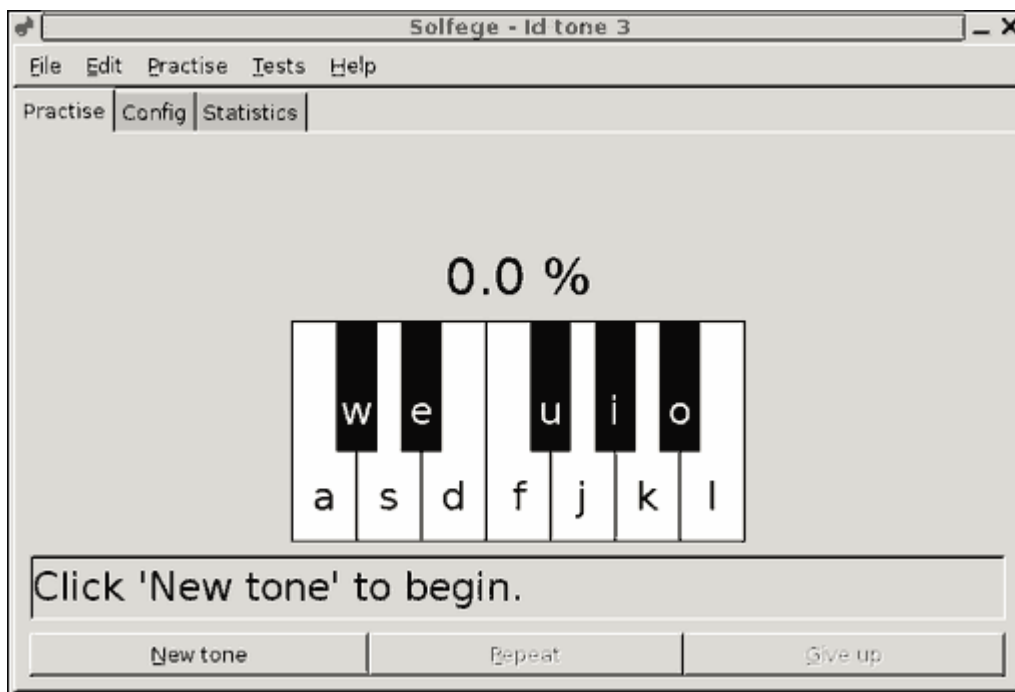


In this exercise, Solfege will play a interval, and you should tell how the interval is intonated. You do this by clicking on one of the buttons labeled 'too small', 'pure' or 'too large'. It is also possible that one of these three buttons are missing.

Key bindings

- New: **Alt-n**
- Repeat: **Alt-r**
- Give up: **Alt-g**
- Show: **Alt-s**

Identify tone



This is a combined tone memory and interval exercise. Some people believe this kind of exercise can give you perfect pitch (absolute pitch), but I don't believe so

The basics are: the program play a tone and you must identify it by comparing it with the last tone played for you.

To get you started the program will play one tone and display its name on the status bar. You identify the tones by clicking on the piano keyboard or using the keyboard shortcuts that are the letters written on each key.

Right click on the piano keyboard to hear a note without actually guessing it. (Some will call that cheating....)

Manual configuration

You can configure the idbyname exercise as you like if you select **Misc → Configure yourself → Id tone**.

There are several ways you can use this exercise. Personally, I have not used this exercise very much, and the sections below are only suggestions.

Note by note

Start with only the notes [c-d-e](#) at weight 1. When your score is at least 96% correct, you [add the tone f](#) and continue. The menu **Misc → Identify tone**, has exercises that will add one and one tone until you practise with all 12 tones.

Heavy A

'Heavy A' describes another way to practise. It requires that you select **Misc → Configure yourself → Id tone**.

Configure with the tone a at weight 11 (or higher) and the rest of the tones at weight 1. This way the program will play the tone a very often, so you will remember the tone, and then you use a as a reference tone to identify the other tones. When you have practised a while, you can reduce the weight of a to make the exercise harder.

Config

On the top of the config page you tell the program how important the different tones are. If you for example give the tone a 11 points and the rest 1 point each, then $(11+11*1)/11*100 = 50\%$ of the random tones will be an a.

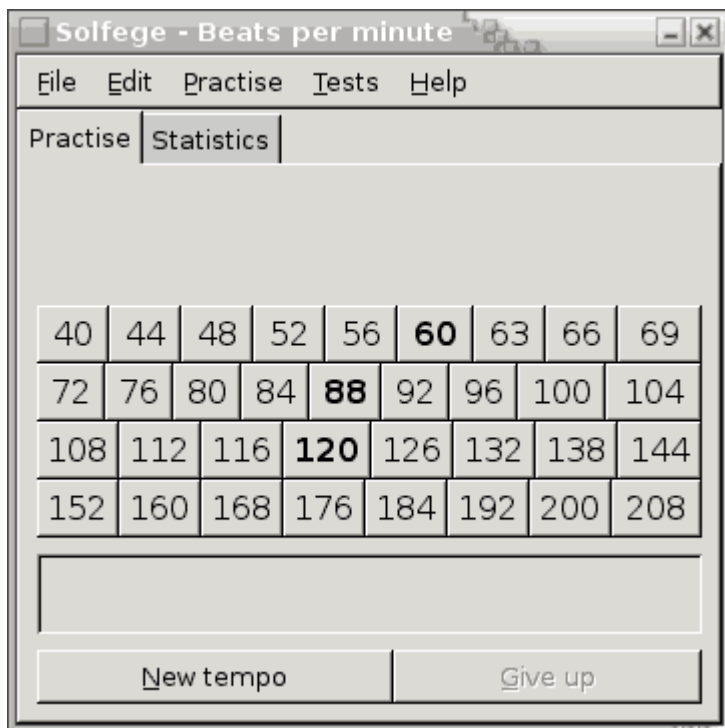
Below that you select what octaves the random tones can be from.

Then you can select if Solfege should give you a new question automatically when you have solved the old.

In the frame below you can set some pretty self explaining options about what happens if you answer wrong.

The keyboard shortcuts can be configured from config file. You can find its location from **Help** → **File locations**.

Beats per minute



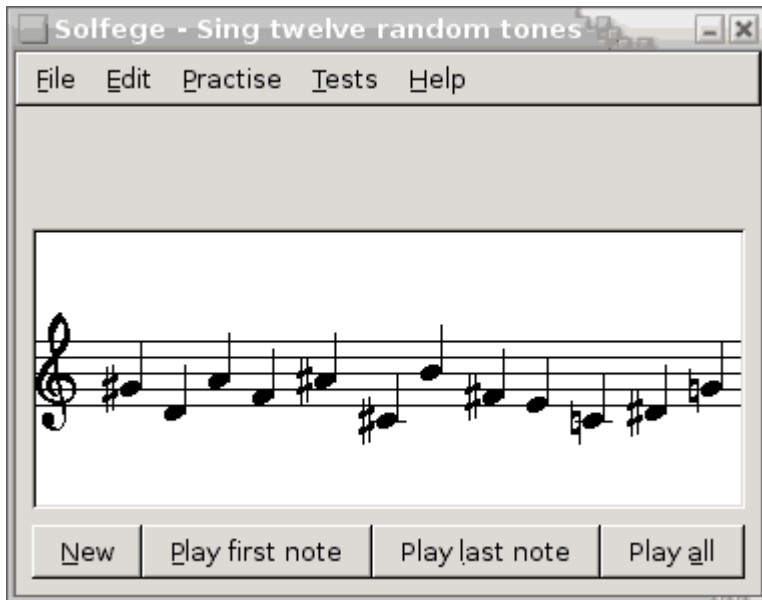
The program will play a tempo, like a methronome. You should try to guess how many beats per minute is played. Each button represents one tempo, and the program will only play in tempos that has a button with bold text. Right-click on buttons to change the status of a tempo.

Note: the rhythm depends on the `gtk_timeout_add` function to play the rhythm, so it is not very precise.

