GNU Solfege 3.16.bzr-checkout User Manual

Tom Cato Amundsen

<tca@gnu.org>

Opphavsrett © 2005 Tom Eykens

Opphavsrett © 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 3 of the License, or (at your option) any later version. The full text of the License is available in Tillegg A, *GNU General Public License version 3* .

Innholdsfortegnelse

1. Introduksjon
- Velkommen til GNU Solfege
- Bugs
- Online resources
  - Download Solfege
  - Mailing lists
- Preferences window
  - Instruments
  - User
  - External programs
  - Interface
  - Øv
  - Sound setup
- Training set editor
- Ear training test print-out editor

2. Hjelp for hver enkelt øvelse
- Harmonic interval
  - Konfigurasjon
  - Tastatursnarveier
- Melodisk intervall
  - Konfigurasjon
  - Tastatursnarveier
- Syng intervall
  - Konfigurasjon
  - Tastatursnarveier
- Identify the chord
  - Tastatursnarveier
- Identify the cadence
- Multiple choice answers to music
  - Tastatursnarveier
- Syng akkord
  - Tastatursnarveier
- Rytme

- Tastatursnarveier
- Tap the rhythm
- Dictation
  - Tastatursnarveier
- Skalaer
  - Tastatursnarveier
- Intonation
  - Tastatursnarveier
- Identify tone
  - Manual configuration
- Beats per minute
  - Tastatursnarveier
- Sing 12 random notes
  - Tastatursnarveier
- Name intervals
- Harmonic progression dictation

3. Music theory
- Skalaer
- Intervals
  - Seconds
  - Thirds
  - Fourth
  - Fifth
  - Sixths
  - Sevenths
- Inverting intervals

4. Extending GNU Solfege
- Introduction
- Lesson files
  - File encoding
  - Useful unicode characters
  - Comments
  - Types
  - Global variables
  - Lesson file contents
  - Header block
  - Question block
  - `music` objects
  - Functions
  - Operators
- The `chordvoicing` 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
  - Question block
- The `identifybpm` 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
The twelvetone module
The mpd module
Midi instrument names
Percussion instrument names
A. GNU General Public License version 3

Figuroversikt

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

Kapittel 1. Introduksjon

Innholdsfortegnelse

Velkommen til GNU Solfege
Bugs
Online resources
Download Solfege
Mailing lists
Preferences window
Instruments
User
External programs
Interface
Øv
Sound setup
Training set editor
Ear training test print-out editor

Velkommen til GNU Solfege

Solfege er et gehørstreningsprogram. Det er fri programvare og er en del av GNU-prosjektet. Les "Online resources" for å finne mer informasjon om epostlister og hvor du kan skaffe deg den siste versjonen av programmet.

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.

Bugs

Report bugs to the bug tracker at http://bugs.solfege.org. Alternately, you can send an email to <bug-solfege@gnu.org>. 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

Instruments

Tempo

Set the tempo (beats per minute) for music and arpeggios. These values are used by exercises written with these exercise modules: `compareintervals`, `harmonicintervals`, `idtone`, `melodicinterval`, `singchord`, `singinterval`, `twelvetone`, `rhythm`, `identifybpm`, `nameinterval`. Other exercises will either have the tempo set in the lesson file or on the config page of the exercise.

Preferred instrument

Set the midi instrument and volume used for most exercises.

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

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 minor and major tenths, 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

Solfege will search the PATH for the programs you enter on this page. So you only have to enter the full path if the programs are installed outside the PATH.

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 extention.

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`.

Miscellaneous

A few exercises make use of the programs CSound and MMA. Lilypond-book is required to make ear training test printouts, and latex is required if the printout should be created in dvi format. Without latex, you can still create html output.

If the file entered is a file ending with `.py`, then the script will be run by the same python interpreter as Solfege itself.

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

Interface

Display user manual in web browser: Use this as an alternative to the help browser included with the program.

Resizeable main window: Allow the user to resize the main solfege window.

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 that your operating system.

