

# Denemo User Manual

Richard Shann

[richard@rshann.plus.com](mailto:richard@rshann.plus.com)

This Manual is released under the Creative Commons Attribution-Share Alike 3.0 Unported license.

Copyright © 2009, 2010, 2011, 2012, 2013, 2014, 2015 Denemo Project

Updates for version 1.3

# Contents

<b>I</b>	<b>Getting Started</b>	<b>1</b>
<b>1</b>	<b>Input Methods</b>	<b>1</b>
1.1	Some Common Keyboard Shortcuts . . . . .	1
1.2	Some Common Mouse Shortcuts . . . . .	2
1.3	Some Common Uses of MIDI in . . . . .	2
1.4	Note and Rest Entry Palettes . . . . .	2
<b>II</b>	<b>Using Denemo</b>	<b>3</b>
<b>2</b>	<b>General Concepts</b>	<b>3</b>
<b>3</b>	<b>Denemo Main Window</b>	<b>3</b>
3.1	The Main Menubar . . . . .	3
3.1.1	File . . . . .	5
3.1.2	Edit . . . . .	5
3.1.3	View . . . . .	5
3.1.4	Input . . . . .	6
3.1.5	More . . . . .	6
3.1.6	Playback . . . . .	6
3.1.7	Help . . . . .	6
3.2	Toolbar . . . . .	6
3.3	Playback Controls . . . . .	7
3.4	Midi In Controls . . . . .	7
3.5	Object Menu Bar . . . . .	7
3.6	Object Menus . . . . .	7
3.6.1	ScoreMenu . . . . .	8
3.6.2	Movements Menu . . . . .	8
3.6.3	Staff/Voices Menu . . . . .	8
3.6.4	Clefs Menu . . . . .	8
3.6.5	Keys Menu . . . . .	8
3.6.6	Time Signatures Menu . . . . .	8
3.6.7	Measures Menu . . . . .	8
3.6.8	Chords Menu . . . . .	8
3.6.9	Notes/Rests . . . . .	8
3.6.10	Directives . . . . .	8
3.6.11	Lyrics . . . . .	8
3.6.12	Notation Magick Menu . . . . .	9
3.7	Palettes . . . . .	9
3.8	The Command Center . . . . .	9
3.9	Music Snippets . . . . .	9
3.9.1	Overview . . . . .	9
3.9.2	Details . . . . .	9
<b>4</b>	<b>The Print View Window</b>	<b>9</b>
4.1	The Buttons . . . . .	9
4.2	Mouse Controls . . . . .	10
4.3	Editing in the Print View . . . . .	10
4.4	Editing Positions and Padding . . . . .	10
4.5	Re-shaping Slurs and Ties . . . . .	10
4.6	Adding Line or Page Breaks. . . . .	10
<b>5</b>	<b>The Object Inspector</b>	<b>10</b>
<b>6</b>	<b>Object Editor</b>	<b>11</b>
<b>7</b>	<b>The Score Layouts Window</b>	<b>11</b>
<b>8</b>	<b>Score and Movement Button Bars</b>	<b>11</b>