Key bindings

- New tempo: **Alt-n**
- Give up: **Alt-g**

Sing 12 random notes

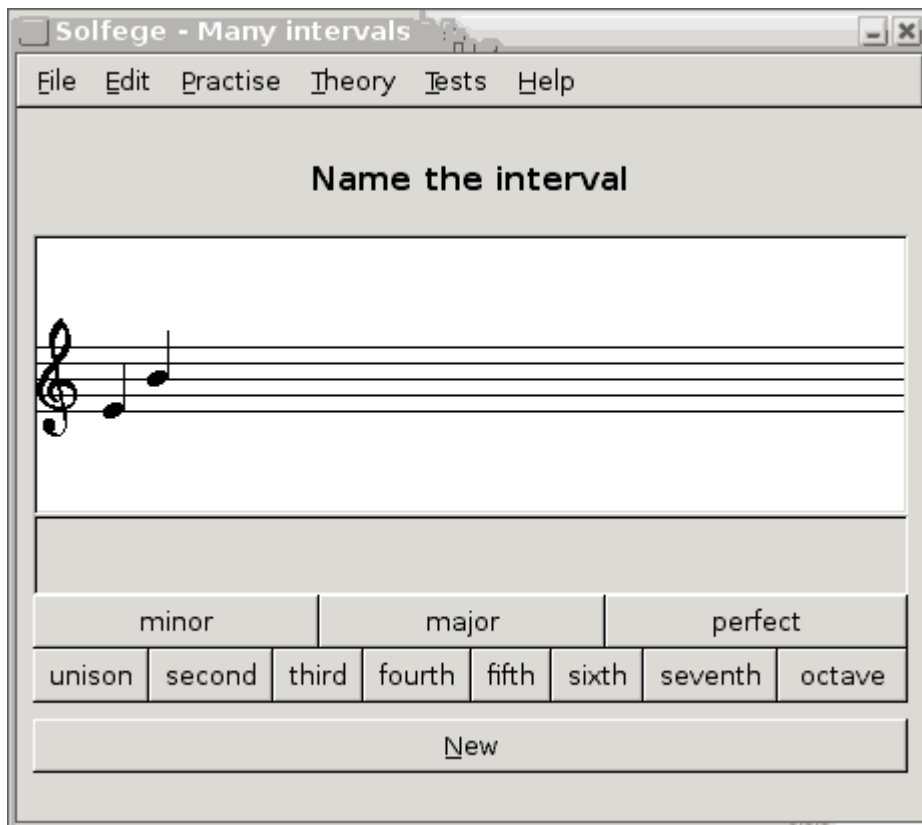


In this exercise, the program will display all the twelve tones in the scale in a random order and play the first one. Then you should sing all the notes and see if the last note matches. So this is more like an exam in sight singing than an exercise for learning how to sing the intervals. For that you should try some of the other interval exercises.

Key bindings

- New: **Alt-n**
- Play first note: **Alt-p**
- Play last note: **Alt-l**
- Play all: **Alt-a**

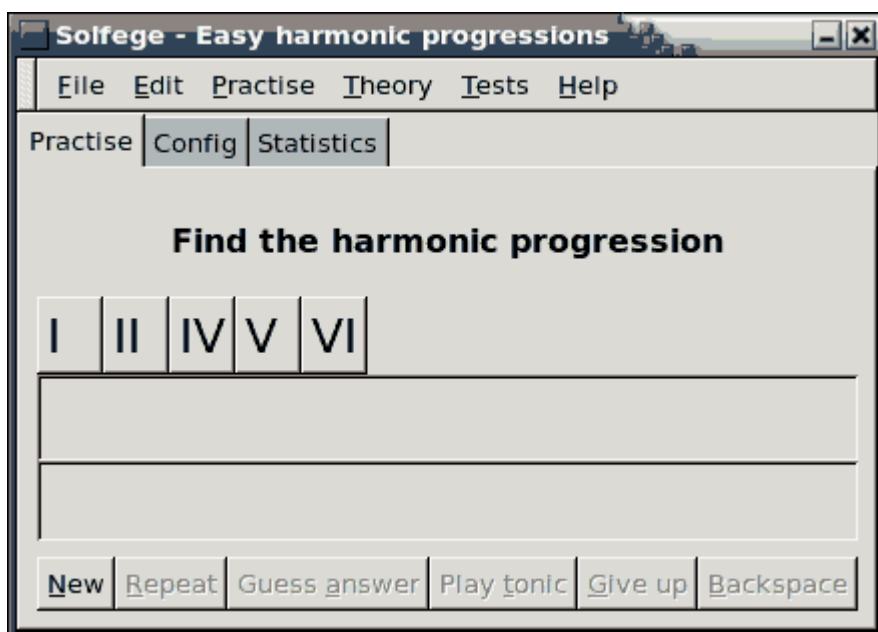
Name intervals



In this exercise, Solfege will display and play an interval, and you should identify the interval. This is a music theory exercise, and not an ear training exercise. To learn how to name intervals you should read [the section called "Intervals"](#).

You identify the interval by clicking on one button telling the specific name and the general name.

Harmonic progression dictation



In this exercise, Solfege will play some music, and you must click on the buttons to build a representation of the harmonic progressions in the exercise.

Chapter 3. Music theory

Table of Contents

[Scales](#)

[Intervals](#)

[Seconds](#)

[Thirds](#)

[Fourth](#)

[Fifth](#)

[Sixths](#)

[Sevenths](#)

[Inverting intervals](#)

Scales

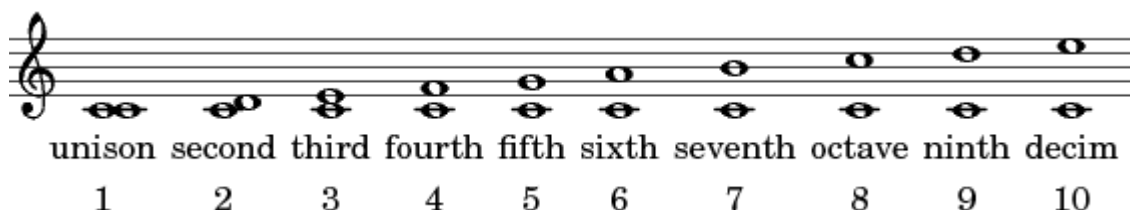
Davide Bonetti has contributed a large set of scale exercises and some pages describing all the scales. You can see the pages [here](#).

Intervals

In music theory we use the word interval when we talk about the pitch difference between two notes. We call them harmonic intervals if two tones sound simultaneously and melodic intervals if they sound successively.

Interval names consist of two parts. Some examples are "major third" and "perfect fifth". In Walter Piston's "Harmony" the two parts are called *the specific name* and *the general name* part. Wikipedia talks about *interval quality* and *interval number*. I have seen people talk about an interval's *numerical size*. I am a little unsure what the best terms to use are, because English is my second language. Comments, and improvements to this article, are very welcome.

You find the general name by counting the steps on the staff, ignoring any accidentals. So if the interval you want to name goes from E to G#, then we count to 3 (E F G) and see that the general name is *third*.



The specific name says the exact size of the interval. Unisons, fourths, fifths and octaves can be diminished, pure or augmented. Seconds, thirds, sixths and sevenths can be minor, major, diminished or augmented. A minor interval is one semitone smaller than a major interval. A diminished interval is one semitone smaller than a pure or a minor interval, and an augmented interval is one semitone larger than a pure or major interval.

Accidentals change the size of intervals. The interval becomes one semitone larger if you add a sharp to the highest tone or a flat to the lowest tone. And it becomes one

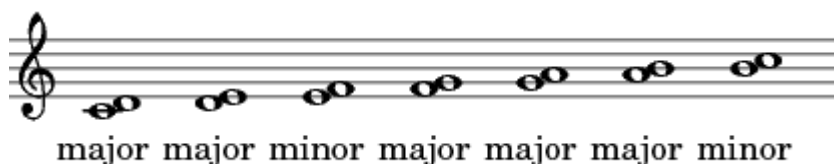
semitone smaller if you add a flat to the highest tone or a sharp to the lowest tone. In the following sections naming of the intervals will be shown in greater detail.

Seconds

Seconds are easy to recognise: the two notes are neighbours on the staff. One note is on a staff line, and the other one is in the space above or below. A minor second is one semitone step, also called a half step. A major second is two semitone steps, also called a whole step.