Øv

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: Enabling this option in exercises using the idbyname and idproperty modules will let you select to practise only a subset of the questions in the lesson file. Practising with expert mode enabled will not save any statistics.

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
>
> On MS Windows this choise is labeled `Windows multimedia output`.

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.

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 traing 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.

Kapittel 2. Hjelp for hver enkelt øvelse

Innholdsfortegnelse

Harmonic interval
Konfigurasjon

Tastatursnarveier
Melodisk intervall
Konfigurasjon
Tastatursnarveier
Syng intervall
Konfigurasjon
Tastatursnarveier
Identify the chord
Tastatursnarveier
Identify the cadence
Multiple choice answers to music
Tastatursnarveier
Syng akkord
Tastatursnarveier
Rytme
Tastatursnarveier
Tap the rhythm
Dictation
Tastatursnarveier
Skalaer
Tastatursnarveier
Intonation
Tastatursnarveier
Identify tone
Manual configuration
Beats per minute
Tastatursnarveier
Sing 12 random notes
Tastatursnarveier
Name intervals
Harmonic progression dictation

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.

Hvis du bruker knappe-grensesnittet, så kan du høyreklikke på knappene for å høre det intervallet de representerer.

Konfigurasjon

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.

Tastatursnarveier

- Nytt interval: **Alt**+**n**
- Repeter: **Alt**+**r**
- Repeat melodic: **Alt**+**m**
- Gi opp: **Alt**+**g**

Liten sekund: 1	Ren kvart: 2	Stor sekst: 3	Liten none: 4
Stor sekund: q	Tritonus: w	Liten septim: e	Stor none: r
Liten ters: a	Ren kvart: s	Stor septim: d	Minor tenth: f
Stor ters: z	Liten sekst: x	Ren oktav: c	Major tenth: v

Melodisk intervall

Denne øvelsen lager tilfeldige intervaller, og du skal forsøke å identifisere dem.

Hvis du bruker knappe-grensesnittet, så kan du høyreklikke på knappene for å høre det intervallet de representerer.

Konfigurasjon

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.

Tastatursnarveier

- New question: **Alt**+**n**
- Repeter: **Alt**+**r**
- Gi opp: **Alt**+**g**

Liten sekund: 1	Ren kvart: 2	Stor sekst: 3	Liten none: 4
Stor sekund: q	Tritonus: w	Liten septim: e	Stor none: r

Liten ters: a	Ren kvart: s	Stor septim: d	Minor tenth: f
Stor ters: z	Liten sekst: x	Ren oktav: c	Major tenth: v

Syng intervall

I denne vil Solfege vise notene til et eller flere intervaller, og du skal synge dem. Dessverre er det ikke mulig å synge svaret i en mikrofon og la programmet av gjøre om du synger riktig. Dette må du gjøre selv.

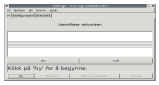
Konfigurasjon

Programmet vil prøve å lage spørsmål der alle tonene er innenfor det omfanget brukeren kan synge. Omfanget kan settes i vinduet brukervalg. Noen ganger er det ikke mulig å holde spørsmålet innenfor dette området, for eksempel hvis øvelsen er satt opp til å lage mange intervaller, og alle intervallene må gå oppover.

Tastatursnarveier

- Nytt interval: **Alt**+**n**
- Nytt interval, det siste var korrekt: **Alt**+**n**
- Nytt intervall, det siste var feil: **Alt**+**e**
- Repeter første tone: **Alt**+**r**
- Spill svar: **Alt**+**s**
- Spill siste tone: **Alt**+**i**

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.

Tastatursnarveier

- New chord: **Alt**+**n**
- Repeter: **Alt**+**r**
- Repeat arpeggio: **Alt**+**a**
- Gi opp: **Alt**+**g**

Identify the cadence

Identify the cadence by clicking on the button with its name. As acknowledged in bug #5 we need cadence exercises in Solfege.

In this release of Solfege, we only have one exercise with cadences in major. In this exercise, a major scale is played to establish the tonic. Maybe this is too little? Do we need a complete I-IV-V-I before the question? Or maybe it is better to write real music that ends in the cadence we want to practise? This are things we need to descide before we release 3.12.0. Comments and music can be added to bug #5.