<b>9</b>	<b>A Survey of the Input Methods</b>	<b>11</b>
9.1	Introduction . . . . .	11
9.2	Playing Notes into Denemo - MIDI . . . . .	12
9.3	Playing Notes into Denemo - Audio (Soundcard Mic Input) . . . . .	12
9.4	Using the Denemo From the PC keyboard . . . . .	13
9.4.1	Note Entry . . . . .	13
9.4.2	Chord Entry . . . . .	13
<b>10</b>	<b>Transcribing from Facsimile or Hand-written Score</b>	<b>13</b>
<b>11</b>	<b>Using a Proof-Read PDF</b>	<b>13</b>
<b>12</b>	<b>Score Setup</b>	<b>13</b>
12.1	Using Templates to Setup Scores . . . . .	14
<b>III</b>	<b>Advanced Topics</b>	<b>15</b>
<b>13</b>	<b>Customizing Denemo</b>	<b>15</b>
13.1	Adding and Editing Key/Mouse Shortcuts . . . . .	15
13.2	More Commands . . . . .	15
13.2.1	Recording a Denemo Macro . . . . .	15
13.2.2	Editing a LilyPond Tweak . . . . .	15
<b>14</b>	<b>Writing Scheme Scripts</b>	<b>16</b>
14.1	Commands Getting User Input . . . . .	16
14.2	Commands Getting Information about Object at Cursor . . . . .	16
14.3	Predicates . . . . .	16
14.4	Iterators . . . . .	17
14.5	Message Bar . . . . .	17
14.6	Move and Search . . . . .	18
14.7	More ... . . . .	18
14.8	... and More . . . . .	28
14.8.1	Denemo Directives . . . . .	29
14.9	LilyPond Editing . . . . .	29
14.9.1	Using the LilyPond Window . . . . .	29
14.10	More Features . . . . .	30
14.10.1	Piano Staves, Orchestral Scores etc . . . . .	30
14.11	Single Staff Polyphony . . . . .	30
14.12	Entering Figured Bass . . . . .	30
14.13	Fret Diagrams . . . . .	30
14.14	Tablature . . . . .	30
14.15	Entering Chord Symbols . . . . .	30
14.16	Musical Scores that Do Things! . . . . .	30
14.16.1	What Happens at Startup . . . . .	30
14.17	Starting Denemo - Command Line Options . . . . .	30
<b>IV</b>	<b>Technical Reference - Denemo Directives</b>	<b>31</b>
<b>15</b>	<b>Denemo Objects</b>	<b>31</b>
<b>16</b>	<b>Denemo Directives</b>	<b>31</b>
16.1	The Directive Fields . . . . .	32
16.2	Directive Edit Scripts . . . . .	33
16.2.1	Initialization Scripts . . . . .	33
16.2.2	Edit Scripts . . . . .	33
<b>V</b>	<b>Obtaining and Installing Denemo</b>	<b>34</b>
.1	To install from source code: . . . . .	34
.2	Generating a Configure Script . . . . .	35

## Part I

# Getting Started

## Introduction

Denemo lets you create musical scores. You can type music in using the keyboard or play it in using a MIDI controller or the microphone input. You can edit your music - the input display window shows you what you are working on in music notation - and you can playback to check how it sounds. The Print View window shows the final printed score, typeset to the highest standards used in the music publishing industry - thanks to the Lilypond music typesetting program.

This separation of the final typesetting window from the input display avoids the constant dragging about of overlapping notation which is typical of music notation programs. It also means you are less likely to input a slur when you mean a tie, or a whole note rest when you mean a whole measure rest, for example - these often look similar in the typeset and are easily confused, but Denemo's input display makes clear the distinction. Nevertheless, if you do wish to tweak the appearance dragging and re-positioning of some notation is possible on the final typeset display.

## 1 Input Methods

Most people will start with playing around with Denemo using the mouse, but by the time you are reading this you will want something more efficient. Try using the pc keyboard. By default, keys **a-g** represent the note names and 0-6 the different durations (whole note, half note etc). When the cursor is appending (blue) pressing key 0 will insert a whole note at the cursor. Pressing the **a** key will append the nearest A. If the duration of the next note is the same you can simply type the note name, it will use the last entered duration. There are keypresses (+/-, and plus/minus on the numeric keypad) to set sharp/flat/double-sharp/double-flat for the next entered note and to sharpen or flatten. Use +/- with the shift key to sharpen or flatten an existing note. Likewise, Shift with a number key edits the duration of a note.

When the cursor is moved on to a note (with the arrow key or mouse) the cursor turns green and then **a-g** edits the note name. To insert a note before a note at the cursor use a double strike **A,A-G,G**. When the cursor is green the keys 0-6 change the duration of the note, while Shift-0-6 insert a note before the cursor. (If you use the numeric keypad you can use **Shift-KP 0-6** for this). The arrow keys move the cursor around, the period key adds a dot while **Alt-a-g** add notes to the chord at the cursor. **Ins** inserts a note in a chord whether the cursor is on (green) or after (blue) a note, a double **Del**, **Del** removes it.

When in the appending position the cursor shows as a large blue or red rectangle on a note-position. Blue indicates a note can be appended at that point without overflowing the measure. Red indicates that the measure is already full. When the cursor is a smaller green rectangle it indicates that you are not in the appending position: you can edit the note/object at the cursor or insert before it. The vertical blue line indicates the insertion point.