To learn to identify seconds, you first have to learn which seconds there are between the natural tones. As you can see in [Figure 3.1, ""](#), only the intervals E-F and B-C are minor seconds. The rest are major intervals. You can check that [Figure 3.1, ""](#) is correct by looking at a piano. You will see that there are no black keys between E and F and between B and C.

Figure 3.1.



If the second have accidentals, then we have to examine them to find out how they change the size of the interval. Let us identify a few intervals!

Figure 3.2.



We remove the accidental from the interval in [Figure 3.2, ""](#) and see that the interval F-G is a major second. When we add the flat to the highest tone, the interval becomes one semitone smaller, and becomes a minor second.

Figure 3.3.



We remove the accidentals, and see that the interval A-B is a major second. You still do remember [Figure 3.1, ""](#), don't you? Then we add the flat to the A, and the interval become a augmented second. And when we add the flat to the B, and the interval becomes a major second.

Figure 3.4.

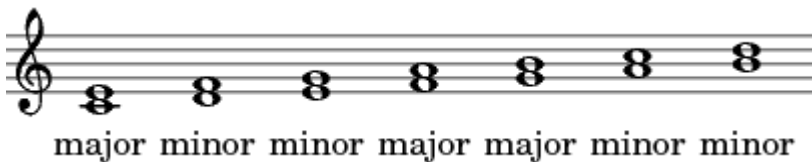


We remove the accidentals, and see that the interval E-F is a minor second. When we add a flat to the lowest tone, the interval becomes one semitone larger, and becomes a major second. And when we add a sharp to the highest tone, the interval becomes one semitone larger, and becomes an augmented second.

Thirds

A minor third is one minor and one major second, or three semitones. A major third are two major seconds, or four semitone steps. [Figure 3.5, ""](#) show the thirds between all the natural tones. You should memorise the major intervals, C-E, F-A and G-B. Then you know that the other four intervals are minor.

Figure 3.5.



Then you examine the accidentals to see if they change the specific name. This is done exactly the same way as for seconds.

Fourth

A pure fourth is $2\frac{1}{2}$ steps, or two major seconds and a minor second. [Figure 3.6, ""](#) show all fourths between natural tones. You should memorise that the fourth F-B is augmented, and that the other six are pure.

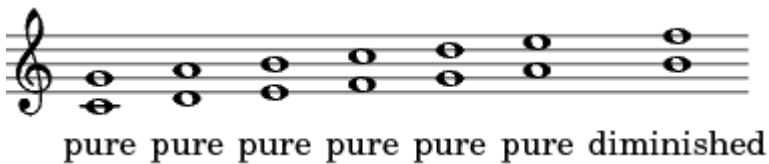
Figure 3.6.



Fifth

A pure fifth is $3\frac{1}{2}$ steps, or three major seconds and a minor second. [Figure 3.7, ""](#) show all fifths between natural tones. You should remember that all those intervals are pure, except B-F that is diminished.

Figure 3.7.



If the interval has accidentals, then we must examine them to see how they change the size of the interval. A diminished fifth is one semitone smaller than a pure interval, and a augmented fifth is one semitone larger. Below you will find a few examples:

Figure 3.8.



We remember from [Figure 3.7, ""](#) that the interval B-F is a diminished fifth. The lowest tone in [Figure 3.8, ""](#) is preceded by a flat that makes the interval one semitone larger and changes the interval from a diminished to a pure fifth.

Figure 3.9.



We know from [Figure 3.7, ""](#) that interval E-B is a perfect fifth. In [Figure 3.9, ""](#) the E has a flat in front of it, making the interval augmented. But then the B is preceded by a double flat that makes the interval two semitone steps smaller and changes the interval to a diminished fifth.

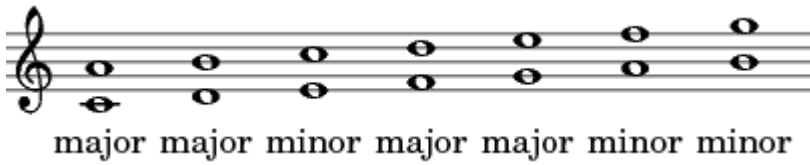
Sixths

Sixths are easiest identified by [inverting the interval](#) and identifying the third. Then the following rule apply:

- If the third is diminished, then the sixth is augmented
- If the third is minor, then the sixth is major
- If the third is major, then the sixth is minor
- If the third is augmented, then the sixth is diminished

If you find inverting intervals difficult, then you can memorise that the intervals E-C, A-F and B-G are minor. The other four are major. Then you examine the accidentals to see if they change the specific name. This is done exactly the same way as for seconds.

Figure 3.10.

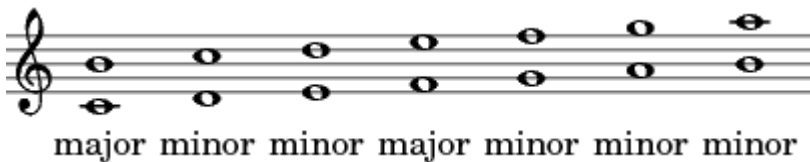


Sevenths

Sevenths are identified the same way as sixths. When you invert a seventh, you get a second.

If you find inverting intervals difficult, then you can memorise that the intervals C-B and F-E are major. The other five are minor. Then you examine the accidentals to see if they change the specific name. This is done exactly the same way as for seconds.

Figure 3.11.



Inverting intervals

You invert an interval when you move the lowest tone of an interval one octave higher or the highest tone one octave lower. The general name changes this way:

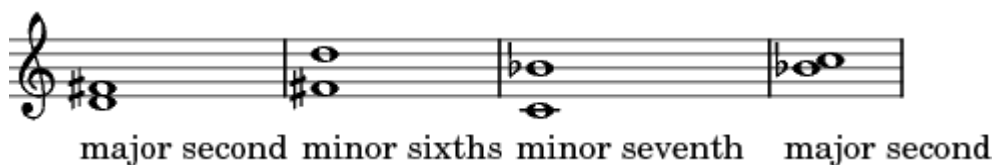
- Second becomes seventh.
- Third becomes sixth.
- Fourth becomes fifth.
- Fifth becomes fourth.
- Sixth becomes third.
- Seventh becomes second.

The specific name changes this way:

- Diminished becomes augmented.
- Minor becomes major.
- Perfect stays perfect.
- Major becomes minor.
- Augmented becomes diminished.

Below are two examples, a major third is inverted and becomes a minor sixth, and a minor seventh is inverted and becomes a major second.

Figure 3.12.



Chapter 4. Extending GNU Solfege

Table of Contents

[Introduction](#)

[Lesson files](#)

[File encoding](#)

[Comments](#)

[Types](#)

[Global variables](#)

[Lesson file contents](#)

[Header block](#)

[Question block](#)

[music objects](#)

[Functions](#)

[Operators](#)

[The chord module](#)

[The compareintervals module](#)

[The dictation module](#)

[The elembuilder module](#)

[The element block](#)

[The header block](#)

[The question block](#)

[The harmonicinterval module](#)

[The idbyname module](#)

[The idproperty module](#)

[The idtone module](#)

[The melodicinterval module](#)

[The nameinterval module](#)

[The rhythm module](#)

[The rhythmtapping module](#)

[The rhythmtapping2 module](#)

[The singsanswer module](#)

[The singchord module](#)

[The singinterval module](#)

[Midi instrument names](#)

[Percussion instrument names](#)

Introduction

GNU Solfege is written so that it can easily be extended, even if you do not know any

computer programming. The steps are:

- Create a lesson file.
- Create a learning tree for your own lesson file. You do this only once.
- Add the lesson file to the learning tree.

Read [the section called “Lesson files”](#) for details on creating lesson files. The easiest way to get started is to take one of the existing lesson files, and modify it. Select **File locations** on the **Help** menu to find out where the included lesson files are stored, and where you should save the additional files you create. It is important to store the lesson files you create in the directory intended for user created lesson files, and not in the applications directory. This to avoid losing files when you upgrade the program.

The file paths is not written here in the user manual because the file path depends on which operating system you run.

You create a learning tree by opening the learning tree editor. Select **Learning tree** from the **Edit** menu. Then click the **New** button and enter a file name. Solfege will suggest a directory to save learning trees, and unless you a good reason to do so, I suggest you save the file there. You can find the location of this directory in the File locations dialog.