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.

Tastatursnarveier

- New chord: **Alt**+**n**
- Repeter: **Alt**+**r**
- Repeat arpeggio: **Alt**+**a**
- Gi opp: **Alt**+**g**

Syng akkord

Som kordirigent må man ofte synge første tone til alle stemmene i koret. Hvis du er mann, så må du synge damestemmene en oktav under det de skal klinge. Og omvendt hvis du er kvinne.

Programmet spiller en enstrøken A (440 hz) og viser en akkord som du skal synge. Siden programmet ennå ikke støtter bruk av mikrofon, så må du selv avgjøre om det du synger er korrekt.

Tastatursnarveier

- Ny: **Alt**+**n**
- 440 hz: **Alt**+**z**
- Spill svar: **Alt**+**s**

Rytme

Programmet spiller en tilfeldig rytme, og brukeren skal reprodusere den. Brukeren skriver inn rytmen ved å klikke på knappene som representerer de forskjellige rytmiske figurene.

Programmet sjekker svaret ditt når du har skrevet inn mange nok rytmefigurer. Et gladfjes vil vises hvis alt er riktig, og hvis noe er feil, vil de feile rytmene bli markert samtidig som et trist fjes vises.

Hvis deler av svaret ditt er feil, så vil alt fra og med den første feile rytmen bli fjernet (eventuelle riktige rytmer i begynnelsen av svaret blir bevart) når du klikker på det triste fjeset eller når du klikker på en ny rytme øvers på siden.

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

Spørsmålene til denne øvelsen blir laget ved å velge rytmiske figurer helt tilfeldig. Dette er nok ikke den beste måten å gjøre det på, og vi håper å finne en bedre måte å gjøre dette på i en senere utgave av programmet.

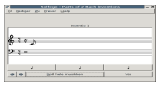
Tastatursnarveier

- Ny: **Alt**+**n**
- Repeter: **Alt**+**r**
- Gi opp: **Alt**+**g**
- Rettestast: **Rettetast**

Tap the rhythm

Programmet spiller en tilfeldig rytme, og brukeren skal reprodusere den. Brukeren gir sitt svar ved å klikke inn rytmen på kjappen *Bank her*.

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 Solfege 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 mistaces.

- 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 succeded.

Tastatursnarveier

- Play the whole music: **Alt**+**p**

- Show: **Alt**+**s**

Skalaer

Skalaer er kompliserte greier. For eksempel er den greske lydiske skalaen (C-D-E-F-G-A-H-C) forskjellig fra kirketonearten med samme navn (C-D-E-Fiss-G-A-H-C). Den følgende tabellen beskriver de skalaene som er i bruk i denne versjonen av GNU Solfege.

Solfege har tre forskjellige skalaøvelser.

- Solfege spiller en skala, og du skal identifisere skalaen ved å klikke på knappen med riktig navn.

- 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 idenitfy 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.

Tastatursnarveier

- Ny: **Alt**+**n**

- Repeter: **Alt**+**r**
- Repeter langsomt: **Alt**+**l**
- Gi opp: **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.

Tastatursnarveier

- Ny: **Alt**+**n**
- Repeter: **Alt**+**r**
- Gi opp: **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 this exercise as you like if you select select “Configure yourself” front the default front page.

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. Exercises are defined that will add one and one tone until you practise with all 12 tones.

Heavy A

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.

Konfigurasjon

If run from "Configure yourself", 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 the preferences window.

Beats per minute

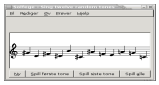
The program will play a tempo, like a methronome. You should try to guess how many beats per minute is played. Each button represents ony tempo, and the program will only play in tempos that has a button with bold text. Right-click on buttons to change the status of a tempo.

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

Tastatursnarveier

- New tempo: **Alt**+**n**
- Gi opp: **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 interal exercises.

Tastatursnarveier

- Ny: **Alt**+**n**
- Spill første tone: **Alt**+**i**
- Spill siste tone: **Alt**+**s**
- Spill alle: **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 "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.