For a more detailed description of Denemo's various input methods, see A Survey of the Input Methods. You can find more shortcuts by exploring the menu system - e.g. under the Notes/Rests menu the Select Duration submenu gives Remove Dot with the shortcut Control-period shown in blue next to the command.

Alternatively, a list is available in the Help menu. There are thousands of commands in Denemo, so get to know the Command Center where you can search for commands by key words such as slur, ossia, beam, Da Capo, cresc. etc.

Initially, Denemo starts with tooltips popping up almost everywhere. When they become too annoying you can tame them with Help->Turn Excessive Tooltips (Off/On), and you can further delay the remaining tooltips appearing via Edit->Change Preferences.

And once you have stopped playing around with the mouse for entering notes you can hide the palettes at the side to give more room for the music display (right click and Edit Palette).

### 1.1 Some Common Keyboard Shortcuts

Here are a few of the keyboard shortcuts that are commonly used in Denemo.

- Letters **a-g** edit the note at the cursor to be A-G, if the cursor is in the appending position then notes are added. Letters **A-G** (either **CapsLock** or **Shift**) insert a note at the cursor.
- Numbers 0-6 are used to refer to the note durations Whole Note ... 64th Note. They insert a duration at the cursor, which you then give a pitch to with a note name. With the shift key held (or the CapsLock on) 0-6 edits the duration of the note at the cursor. The corresponding Numeric Keypad keys can be used instead.
- The period (.) dots a note, **Ctrl-.** removes a dot, on the numeric keypad Decimal (.) cycles through dotted, double-dotted, no-dot.
- **Alt-0-6** inserts a rest.
- **Alt-a-g** Add notes to a chord. Or position the cursor and use **Insert** to add a note, **Ctrl-Insert** to remove it.
- 7 starts a slur, 8 extends it, while 9 reduces it.
- **Shift-/** inserts a cautionary accidental.

- **Esc** switches between various views of the music which can allow more music on the screen at once.
- **TAB** alternately inserts a start or stop triplet marker.
- **Multiply** (\* on the numeric keypad) ties/unties the note at the cursor.
- **Divide** (/ on the numeric keypad) sets/uses the note at the cursor as a grace note.
- **Add/Subtract** (+/- on the numeric keypad) sharpens/flattens the note at the cursor.

Two-key shortcuts are also available such as "**B,s**" for start repeat barline and "**B,e**" for end repeat barline.

## 1.2 Some Common Mouse Shortcuts

Here are a few of the mouse shortcuts that are commonly used in Denemo (keyboards may vary, for control, shift, alt etc modifiers). Note that under the Input menu is an option to turn on more mouse-friendly buttons, this is the default for Windows users.

- **Scroll Wheel** pans up/down to bring staves out of view into the window.
- **Shift Key** and **Scroll Wheel** pans the score left/right.
- **Control Key** and **Scroll Wheel** zooms.
- **Right-button** click edits at the cursor.
- **Double-click** brings up the **Object Inspector** on the current object, from there you can launch the **Object Editor**.
- **Shift-right-button** click edits things attached to the object at the cursor.
- **Double click left-button** describes the object clicked on.
- **Ctrl-Shift** left-button drag allow you to move stuff in the display if it is cluttered. Typesetting is not affected.
- **Ctrl-Shift Key** and **Right mouse button** gives the menu of directives to insert at the cursor.
- **Shift Key** and **Left mouse button** drags notes up and down.

## 1.3 Some Common Uses of MIDI in

The MIDI keyboard too can be customized to perform different actions. Usually, playing notes adds or edits the score (like hitting note names at the pc-keyboard). With the sustain pedal pressed chords are generated (the **Alt** key can be used for this too). If the interval between the notes played is augmented or diminished it is played on a different channel, so that you are alerted to possible pitch spelling errors (e.g. inputting A-sharp for B-flat). By holding down the **Ctrl** key a score can be checked by playing the notes - the cursor only advances if the correct note is played, and the **Shift** key can be held down to route the MIDI keyboard straight to the output (e.g. to check a phrase before playing it in). There is a button in the MIDI-in Controls to do this too. The Pitch Bend controller can be used to set the range of sharps and flats to be used, and the modulation controller can be used to mark sections of the music. With the MIDI controller set as Input source (Input menu) the duration keys create pure durations (notes colored yellow/brown) - you can enter as much of the rhythm as you wish, and then play the notes on top.