Then you create a menu and a submenu with the learning tree editor, and finally adds the lesson file to the selected submenu by clicking the **Add lesson** button.

Lesson files

In GNU Solfege, each exercise is created by a lesson file interpreted by one of the exercise modules.

Exercise modules

harmonicintervals

Train harmonic intervals.

melodicintervals

Train one or more melodic intervals.

singinterval

This is an exercise where the program display an interval and play the first tone. Then the user should sing the interval, and then click a button to hear the correct answer. There is no microphone support yet.

idbyname

This is a very generic exercise. In its most basic form, the program will play some sound, and you have to select among several buttons that in some way represents the music.

chord

The chord module act as a specialized idbyname module. The difference is that with the chord module you can write lesson files where the user should tell what inversion the chord is in, and what the top tone is.

chordvoicing

A two-step exercise. First you should identify the chord. Then you should stack the tones in the chord in the correct order.

compareintervals

Solfege plays two intervals, and you should say which one is largest.

rhythm

A simple rhythm exercise. Solfege will randomly generate rhythm patterns that the user should recreate by clicking on buttons.

dictation

harmonicprogressiondictation

idtone

identifybpm

twelvetone

singchord

File encoding

Solfege by default expects the content of lesson files to be in UTF-8 encoding. Modern editors often let you specify the encoding in the "Save As" dialog. One example is gedit. Other programs, like vim and emacs let you specify the encoding inside the text file.

If this sounds complicated, you can safely ignore the whole encoding issue if you restrict yourself to use only standard ascii characters. That is only the letters a to z.

If you create lesson files with a different encoding, you have to declare the encoding in a special comment at the top of the file. This because Solfege and the tools used to translate Solfege cannot guess the encoding safely. We follow the same conventions as the Python language. See [PEP-0263](#) for the details.

What you have to do is add a comment to one of the first two lines of the lesson file, where part of the line matches `coding=encoding` or `coding: encoding`. Extra characters on the line are ignored, so if you use the emacs or vim editors, you can conveniently tell the editor about the file encoding. The following example sets the charset to ISO 8859-1, a charset commonly used in many west-european languages:

```
# -*- coding: iso-8859-1 -*-
```

Russians might want to use koi8-r:

```
# -*- coding: koi8-r -*-
```

Same as above, but in a format that works with the vim:

```
# vim: set fileencoding= koi8-r :
```

The program use the python libs to convert to unicode, so it should understand almost any encoding you can think of. If you see some characters are missing, for example when the name of questions are displayed on buttons, then most likely you have done something wrong with the encoding.

Comments

Everything after # on a line is ignored. Example:

```
# This line is ignored. The next line is not.  
question { bla bla }
```

Types

Strings

Strings are quoted with the " character. Example:

```
"this is a string"
```

Use tripple quotes for strings that contain line breaks, or if the string itself has to contain the " character:

```
description = """<h1>Long desription<h1> This lessonfile need  
very much descriptions. Qoutes (") are ok here. bla bla bla"""
```

NB: All strings have to be unicode strings. If you get error messages like this one:

```
In line 21 of input: does not recognise this string ';' as a valid token.'  
(line 20): question {  
(line 21): question {  
(line 22):   name = _("Ionia")
```

then you must check the encoding of your file, and maybe you should read [the section called "File encoding"](#). You can change the encoding of a file using the **iconv** program:

```
iconv -f YOUR_ENCODING -t utf8 your.file
```

Tempo

The tempo of music is entered as bpm/beat len. The following example will set the tempo to 120 beats per minute, each beat being a quarter note.

```
tempo = 120/4
```

Global variables

Global variables can save you a few key strokes.

```
s = "\score\relative c'{ %s }
question {
# instead of music = music("\score\relative c'{ c d e f g2 g2 }")
music = music(s % "c d e f g2 g")
}
```

Lesson file contents

A lesson file consist of one header block and zero or more question blocks:

```
header {
  ASSIGNMENT
  ASSIGNMENT
  ...
}
question {
  ASSIGNMENT
  ...
}
```

Header block

The header block can be placed anywhere in the file, but by convention it should be the first block in the file.

Variables shared by many exercise modules

`module`

Tell what exercise module that will run the lesson file. This variable is required for all lesson files. (The variable was added in Solfege 2.9.0 where it replaced the content variable.). Example:

```
module = idbyname
```

`lesson_id`

Each file need a unique identifier. The identifier can be any string you like, and if you don't add one, Solfege will add one for you. Solfege will also offer to create a new `lesson_id` if you have two files with identical `lesson_id`. Example:

```
lesson_id = "5b30c9ae-09f1-40b3-9333-4789638dc851"
```

`version`

Tell the version of solfege the lessonfile is known to work with. This variable is not required, but it should be used because it can (but don't guarantee to) help avoid trouble if the lesson file format changes in the future. Example:

```
version = "3.0.7"
```

`title`

Short one-line description that will be used for creating the menu entry for the exercise. You should add this to all lesson files. Example:

```
title = "Minor and major chords in root position"
```

lesson_heading

A short heading that will be displayed above the exercise. It should say what the purpose of the exercise is. Some modules provide a default value, others leave the string empty. Example:

```
lesson_heading = _("Identify the chord")
```

help

This variable say which help file from the user manual will be displayed when the user presses F1. Example:

```
help = "idbyname-intonation"
```

By default, Solfege will display the help file that has the same name as the exercise module being used in the lesson file.

theory

This variable say which help file from the user manual will be displayed when the user presses F3. Pressing F3 should display music theory about the exercise. Don't include this variable if there are no music theory written. Example:

```
theory = "scales/maj"
```

random_transpose

In some exercises the program can transpose the music to create variation. The default value is yes. (The default value changed from no to yes in Solfege 3.0.)

Used in modules: chord, chordvoicing, harmonicprogressiondictation, idbyname, singanswer, singchord

Possible values

```
random_transpose = no
```

No transposition will be done.

```
random_transpose = yes
```

The exercise will do random transposition. What kind of transposition depends on the exercise, but you get a ok result from this. This is the default value.

```
random_transpose = accidentals, INTEGER1, INTEGER2
```

Transpose the question by random and make sure the key signature of the question does not get more than a certain number of accidentals. In this context, the number of accidentals can be described by an integer value. A

negative value denote a number of flats (b), and a positive number denote a number of sharps (#). Zero mean no accidentals. The integers INTEGER1 and INTEGER2 defines a range of allowed number of accidentals.

For this transposition mode to work properly, the music in the lessonfile has to be in the keys c major or a minor, or the question must have a key variable telling the key signature.

`random_transpose = key, INTEGER1, INTEGER2`

Transpose the music INTEGER1 steps down or INTEGER2 steps up the circle of fifth. In this context up is more sharps and down is more flats. This is real transposition where both the key and the notes are transposed.

`random_transpose = semitones, INTEGER1, INTEGER2`

Transpose the music at most INTEGER1 semitones down or INTEGER2 semitones up. This is real transposition where both the key and the notes are transposed. You will easily end up with music in the keys with LOTS of accidentals.

`enable_right_click = no`

By default, Solfege will let the user right-click on buttons to hear the music they represent without guessing. Set this variable to no for lesson files where it does not make sense, for example in a `idbyname` lesson file where many questions have the same name.

Modules: `idbyname`, `chordvoicing` and `chord`.

`disable_unused_intervals = no`

By default, Solfege will make the buttons insensitive for intervals that are not being asked. Set this variable to no if you want all buttons to be sensitive.

Modules: `harmonicinterval` and `melodicinterval`.

`ask_for_intervals_0`

Select which intervals to ask for. 1 for minor second, 2 for major second, 3 or minor third etc. Use a negative number for descending intervals. To ask for more than one interval create the variables `ask_for_intervals_1`, `ask_for_intervals_2` etc. In the following example Solfege will ask for two intervals. The first will be either a minor second or a major second, both intervals going up. And the second interval will be either major second or minor third, both intervals going down.

```
ask_for_intervals_0 = [1, 2]
ask_for_intervals_1 = [-2, -3]
```

Modules: `melodicinterval` and `singinterval`.