Kapittel 3. Music theory

Innholdsfortegnelse

Skalaer
Intervals
    Seconds
    Thirds
    Fourth
    Fifth
    Sixths
    Sevenths
Inverting intervals

Skalaer

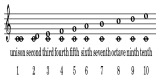
Davide Bonetti has contributed a large set of scale exercises and some pages describing all the scales. You can see the pages here.

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 simultaneosly and melodic intervals if they sound successively.

Interval names consist of two parts. Some examples are "major third" and "perfect fifth". In Walter Pistons "Harmony" the two parts are called *the specific name* and *the general name* part. Wikipedia talk about *interval quality* and *interval number*. I have seen people talk about an intervals *numerical size*.

You find the general name by counting the steps on the staff, ignoring any accidentals. So if the inteval 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 say 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 a augmented interval is one semitone larger than a pure or major interval.

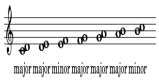
Accidentals change the size of intervals. The interval becomes one semitone larger if you add a sharp to the highest tone or a flat to the lowest tone. And it becomes one semitone smaller if you add a flat to the highest tone or a sharp to the lowest tone. In the following sections naming of the intervals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

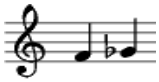
To learn to identify seconds, you first have to learn which seconds there are between the natural tones. As you can see in Figur 3.1, “”, only the intervals E-F and B-C are minor seconds. The rest are major intervals. You can check that Figur 3.1, “” is correct by looking at a piano. You will see that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

Figur 3.1.



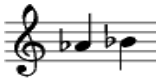
If the second has accidentals, then we have to examine them to find out how they change the size of the interval. Let us identify a few intervals!

Figur 3.2.



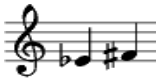
We remove the accidental from the interval in Figur 3.2, “” and see that the interval F-G is a major second. When we add the flat to the highest tone, the interval becomes one semitone smaller, and becomes a minor second.

Figur 3.3.



We remove the accidentals, and see that the interval A-B is a major second. You still do remember Figur 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.

Figur 3.4.

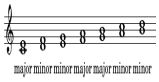


We remove the accidentals, and see that the interval E-F is a minor second. When we add a flat to the lowest tone, the interval becomes one semitone larger, and becomes a major second. And when we add a sharp to the highest tone, the interval becomes one semitone larger, and becomes an augmented second.

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. Figur 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.

Figur 3.5.

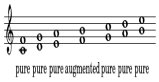


Then you examine the accidentals to see if they change the specific name. This is done exactly the same way as for seconds.

Fourth

A pure fourth is 2½ steps, or two major seconds and a minor second. Figur 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.

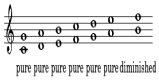
Figur 3.6.



Fifth

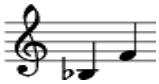
A pure fifth is 3½ steps, or three major seconds and a minor second. Figur 3.7, “” show all fifths between natural tones. You should remember that all those intervals are pure, except B-F that is diminished.

Figur 3.7.



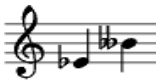
If the interval has accidentals, then we must examine them to see how they change the size of the interval. A diminished fifth is one semitone smaller than a pure interval, and a augmented fifth is one semitone larger. Below you will find a few examples:

Figur 3.8.



We remember from Figur 3.7, “” that the interval B-F is a diminished fifth. The lowest tone in Figur 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.

Figur 3.9.



We know from Figur 3.7, “” that interval E-B is a perfect fifth. In Figur 3.9, “” the E has a flat in front of it, making the interval augmented. But then the B is preceded by a doble flat that makes the interval two semitone steps smaller and changes the interval to a diminished fifth.

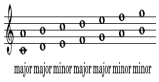
Sixths

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.

Figur 3.10.

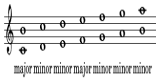


Sevenths

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.

Figur 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:

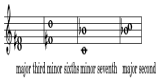
- Second becomes seventh.
- Third becomes sixth.
- Forth becomes fifth.
- Fifth becomes fourth.
- Sixth becomes third.
- Seventh becomes second.