## 1.4 Note and Rest Entry Palettes

On starting Denemo for the very first time a series of palettes are placed near the main window with buttons to insert and change notes. For more serious use these just waste space, so by right clicking on them and choosing **Edit this Palette** they can be hidden.

## Part II

# Using Denemo

## 2 General Concepts

The unit of work in Denemo is a musical score, which can be saved in a single file (with .denemo suffix). This is represented on the screen by a "tab". If you have several tabs open at once they appear just above the music in the main window, and you can switch between them by clicking on the tab.

One score may contain several movements, which you can move between (PgUp, PgDown), insert duplicate, merge and delete with the Movements menu. A movement is a continuous piece of music with titles etc.

When you have more than one movement numbered buttons appear in the score titles bar for you to navigate by. The first thing on the status bar after any pending accidental is the movement number.

Within a movement there are staves (arranged vertically) and within the staves Denemo Objects. These can be notes, chords, key changes, time signature changes and Denemo Directives (see Denemo Directives). The notes and chords are displayed in conventional format (though not fully typeset). The Denemo Directives are used for most things that are not chords or notes etc: Metronome marks, repeat barlines etc are good examples. They can also be attached to chords, individual notes in a chord, to a staff as a whole and to the score as a whole; in this case the directive can be thought of as an attribute of the object it is attached to. Each Denemo Directive carries its own display method. For example the Close Repeat barline appears as a Denemo Object in the input display, while a Directive attached to a staff (e.g. Instrument Name, or Smaller Staff Size) may appear in a menu under a tools icon to the right of the staff. In the case of "Smaller Staff" the directive directs the LilyPond typesetter to make the staff smaller, and it can be edited from the menu under the tools icon to the left of the clef. Another tools icon appears if directives are attached to voices within a staff, again to the left of the clef, below the staff one.

If there is more than one voice on a staff it is best displayed on a separate staff in the Denemo Display, for ease of editing - the clef is drawn pale and there is no timesignature for such extra voices so it is easy to understand what is going on. The Print Preview window as usual shows the final typeset appearance.

In the Denemo display a cursor shows where the next note will be entered/edited. It is red for an over-full bar, blue for appending into an under-full one and green when editing (i.e. on an already entered object).

The menus in Denemo are unusual: they not only let you do some particular action, but also each menu item lets you add the command to a palette, enquire what the action does in more detail and set keyboard/mouse shortcuts for the action. A single keystroke can be set as a shortcut simply by pressing the key while the menu item is selected. All the extra functionality of menu items can be accessed by right-clicking the menu item, while the usual left click is for executing the action itself. The menus can always be torn off for working with particular items (e.g. working with different movements or with measures, dynamics etc.). When you place a command in a palette you can add to an existing palette or create a new one - you can choose or create any number of these, free-floating or docked in the main display. Also available by right-clicking is creating new actions - often by modifying ones that are already there - using the Scheme scripting window.

When a file is loaded it opens at the point where you left off editing it. The position and size of the window is restored as well as the position and size of the source pdf you are transcribing from.

## 3 Denemo Main Window

The main window has menus and toolbars at the top, and palettes at top and/or right hand side. Which menus and palettes are shown are selected via the View menu. At the bottom is a status line showing which movement you are in and what sort of object the cursor is on. If there are any MIDI filters that active they will be noted at the right in the status bar.

In between is where the music input is displayed, the Denemo Display area. When zoomed out you see just the few measures you are working on. By dragging the red bar at the bottom of the score upwards you get space for more of the music. If you have many staves they may not all fit: you can still drag the red bar upwards to see several lines of just one or two staves for instance. You can hide staves in the display if needed using the Staves->Display Effects menu. If you need more room you can hide the menus (using the View Menu), and arrange a "page view" of the input music - useful when using playback.