`intervals`

This variable tell which intervals should be asked for in exercises using the

harmonicinterval module. 1 for minor second, 2 for major second, 3 or minor third etc. Example that will practise thirds:

```
intervals = [3, 4]
```

Modules: harmonicinterval.

test

This variable defines the test for the exercise. In a test, Solfege will ask all the questions in the lessonfile a number of times. This variable is always used together with test_requirement. In the following example, each question will be asked 3 times:

```
test = "3x"
```

Modules: harmonicinterval, idbyname, melodicinterval and singinterval.

test_requirement

This variable defines how large percentage of the questions has to be answered correctly to pass the test. Example:

```
test_requirement = "90%"
```

Modules: harmonicinterval, idbyname, melodicinterval and singinterval.

have_repeat_arpeggio_button = yes

Set to yes if you want the exercise to have a "Repeat arpeggio" button.

Modules: singanswer.

have_music_displayer = yes

Set to yes if you want the question to have a music displayer.

In the idbyname module, setting this variable will add a music displayer where the program will display the answer when the user gives up or answers the question correctly. You might also want to read about [at_question_start](#).

In the singanswer module, setting this variable will add a music displayer where the music will be displayed when the question is displayed.

Modules: idbyname, elembuilder and singanswer.

at_question_start

This variable changes what happens when the user clicks **New**. By default, Solfege will play the music when the user clicks **New**, and only display the music when the question is answered correctly and the have_music_displayer variable is set to yes. Setting this variable will also set have_music_displayer to yes.

```
at_question_start = show
```

The exercise will get a **Play music** button. When the user clicks **New** the music will be displayed in the music displayer, but no music is played. Click **Play music** to hear the music.

at_question_start = play

The exercise will get a **Display music** button. When the user clicks **New** the music is played. Click **Display music** to see the music.

at_question_start = show, play

When the user clicks **New** the music is both played and displayed.

Modules: idbyname, elembuilder and rhythmtapping2.

vmusic




































This variable holds a representation of the question intended to be displayed. This can be necessary if the music is a .wav or .mp3 file. It will be used when the user clicks Show music or when the question is answered correctly (if we have a musicdisplayer). Added to idbyname in Solfege 2.5.1 and to elembuilder in 3.9.2.

Modules: idbyname and elembuilder.

rhythm_elements

A list of integers (1-34) telling what elementes we should use when creating questions. Example:

rhythm_elements = 0, 1, 2, 3, 4

0:  , 1:  , 2:  , 3:  , 4:  , 5:  , 6:  , 7:  , 8:  , 9:  , 10:  , 11:  , 12:  , 13:  , 14:  , 15:  , 16:  , 17:  , 18:  , 19:  , 20:  , 21:  , 22:  , 23:  , 24:  , 25:  , 26:  , 27:  , 28:  , 29:  , 30:  , 31:  , 32:  , 33:  , 34: 

Modules: rhythm and rhythmtapping2

Variables that has been obsoleted

number_of_intervals = INTEGER

Made obsolete in Solfege 3.1.5. Solfege will find this number automatically now, so this variable is ignored.

Question block

Variables you can define in the question block

name

Questions written for the [idbyname](#) or the [chord](#) exercise modules need a name.

music

For most lesson files the music representing the question is assigned to this variable. Note that there is a shortcut. Instead of:

```
question {
  name = "Lisa gikk til skolen"
  music = music(...)"
}
```

you can write:

```
question {
  name = "Lisa gikk til skolen"
  music(...)
}
```

Music objects are documented in [the section called "music objects"](#).

tempo

Set the tempo for this questions music. The variable is defined "beats per minute" / "notelen per beat". Example:

```
tempo = 150 / 4
```

This variable can also be defined globally for the whole lesson file. Do do so you should put it in the beginning of the file, outside any question blocks.

Modules: [idbyname](#), [chord](#), [chordvoicing](#) and [rhythmtapping](#).

instrument

By default, Solfege will use the instrument specified on the [preferences window](#) when playing questions. This variable let you select a different instrument. Example:

```
instrument = "cello", 100
```

The instrument name has to be quoted. The integer is the velocity of the tones, and it should be in the range 0-127. You can see a list of instrument names in [the section called "Midi instrument names"](#). For lesson files where it makes sense, it is possible to specify three set of instruments. The following example will play bass for the lowest tone, piano in the middle and clarinet on the top tone:

```
instrument = "bass", 100, "acoustic grand", 100, "clarinet", 100
```

This variable can also be defined globally for the whole lesson file. Do do so you should put it in the beginning of the file, outside any question blocks.

Modules: [idbyname](#), [chord](#), [singanswer](#) and [chordvoicing](#)

set

The set variable is used by some exercise modules to select which question to

play when the user right clicks on one of the answer buttons. This can be useful if the lesson file has many questions with the same name, and you want solfege to play the question that is most closely related to the question being asked. You can assign whatever value you want. A good suggestion is to use integers.

In lesson files that does not use the set variable, solfege will play the first question it can find with the same name as the button the user right clicks on.

If the lesson file uses the set, or more precisely, if the question being asked has the variable defined, the program will first try to find a question where the set variable matches the question being asked, and the name matches the button clicked. If no match is found, the program will select a question to play as if the set variable was not used at all.

Modules: idbyname and chordvoicing.

music objects

Each question in your lesson files will define one or more music objects.

music

This is music entered completely following the music format FIXME spec. This means you have to enter complete code with a `\staff` command. Example:

```
variable = music("\staff\relative c' { c' d' }")
```

chord

Enter the tones from the lowest to the highest tone, like this:

```
variable = chord("c' e' g'")
```

satb

This type of music is used by the singchord exercises. It let you say which tones of a chord the different voices in a choir will sing. Take this, for example:

```
variable = satb("c''|e'|g|c")
```

The `c''` will be sung by the soprano, `e'` by the alto, `g` by the tenor and `c` by the bass. Please notice that when this music is played in arpeggio, the tones to be sung by the women, will be played one octave deeper, of the user is a male. And vice versa if the user is a female or a child.

voice

This musictype saves some key strokes if you want to enter a melody.

```
variable = voice("c'4 c' g' g' | a' a' g'2")
```

is the same as

```
variable = music("\staff{ c'4 c' g' g' | a' a' g'2")
```

rvoice

rvoice is similar to voice except that the music is in \relative mode, relative to the first tone. The following two statements produce the same music:

```
variable = rvoice("c'4 c g' g | a a g2")
\staff\relative c' { c4 c g' g' | a a g2 }
```

percussion

This music object provides a simple way to play rhythms with percussion instruments. Each tone represents a percussion instrument as defined in [the section called "Percussion instrument names"](#). In the following example, the tone *c* is translated to the midi sound *Side Stick* and *d* to a *Mute triangle*.

```
variable = rhythm("d4 d d d c8 c8 c4")
```

rhythm

This music object let you write questions that taps rhythms with the two instruments defined in the preferences window. The tone *c* will play the rhythm representing the question, and the tone *d* can be used if you want to write some sort of "count-in" before the question starts. Example:

```
rhythm("d4 d d d c8 c8 c4 c c8 c8")
```

You should only use two pitches, *c* and *d*. Other pitches will print a warning, but will still work in the current implementation. To play real percussion with many different instruments you should use the [percussion](#) music object.

midifile

Play a midi file. The path given to the file is relative to the directory the lesson file is stored in. Example:

```
variable = midifile("share/example.mid")
```

wavfile

Play a .wav file. The path given to the file is relative to the directory the lesson file is stored in. Example:

```
variable = wavfile("share/fifth-small-220.00.wav")
```

mp3file

Play a MP3 file. Similar to wavfile.

oggfile

Play an Ogg Vorbis file. Similar to wavfile.

cmdline

Run an external program. Example:

```
cmdline("./bin/csound-play-harmonic-interval.sh 220.000000 320.100000")
```

Functions

–

`_` takes a string as its only argument. Use this if you want Solfege to translate the string for you. Example:

```
title = _("Bla bla title")
```

`include`

Includes another file. Example:

```
include("singchord-1")
```

The lesson header variables will be taken from the including lesson file. Only a variable is only defined in the included lesson file, and not in the including lesson file, then the value will be taken from the included file.