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.

Figur 3.12.



Kapittel 4. Extending GNU Solfege

Innholdsfortegnelse

Introduction
Lesson files
- File encoding
- Useful unicode characters
- Comments
- Types
- Global variables
- Lesson file contents
- Header block
- Question block
- `music` objects
- Functions
- Operators

The `chordvoicing` 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
- Question block

The `identifybpm` 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
The `twelvetone` module
The `mpd` 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 and save it it in the first directory listed when you select **User Exercises** from

the **File** menu. Create the directory if it does not exist.

- Select **User Exercises** once again to see the file show up in the list.
- Click the link to your lesson file and enjoy!

To get started, you can copy one of the lesson files included in GNU Solfege. The lesson files are located in the `exercises/standard/lesson-files` subdirectory of the installation directory. You can find the installation directory by selecting **File Locations** from the **Help** menu. 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 because all files in the installation directory may be removed while upgrading the program.

If you create many lesson files, you might want to group them together in a separate subdirectory and attach them to a front page file. This way you have a set of files in a subdirectory that you can easily distribute to other students. So create a new directory side by side the `user` directory you found by selecting **User Exercises** earlier this article. Your files might be structured like this:

```
myfiles/myfrontpage.txt
myfiles/lesson-files/chords-1
myfiles/lesson-files/chords-2
```

To create a new front page file, you should select **Edit Front Page** from the **File** menu, and then click **New** on the toolbar of the dialog that pops up.

Lesson files

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

Deprecated modules: chord, harmonicprogressiondictation,

Missing documentation: chordvoicing, identifybpm, twelvetone

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.

Useful unicode characters

Unicode has some characters that you might want to use to make labels look more professionally. If your editor use unicode by default, you may copy-and-paste the characters you need from here, if you are viewing this documentation in a web browser. The number is a hexidecimal number.

ø `00F8 LATIN SMALL LETTER O WITH STROKE`

> Half-diminished seventh chord.

° `00B0 DEGREE SIGN`

> Diminished seventh chord.

△ `25B3 WHITE UP-POINTING TRIANGLE`, Δ `0394 GREEK CAPITAL LETTER DELTA`

> Major seventh chord. We do not know which character to recommend. Solfege does not care, so you can use the symbol you like.

♭ `266D MUSIC FLAT SIGN`

> This sign can be used instead of the letter 'b' for a flat sign.

♯ `266F MUSIC SHARP SIGN`

> This can be used instead of the letter '#' for the sharp sign.

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 ';lt;' 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 "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/beatlen`. 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. And there is a limitation that the header has to be within the first 40000 characters of the file.

Variables shared by many exercise modules

module

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

```
module = idbyname
```

replaces

A string or list of strings with hash values of lesson files that this lesson file can replace without dropping the statistics. Use this only when you know what you are doing. The hash value is calculated by `solfege.lessonfile.hash_of_lessonfile()`. **tools/hash-of-file.py** can be used to get the hash value of files before modifying them.

```
replaces = "bf7dd374206451bff43d61fc8191f5fb3e88d007"
replaces = "bf7dd374206451bff43d61fc8191f5fb3e88d007",
"cdb2f9415171650ee7682028788c1c42c62fdbf"
```

lesson_id

This variable is deprecated in Solfege 3.15.3. It should remain in existing lesson files for some time for backward compatability. But it should not be added to new lesson files.

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 o sharps (#). Zero mean no accidentals. The integers INTEGER1 and INTEGER2 defines a range of allowed number of accidentals.

For this to work properly the music must either be in C major or A minor, or the key of the music has to be set with the `key` variable.

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.

For this to work properly the music must either be in C major or A minor, or the key of the music has to be set with the `key` variable.

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.

random_transpose = atonal, INTEGER1, INTEGER2

Transpose the music at most INTEGER1 semitones down or INTEGER2 semitones up. Similar to `semitones`, but the notes will be tranposed one by one, and the key will not change.

`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 that 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 lesson file 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`.

`music_displayer_stafflines = INTEGER`

The number of empty staff lines to display when we have no music to display.