Operators

Operators can only be used on strings. `+` is used for joining strings, and `%` is similar to what you find in python, but it is very limited. It only know about `%s`. One example:

```
"\staff\relative c' {%s}" % "c d e"
```

evaluates to

```
\staff\relative c' {c d e}
```

The chord module

Warning

The chord module has been superseded by the [idproperty](#) module. This module will not be developed further, and will eventually be removed from the program. You should modify your lessons to use the [idproperty](#) module. The chord module will be removed from the solfege program in the next development series, in Solfege version 3.11.0.

It is easy to do the convert. In Solfege 3.10 the lesson file heading would contain the following line:

```
module = chord
```

Replace that with this

```
module = idproperty  
flavour = "chord"
```

The chord module let you identify different properties of chords, such as their name, inversion, top tone etc.

The properties are defined by the props variable in the lesson file header, and there should be a variable prop_labels that defines the label to use. props and prop_labels must be lists of equal length. You only have to define these two variables if you need other properties than the default ones: name, inversion and toptone.

Below is a minimal lesson file. It will create an exercise that will play a minor or major chord and the user answers with two buttons labeled "Minor" and "Major" and two buttons representing the inversion. Notice that unused properties, toptone in this example, are hidden.

```
header {
  module = chord
  title = "Minor and major chords"
  lesson_id = "e263d40a-d8ff-4000-a7f2-c02ba087bf72"
  qprops = "name", "inversion", "toptone"
  qprop_labels = _("Chord type"), _("Inversion"), _("Toptone")
}
question {
  name = "Major"
  music = chord("c' e' g'")
  inversion = 0
}
question {
  name = "Minor"
  music = chord("es' g' c'")
  inversion = 1
}
```

The inversion property is special. If assigned integer values, like in the example, the integer values will be replaced with strings. So 0 is replaced with "root position", 1 with "1. inversion" etc.

The compareintervals module

Here is a minimal lesson file:

```
header {
  countin_perc = compareintervals
  title = "Compare intervals"
  lesson_id = "9f830e12-1f50-4fa9-8688-1e04469692fa"
}
```

This file will make an exercise that ask you to compare harmonic intervals. And since you do not say which intervals, it will ask for all intervals from a small second up to a major decim.

first_interval_type, second_interval_type

Let you select if the intervals you are asked to compare should be a melodic or a harmonic interval. The default value is melodic. Possible values: harmonic and melodic.

```
first_interval_type = melodic
second_interval_type = harmonic
```

Modules: compareintervals.

first_interval, last_interval

Select which intervals to select from when creating the questions. This variable should be defined the same way as [ask_for_intervals_0](#). If these two variables are not defined, then the user will be able to select which intervals to practise from the Config page of the exercise.

Modules: compareintervals.

The dictation module

Example:

```
header {
  module = dictation
  lesson_id = "a265df62-e007-4a1b-9057-cd05397e88a2"
  title = _("Norwegian children songs")
  version = "2.1.10"
}

question {
  name = "Bæ, bæ, lille lam"
  tempo = 130/4
  breakpoints = 2/1, 4/1, 8/1, 10/1, 12/1, 14/1
  music = rvoice("""
    \time 4/4
    c'2 g' | e4 e c2 | d4 d g, g | c1 |
    c2 g' | e4 e c2 | d4 d g, g | c1 |
    a'4 f f f | g2. e4 | f d d d | e2. c4 |
    a'2 f | g e4 e | f b, b b | c1 |
  """)
}

question {
  # this tempo definition overrides the global
  tempo = 160/4
  name = "Lisa gikk til skolen"
  breakpoints = 2/1, 4/1, 6/1
  music = rvoice("""
    \time 4/4
    c' d e f | g2 g2 | a4 a a a | g1 |
    f4 f f f | e2 e | d4 d d d | c1
  """)
}

question {
  name = "Det satt to katter på et bord..."
  tempo = 96/4
  music = rvoice("""
    \key g \major \time 2/4
    d'8 | [g g] [fis e] | [fis g] a4 | [d,16 d d d] [e8 fis] | g2 """)
}
```

By default, the dictation exercise will show the first column of music, and then the user should write the rest. But if the first column is not good enough, for example if

there are only rests on the first beat, these two variables can tell the program how much music to display:

`clue_end`

The following example will display the music on all staves in the first quarter note:

```
clue_end=1/4
```

`clue_music`

This is an alternative to `clue_end`. The music assigned to `clue_music` will be shown to the user when he should start the dictation. You should not use both `clue_end` and `clue_music` in the same question.

`breakpoints`

Set breakpoints in the music, so you can hear the music in parts when doing the dictation.

The `elembuilder` module

Here is a minimal lesson file:

```
element progI { label = "I" }
element progIV { label = "IV" }
element progV { label = "V" }

header {
  lesson_id = "3f3872c0-ef2e-4132-9fb1-97f75c7b28fd"
  module = elembuilder
  title = "progression test"
  elements = auto
  # uncomment if you want a music displayer.
  # have_music_displayer = yes
}

question {
  music = rvoice("<c' e g> <b d g> <c e g>")
  elements = progI, progV, progI
  name = "I-V-I"
}

question {
  music = rvoice("<c' e g> <c f a> <c e g>")
  elements = progI, progIV, progI
  name = "I-IV-I"
}
```

The element block

This block defines the elements the user can put together to answer the question. Each block is named by the string between `element` and `{`. The block defines one variable, `label` that is the label the button will get.

`label` can either be a plain string, or a `progressionlabel`. `Progressionlabel` strings are displayed a little larger than the default font, and a simple syntax let you get small subscript and superscript numbers. Try `I-(6,4)V(6,4)-I` or `I-IV(6)-V(6)-I` to

get an idea how it works.

The header block

elements

This variable defines which elements to display. Set this to auto to display all elements that are needed to answer the questions in the lesson file. You can display more elements that needed to make it more difficult for the user. An example:

```
elements = progI, progIV, progV, progIV, progV_6
```

music_displayer_stafflines

Set this if you want the music displayer to show more than one empty staff line when the music displayer have no music to display.

See also [at_question_start](#).

The question block

elements

This variable defines which elements defines the question.

tonic

The exercise will have a "Play tonic" button if this variable is defined in a question in the lesson file. The variable should contain some music to play to the user so that he knows the tonic of the question. This can be useful in harmonic progressions that does not start on the tonic. This variable is optional. Example:

```
tonic = chord("c e g")
```

See also [vmusic](#).

The harmonicinterval module

User documentation is in [the section called "Harmonic interval"](#).

Here is a minimal lesson file:

```
header {  
  module = harmonicinterval  
  lesson\_id = "a400df62-e007-4a1b-9057-cd05397e88a2"  
  version = "3.1.4"  
  title = "Seconds"  
  intervals = [1, 2]  
  test = "3x"  
  test\_requirement = "90%"  
}
```

Additional variables you can put in the header. Click on the link to get an explanation:

- [disable_unused_intervals](#)
- [lesson_heading](#)

The idbyname module

Here is a minimal lesson file:

```
header {
  module = idbyname
  lesson\_id = "a400df62-e007-4a1b-9057-cd05397e88a2"
  version = "3.1.4"
  title = "Menuitem title"
}
question {
  name = "Major"
  music = chord("c' e' g'")
}
question {
  name = "Minor"
  music = chord("c' es' g'")
}
```

Optional idbyname header variables

`filldir = vertic`

Tell the direction the buttons are filled. Default value is `horiz`.

Modules: `idbyname`.

`fillnum`

Tell how many buttons there are in each row or column. The default value is `1`.

Modules: `idbyname`.

`labelformat = progression`

The default value is `normal`. Set to `progression` for lesson files where the name of the questions is a harmonic progression, written in a undocumented, but not difficult format. Check some existing lesson file to see how it works.

Modules: `idbyname`

`have_repeat_slowly_button = yes`

Set to `yes` if you want the exercise to have a "Repeat slowly" button.

Modules: `idbyname`.

See also [at_question_start](#).

Optional question variables

`vmusic`

See [vmusic](#).

cuemusic

Will be displayed in the music player when the user clicks New. Ignored if `at_question_start = play`, `show` or `at_question_start = show`, because then the content of `music` or `vmusic` is displayed when the user clicks New. (Added in Solfege 2.5.1)

The `idproperty` module

The `idproperty` module let you create exercises where solfege will play some music and you have to identify different properties of the music.

Below is a minimal lesson file. It will create an exercise that will play a minor or major chord and the user answers with two buttons labeled "Minor" and "Major" and two buttons representing the inversion. Notice that unused properties, `toptone` in this example, are hidden.

```
header {
  module = idproperty
  flavour = "chord"
  title = "Minor and major chords"
  lesson_id = "e263d40a-d8ff-4000-a7f2-c02ba087bf72"
}
question {
  name = "Major"
  music = chord("c' e' g'")
  inversion = 0
}
question {
  name = "Minor"
  music = chord("es' g' c'")
  inversion = 1
}
```

`flavour = "chord"` will add the following definitions to the lesson file header, unless if they are missing:

```
new_button_label = _("New chord")
lesson_heading = _("Identify the chord")
qprops = "name", "inversion", "toptone"
qprop_labels = _("Name"), _("Inversion"), _("Toptone")
```

`new_button_label` is the label to put on the **New** button. The default value is `_("New")`.

`lesson_heading` will set the heading to be displayed when you practise. The default value is an empty string, that will hide the heading.

The properties are defined by the `props` variable in the lesson file header, and there should be a variable `prop_labels` that defines the label to use. `props` and `prop_labels` must be lists of equal length.

The exercise will have a **Repeat arpeggio** button if one or more of the questions can be played arpeggiated. Set the lesson file header variable `have_repeat_arpeggio_button` to `no` to disable hide the button.

If the exercise have a inversion property, it will be treated special. If assigned integer values, like in the example, the integer values will be replaced with strings. So 0 is replaced with "root position", 1 with "1. inversion" etc.

The idtone module

Here is a minimal lesson file:

```
header {  
  module = idtone  
  title = "Id tone 3"  
  lesson\_id = "e263d70a-d8ff-4000-a7f2-c02ba087bf72"  
  black_keys_weight = 0, 0, 0, 0, 0  
  white_keys_weight = 1, 1, 1, 0, 0, 0, 0  
}
```

The 'weight' of a tone tell how big chance is it that the program will select this tone as the next to identify. Think of the weight of a tone as the number of lottery tickets with the name of the tone.

The variable `black_keys_weight` set the weight of the tones `c#`, `d#`, `f#`, `g#` and `a#`, and `white_keys_weight` will set the weight of the tones `c`, `d`, `e`, `f`, `g`, `a`, `b`. In the example above, the tones `c`, `d` and `e` get an equal weight of 1, the other tones 0. This mean that the only tones that will be asked for are `c`, `d` and `e`, and that the three tones share the same probability to be selected.

The melodicinterval module

User documentation is in [the section called "Melodic interval"](#).

Here is a minimal lesson file:

```
header {  
  module = melodicinterval  
  lesson\_id = "a400df62-e007-4a1b-9057-cd05397e88a2"  
  version = "3.1.4"  
  title = "Seconds and thirds"  
  ask\_for\_intervals\_0 = [1, 2, 3, 4, -1, -2, -3, -4]  
  test = "3x"  
  test\_requirement = "90%"  
}
```

Additional variables you can put in the header. Click on the link to get an explanation:

- [disable_unused_intervals](#)
- [lesson_heading](#)

Tests are only partially implemented for the `melodicinterval` exercise module: tests where each question is made by more than one interval does not work yet.

The nameinterval module

Here is a minimal lesson file:

```
header {
  lesson_id = "5623c43e-f529-4376-a0c9-c7d533050360"
  module = nameinterval
  title = _("Fifths")
  intervals = p5, a5, d5
}
```

intervals

A list the the intervals to ask for. The intervals are written in a short form, a letter and a number, like d5 or m7. The letters are telling the interval quality are 'd' for diminished, 'a' for augmented, 'm' for minor, 'M' for major and 'p' for perfect.

tones

This variable sets the range of tones that can be used when constructing the intervals. The note names as to be quoted. The default value is "b", "g' '". Example:

```
tones = "c'", "f'" # valid
tones = c', f'    # not valid
```

accidentals

This variable defines how many accidentals the tones making the interval can have. The value 0 means no accidentals, 1 means that flats and sharps are allowed, and 2 means that double flats and double sharps are allowed. The default value is 1. Example:

```
accidentals = 2
```

clef

Set which clef to use. The default value is violin. Possible values: violin, treble, subbass, bass, baritone, varbaritone, tenor, alto, mezzosoprano and french. Example:

```
clef = bass
```

The rhythm module

Here is a minimal lesson file:

```
header {
  module = rhythm
  lesson\_id = "7a4910be-de17-4ce3-9d15-78d48ccf945e"
  version = "3.1.4"
  title = "Easy rhythms"
  rhythm\_elements = 1, 2, 3, 4
}
```

visible_rhythm_elements

Define this variable if you want more rhythm elements that the one to be asked for. This variable must include both the rhythm elements defined in `rhythm_elements` and the extra elements. Example:

rhythm_elements = 0, 1, 2, 3, 4, 5, 6

countin_perc

An integer value between 35 and 81, representing the percussion instrument used to give you the beat before the question. The default value is 80. Example:

countin_perc = 35

35 Acoustic Bass Drum	51 Ride Cymbal 1	67 High Agoga
36 Bass Drum	52 Chinece Cymbal	68 Agogo Low
37 Side Stick	53 Ride Bell	69 Cabasa
38 Acoustic Snare	54 Tambourine	70 Maracas
39 Hand Clap	55 Splash Cymbal	71 Short Whistle
40 Electric Snare	56 Cowbell	72 Long Whistle
41 Low Floor Tom	57 Crash cymbal 2	73 Short Guiro
42 Closed Hi Hat	58 Vibraslap	74 Long Guiro
43 High Floor Tom	59 Ride Cymbal 2	75 Claves
44 Pedal Hi Hat	60 Hi Bongo	76 Hi Wood Block
45 Low Tom	61 Low Bongo	77 Low Wood Block
46 Open HiHat	62 Mute Hi Conga	78 Mute Cuica
47 Low-Mid Tom	63 Open High Conga	79 Open Cuica
48 Hi-Mid Tom	64 Low Conga	80 Mute Triangle
49 Crash Cymbal 1	65 High Timbale	81 Open Triangle
50 High Tom	66 Low Timbale	

Modules: rhythm

rhythm_perc

Same as [countin_perc](#), but setting the instrument used to play the question. The default value is 37.

Modules: rhythm

count_in

The number of beats as count in. The default value is 2.

Modules: rhythm

bpm

The tempo, in beats per minute. The default value is 60.

Modules: rhythm

num_beats

The number of elements the question is made of. The default value is 4.

Modules: rhythm

The rhythmtapping module

Exercises using this module will play some music and then the user should tap the rhythm. The program will then say if the users rhythm is similar enough to the rhythm

played by the computer.

Here is a minimal lesson file:

```
header {
  module = rhythmapping
  lesson\_id = "82b718e8-f174-446f-8297-58ddd17dae03"
  version = "3.7.0"
  title = "Rhythm tapping test"
}
question {
  music = rhythm("c4 c8 c8")
}
question {
  music = music("\staff\relative c'{c4 d8 e f4}\addvoice\relative c'{c4 b8 c a4}")
  rhythm = rhythm("c4 c8 c c4")
}
```

The first question in the example is very simple and self explaining. Solfege will play the rhythm defined in the music variable, and the user should tap that rhythm.

The second question is a little more complicated. Here Solfege will play the music defined in the music variable. And when the user taps the rhythm, Solfege will compare the users rhythm with the rhythm defined in the rhythm variable. The reason for using two variables is that Solfege is not smart enough to figure out the rhythm if you enter polyphonic music. It make noe difference if you set the rhythm variable to be a rhythm music object, or another single voice type like rvoice. This might change in the future. You as a lesson file author must make sure the rhythms in the two variables are in fact the same.

The rhythmapping2 module

Solfege will play a generated rhythm, and the user should tap the same rhythm.