Modules: `idbyname` and `elembuilder`.

`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`

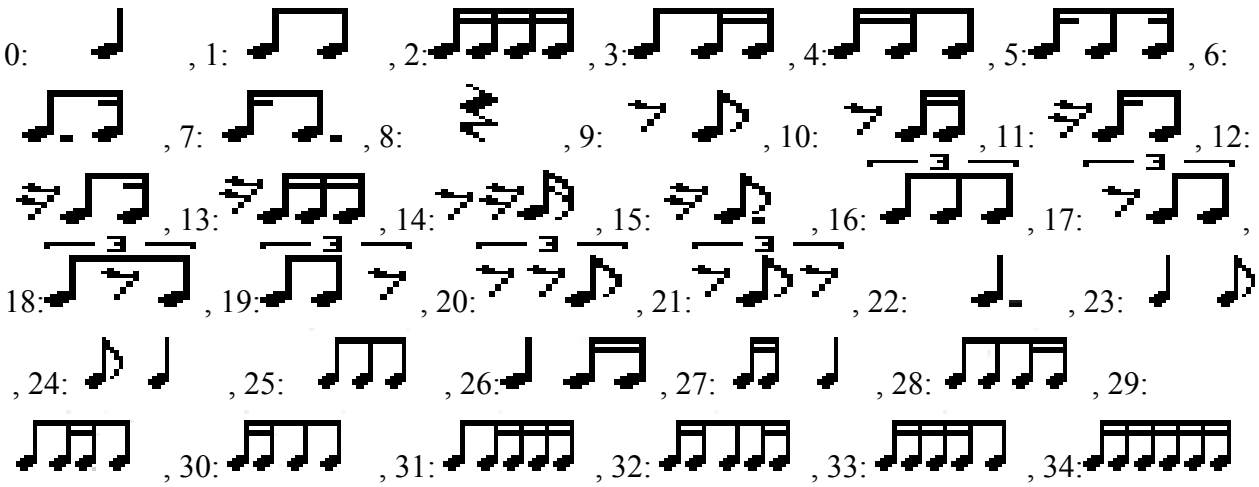
This variable holds a representation of the question intended to be displayed. This can be necessary if the music is a .wav or .mp3 file. It will be used when the user clicks Show music or when the question is answered correctly (if we have a musicdisplayer). Added to `idbyname` in Solfege 2.5.1 and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



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 elembuilder exercise modules need a name. A name is optional for dictation module.

`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 "`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 volume, and it should be in the range 0-127. You can see a list of instrument names in "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`.

`key`

Needed to make some random transposition work properly if the music is not in C major on A minor. Two examples:

```
key="b \minor"
key="g \major"
```

`music` objects

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

music(*musiccode*)

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' }")
```

music3(*musiccode*)

The music object can be used for music that has 3 or more staffs. It works the same way as music, but if "Use different instruments for chords and harmonic intervals" is checked in the preferences window, the 3 instruments you can select the same place will be used instead of the preferred MIDI instrument.

chord(*musiccode*)

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

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

satb(*musiccode*)

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(*musiccode*)

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(*musiccode*)

`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(*percussioncode*)

This music object provides a simple way to play rhythms with percussion instruments. Each tone represents a percussion instrument as defined in "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(*musiccode*)

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(*filename*)

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(*filename*)

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(*filename*)

Play a MP3 file. Similar to `wavfile`.

oggfile(*filename*)

Play an Ogg Vorbis file. Similar to `wavfile`.

csound(*orchestra*, *score*)

Given a CSound orchestra and score, this music object will generate a WAV file and play it. Example:

```
csound(load("share/sinus.orc"), """
    f1 0 4096 10 1
    i1 0 1 220.0
    i1 + 1 329.04
    """)
```

mma(*mmacode*), **mma**(*groove, mmacode*)

Create a music object that use MMA to generate music that it will play. If you create the object with one argument, *mmacode* should be a string with complete MMA code. With two arguments, `groove` is a string with the name of the groove, and `mmacode` is comple MMA code, except it could be missing the initial "Groove" instruction. The groove from `groove` will be prepended the string.

cmdline(*shell code*)

Run an external program. Example:

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

Functions

- **`_(`*`message`*`)`**

  Return the translation of *`message`* if it exist. Return the string unchanged if not.

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

- **`include(`*`filename`*`)`**

  Read the file *`filename`* into the lesson file and parse it as a part of the file. The filename is relative to the location of the lesson file.

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

  The lesson header variables will be taken from the including lesson file. Only if 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.

- **`load(`*`filename`*`)`**

  Read the file *`filename`* from disk and return it as a string. The filename is relative to the location of the lesson file.

  ```
  orc = load("share/sinus.orc")
  ```

Label functions

We call these functions *label functions* because we use them to create the label for some questions in the program. You should only use these functions where they are documented to work.

- **`pangomarkup(`*`pangostring`*`)`**

Return a label that the program can put on a button. The label is created using GTK pangomarkup. Google for "pango markup" to get the markup explained. Notice that you have to use triple qoutes around the string.

```
pangomarkup("""<span size="xx-large">V</span>""")
```

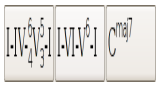
- **`progressionlabel(`*`str`*`)`**

This function has existed in Solfege for a while, but it has not been documented until now. Should we find a shorter function name? An alias can be added so that the old long function name still works.

Return a label. *`str`* is interpreted like this:

- Each letter outside of a parentheses is displayed with a large serif font.
- The text inside parentheses is displayed as a superscript: smaller letters above the baseline.
- If the text inside the parentheses is divided by a comma, the text before the comma is superscript and after the comma is subscript.

```
progressionlabel("I-IV-(6,4)V(5,3)-I")
progressionlabel("I-VI-V(6)-I")"
progressionlabel("C(maj7)")
```



- **`rnc(`*`str`*`)`**

Display a sequence of roman numeral chords. The chords are separated by whitespace and an optional hyphen. The exact implementation of this is still open for discussion. The current developent version of Solfege will divide each chord in 3 parts and give them different font sizes, and also try to make the chord compact, so that it should not take too much space on screen.

- The first part of the chord is the roman numberal, including an optional `b` or ♭ (unicode character `U+266D MUSIC FLAT SIGN`).
- The second part is the letters (if any) between the first and the third part.
- The third part is from the first digit and the rest of the chord.

```
rnc("Imaj7-IIm7-V9-Imaj7")
```

Spaces are not allowed in the chord name.

New in version 3.11.0.

- **`chordname(`*`str`*`)`**

Display a sequence of chords. The chords are separated by whitespace. Each chord consist of up to four parts, and part two to four are optional:

```
[notename][txt1][:txt2][/bass]
```

`notename` and `bass` music be a notename in the format understood by the music parser. You can read more about this in "The `mpd` module". Example:

```
g:11b9 cm/g ges:Δ besm:7/f
```

New in version 3.11.1.

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 `chordvoicing` module

Still undocumented...

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 tenth.

`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 staffs 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 one of the label functions.

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 and music_displayer_stafflines.

The question block

`elements`

This variable defines which elements defines the question. It can be elements, as defined in the example above, or strings or labels defined by the label functions.

`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")
```

`name`

The name is needed for storing statistics. A string or a label created by the label functions.

See also vmusic.

The `harmonicinterval` module

User documentation is in "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

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.

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.

> **Advarsel**
>
> Using this variable is deprecated. Do not use it for new lesson files.

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 and music_displayer_stafflines.

Question block

Required question variables

- name. Can be a string or a label created by the label functions.

- music

Optional question variables

`vmusic`

See vmusic.

`cuemusic`

Will be displayed in the music displayer 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 `identifybpm` module

Still undocumented...

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 emptry 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 "Melodisk intervall".

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

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

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 = rhythmtapping
  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 enought 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 `rhythmtapping2` module

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

Here is a minimal lesson file:

```
header {
  module = rhythmtapping2
  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 "Syng akkord".

The `singinterval` module

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.

User documentation is in "Syng intervall".

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%"
}
```

The `twelvetone` module

Still undocumented...