Here is a minimal lesson file:

```
header {
  module = rhythmapping2
  lesson\_id = "7a4916be-de47-42e3-9d15-78d48ccf945e"
  version = "3.7.0"
  title = "Rhythm tapping test"
  rhythm\_elements = 1, 2, 3, 4
}
```

See also [at_question_start](#).

The singanswer module

Here is a minimal lesson file:

```
header {
  module = singanswer
  lesson\_id = "a400df62-e007-4a1b-9057-cd05397e88a2"
  version = "3.1.4"
  title = "Sing the root of the chord"
}
question {
```

```

    question_text = "Sing the root"
    music = chord("c' e' g'")
    answer = chord("c'")
}
question {
    question_text = "Sing the root"
    music = chord("a' c' e'")
    answer = chord("a'")
}

```

Additional variables you can put in the header. Click on the link to get an explanation:

- [have_repeat_arpeggio_button](#)

The singchord module

Questions for this exercise need to have the key variable set if the key signature is anything else than "c" major (or "a" minor). Example:

```

header {
    module = singchord
    lesson\_id = "a404df62-e037-6a1b-9027-cd05397e88a2"
    version = "3.1.4"
    title = "Simple chords"
}
question { music = satb("c'|e'|g|c") }
question { music = satb("a'|e'|c'|a") }
question { key="d \major" music = satb("a'|fis'|d'|d") }
question { key="f \minor" music = satb("as'|f'|c'|f") }

```

See also [the section called "Sing chord"](#).

The singinterval module

User documentation is in [the section called "Sing interval"](#).

Here is a minimal lesson file:

```

header {
    module = singinterval
    lesson\_id = "a400df62-e007-4a1b-9057-cd05397e88a2"
    version = "3.1.4"
    title = "Thirds"
    ask\_for\_intervals\_0 = [3, 4]
    test = "3x"
    test\_requirement = "90%"
}

```

Midi instrument names

acoustic grand	contrabass	lead 7 (fifths)
bright acoustic	tremolo strings	lead 8 (bass+lead)
electric grand	pizzicato strings	pad 1 (new age)
honky-tonk	orchestral strings	pad 2 (warm)
electric piano 1	timpani	pad 3 (polysynth)
electric piano 2	string ensemble 1	pad 4 (choir)
harpsichord	string ensemble 2	pad 5 (bowed)

clav	synthstrings 1	pad 6 (metallic)
celesta	synthstrings 2	pad 7 (halo)
glockenspiel	choir aahs	pad 8 (sweep)
music box	voice oohs	fx 1 (rain)
vibraphone	synth voice	fx 2 (soundtrack)
marimba	orchestra hit	fx 3 (crystal)
xylophone	trumpet	fx 4 (atmosphere)
tubular bells	trombone	fx 5 (brightness)
dulcimer	tuba	fx 6 (goblins)
drawbar organ	muted trumpet	fx 7 (echoes)
percussive organ	french horn	fx 8 (sci-fi)
rock organ	brass section	sitar
church organ	synthbrass 1	banjo
reed organ	synthbrass 2	shamisen
accordion	soprano sax	koto
harmonica	alto sax	kalimba
concertina	tenor sax	bagpipe
acoustic guitar (nylon)	baritone sax	fiddle
acoustic guitar (steel)	oboe	shanai
electric guitar (jazz)	english horn	tinkle bell
electric guitar (clean)	bassoon	agogo
electric guitar (muted)	clarinet	steel drums
overdriven guitar	piccolo	woodblock
distorted guitar	flute	taiko drum
guitar harmonics	recorder	melodic tom
acoustic bass	pan flute	synth drum
electric bass (finger)	blown bottle	reverse cymbal
electric bass (pick)	skakuhachi	guitar fret noise
fretless bass	whistle	breath noise
slap bass 1	ocarina	seashore
slap bass 2	lead 1 (square)	bird tweet
synth bass 1	lead 2 (sawtooth)	telephone ring
synth bass 2	lead 3 (calliope)	helicopter
violin	lead 4 (chiff)	applause
viola	lead 5 (charang)	gunshot
cello	lead 6 (voice)	

Percussion instrument names

The first column is the integer value for the instrument. The second column tell the name of the note you should enter in the [rhythm](#) music object.

35 b,,	Acoustic Bass Drum	59 b	Ride Cymbal 2
36 c,	Bass Drum 1	60 c'	Hi Bongo
37 cis,	Side Stick	61 cis'	Low Bongo
38 d,	Acoustic Snare	62 d'	Mute Hi Conga
39 dis,	Hand Clap	63 dis'	Open High Conga
40 e,	Electric Snare	64 e'	Low Conga
41 f,	Low Floor Tom	65 f'	High Timbale
42 fis,	Closed Hi Hat	66 fis'	Low Timbale
43 g,	High Floor Tom	67 g'	High Agogo
44 gis,	Pedal Hi Hat	68 gis'	Agogo Low
45 a,	Low Tom	69 a'	Cabasa
46 ais,	Open HiHat	70 ais'	Maracas
47 b,	Low-Mid Tom	71 b'	Short Whistle
48 c	Hi-Mid Tom	72 c''	Long Whistle
49 cis	Crash Cymbal 1	73 cis''	Short Guiro
50 d	High Tom	74 d''	Long Guiro
51 dis	Ride Cymbal 1	75 dis''	Claves
52 e	Chinese Cymbal	76 e''	Hi Wood Block
53 f	Ride Bell	77 f''	Low Wood Block
54 fis	Tambourine	78 fis''	Mute Cuica
55 g	Splash Cymbal	79 g''	Open Cuica
56 gis	Cowbell	80 gis''	Mute Triangle

57 a Crash Cymbal 2 81 a'' Open Triangle
58 ais Vibraslap

GNU General Public License

Version 2, June 1991

Copyright © 1989, 1991 Free Software Foundation, Inc.

Free Software Foundation, Inc.
51 Franklin Street, Fifth Floor,
Boston,
MA
02110-1301
USA

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Version 2, June 1991

Table of Contents

[Preamble](#)

[TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION](#)

[Section 0](#)

[Section 1](#)

[Section 2](#)

[Section 3](#)

[Section 4](#)

[Section 5](#)

[Section 6](#)

[Section 7](#)

[Section 8](#)

[Section 9](#)

[Section 10](#)

[NO WARRANTY Section 11](#)

[Section 12](#)

[How to Apply These Terms to Your New Programs](#)

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software - to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive

source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps:

1. copyright the software, and
2. offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

Section 0

This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program " means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification ".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

Section 1

You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

Section 2

You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of [Section 1](#) above, provided that you also meet all of these conditions:

1. You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.
2. You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.
3. If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License.

Exception:

If the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or

distribution medium does not bring the other work under the scope of this License.

Section 3

You may copy and distribute the Program (or a work based on it, under [Section 2](#) in object code or executable form under the terms of [Sections 1](#) and [2](#) above provided that you also do one of the following:

1. Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
2. Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,
3. Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interfacedefinition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

Section 4

You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Section 5

You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on

the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

Section 6

Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

Section 7

If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

Section 8

If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

Section 9

The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

Section 10

If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY Section 11

BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

Section 12

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to

the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.> Copyright (C)
<year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301, USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'. This is free software, and you are welcome to redistribute it under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than `show w' and `show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program `Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989 Ty Coon, President of Vice

This General Public License does not permit incorporating your program into proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.

Appendix B. Not really documentation...

Table of Contents

[Not really documentation...](#)
[Welcome to GNU Solfege](#)

Not really documentation...

This appendix contain the file that is displayed when the program starts, before the user selects an exercise.

Welcome to GNU Solfege

Solfege is a [free](#) eartraining program. The program is part of the [GNU Project](#). Check [the section called "Online resources"](#) for info on mailinglists and where to get the latest version of Solfege.

Select a exercise from the menu to start practising, or click [here](#) to read the user manual.

One of the ideas of this program is that you can extend the program without having to dig into the source code. If you want to practise some special chords or want to practise dictation with some music not included, you can write lesson files and put them into a `lessonfiles/` subdirectory in your `$HOME` directory. If you create good lesson files, you really should consider contributing them by sending them to the mailinglist so I can add them to the next version of this program.