The mpd module

The module is not documented yet. The input format is similar to the one used by GNU Lilypond, but only the simplest construct works.

Quick note: Notenames understood by the program are c, d, e, f, g, a, b, with 'is', 'isis', 'es', or 'eses' added. For example 'fis', 'bes', 'gisis'.

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
```

Tillegg A. GNU General Public License version 3

Version 3, 29 June 2007

Copyright © 2007 Free Software Foundation, Inc. http://fsf.org/

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

Preamble

The GNU General Public License is a free, copyleft license for software and other kinds of works.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change all versions of a program—to make sure it remains free software for all its users. We, the Free Software Foundation, use the GNU General Public License for most of our software; it applies also to any other work released this way by its authors. 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 them 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 prevent others from denying you these rights or asking you to surrender the rights. Therefore, you have certain responsibilities if you distribute copies of the software, or if you modify it: responsibilities to respect the freedom of others.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must pass on to the recipients the same freedoms that you received. 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.

Developers that use the GNU GPL protect your rights with two steps: (1) assert copyright on the

software, and (2) offer you this License giving you legal permission to copy, distribute and/or modify it.

For the developers' and authors' protection, the GPL clearly explains that there is no warranty for this free software. For both users' and authors' sake, the GPL requires that modified versions be marked as changed, so that their problems will not be attributed erroneously to authors of previous versions.

Some devices are designed to deny users access to install or run modified versions of the software inside them, although the manufacturer can do so. This is fundamentally incompatible with the aim of protecting users' freedom to change the software. The systematic pattern of such abuse occurs in the area of products for individuals to use, which is precisely where it is most unacceptable. Therefore, we have designed this version of the GPL to prohibit the practice for those products. If such problems arise substantially in other domains, we stand ready to extend this provision to those domains in future versions of the GPL, as needed to protect the freedom of users.

Finally, every program is threatened constantly by software patents. States should not allow patents to restrict development and use of software on general-purpose computers, but in those that do, we wish to avoid the special danger that patents applied to a free program could make it effectively proprietary. To prevent this, the GPL assures that patents cannot be used to render the program non-free.

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

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The “source code” for a work means the preferred form of the work for making modifications to it. “Object code” means any non-source form of a work.

A “Standard Interface” means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming language, one that is widely used among developers working in that language.

The “System Libraries” of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A “Major Component”, in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The “Corresponding Source” for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work’s System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users’ Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law

fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey 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; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a. The work must carry prominent notices stating that you modified it, and giving a relevant date.

b. The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c. You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d. If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of

these ways:

a. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b. Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c. Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d. Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e. Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A “User Product” is either (1) a “consumer product”, which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, “normally used” refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

“Installation Information” for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

“Additional permissions” are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a. Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b. Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c. Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d. Limiting the use for publicity purposes of names of licensors or authors of the material; or

e. Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f. Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered “further restrictions” within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An “entity transaction” is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party’s predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A “contributor” is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor’s “contributor version”.

A contributor’s “essential patent claims” are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, “control” includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor’s essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a “patent license” is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To “grant” such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. “Knowingly relying” means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient’s use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is “discriminatory” if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others’ Freedom.

If 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

convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Use with the GNU Affero General Public License.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU Affero General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the special requirements of the GNU Affero General Public License, section 13, concerning interaction through a network will apply to the combination as such.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU 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 that a certain numbered version of the GNU General Public License “or any later version” applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU General Public License can be used, that proxy’s public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

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.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS 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.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

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 state 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 3 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, see http://www.gnu.org/licenses/.
```

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

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

```
program Copyright (C) year name of author
This program 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, your program's commands might be different; for a GUI interface, you would

use an “about box”.

You should also get your employer (if you work as a programmer) or school, if any, to sign a “copyright disclaimer” for the program, if necessary. For more information on this, and how to apply and follow the GNU GPL, see http://www.gnu.org/licenses/.

The GNU 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 Lesser General Public License instead of this License. But first, please read http://www.gnu.org/philosophy/why-not-lgpl.html.