### 3.1 The Main Menubar

The Main Menubar has menus for overall control of the program. It contains the following submenus:

- File
- Navigation
- Edit
- View
- Input

File Navigation Edit View Input Playback More Help Educational

Playback Control

Tempo: 150 Set Tempo Volume 1.00

Midi In Control

Enharmonic selection

Shift Accidentals Flatwise *minus* C C# D Eb E F F# G G# A Bb B Shift Accidentals Sharpwise *equal*

MIDI in -> Score

Create Snippet Delete Snippet

Score Movements Staves/Voices Clefs Keys Time Signatures Measures Chords Notes/Rests Directives Lyrics

Print transposed: c es Score Font: 20 Print Part (Print) Transposed

Movement Title: III - Minuetto Movement Piece: Allegro (Page Break)

Movement 3: Note : Staff 1 Measure 71 Object 0 Not Appending Not Off End MIDI Input: No filtering

- Playback
- More
- Educational
- Help

### 3.1.1 File

Use the **File** menu to perform global operations related to storing and retrieving from file systems, importing from MusicXML, MIDI, and LilyPond formats, printing full score or parts, exporting to MIDI, Ogg, Wav, LilyPond (parts or score), PDF and Png formats. This is also where you open PDF files containing source material - e.g. manuscripts - that you wish to transcribe or proof-reading comments you wish to incorporate in your score. Another sort of source material can be loaded from here - audio files can be loaded for transcribing as well as MIDI files.

### 3.1.2 Edit

The **Edit** menu collects command for editing: mostly editing objects in the display, the Denemo Objects but also editing global properties of the score and your preferences.

There are commands for deleting the object before the cursor, the object at the cursor and for partially deleting from the object at the cursor (e.g. deleting notes from chords). Other delete commands are in the Object Menu (deleting movements, staves, measures etc).

There are commands for editing all the Denemo Objects of a chosen type across the whole of a score. Commands for managing *the selection* are also here. Editing using the keyboard to invoke a palette button is here too (usual shortcut is p).

### 3.1.3 View

Use the **View** menu to toggle toolbars and palettes used with the mouse.

Menu Item	Description
Hide/Show Menus	Three ways of showing the Denemo display area are supported. With/without the menus and as a multiline page (actually, any page can be a multi-line page, but usually the third one is chosen for this. This command cycles through the three displays - usually the Escape key is the shortcut.
Typeset Music	Shows the score as engraved by the LilyPond typesetter. Some things such as the shape of slurs and position of marks can be edited here graphically.
Command Center	Search for commands by entering likely words, set up one key or two key shortcuts, load customized shortcuts or commands ...
Score Layout	Show the score layouts associated with this score. The score layout is the final section of the LilyPond syntax that describes how to layout the staves, voices, lyrics, titles etc.
Lyrics	Shows any lyrics for the current staff/voice. Each verse has its own tab, when selected the lyric syllables are underlaid in the Denemo display, so you can adjust by typing in the lyrics view window
Snippets	Shows a menu bar with snippets - a selection of Denemo Objects used either for pasting or to define a rhythmic pattern to be followed when entering pitches.
Tools	The conventional icons for Open, Print etc
Playback Controls	When checked a set of playback/record controls are placed above the display
Midi In Control	When checked a set of controls for a connected MIDI keyboard are placed above the display
Score Titles, Controls etc	If this is checked any Titles, indent settings etc applying to the score and movement can be shown as buttons above the display. They must be created with their graphic field set for this
Object Menu	Menus of all the Denemo commands listed under type of object from Score down to the type of Denemo Objects.
LilyPond	Pops up a window for customizing the syntax that Denemo generates for the LilyPond engraver to typeset
Scheme Script	Pops up a window for showing scripts written in Scheme. These can be executed or saved as new commands. Sequences of commands can be recorded here.
Score	This hides/shows the main Denemo display.
Palettes	Gives access to palettes of buttons that can be arranged to make commands available via mouse or keyboard (via the Activate Palette Button command).
Display Zoom	Zoom the main Denemo display (usually done with ctrl-mouse wheel).



### 3.1.4 Input

Use the **Input** menu to select external sources (Audio from the Mic input or MIDI) for inputting notes to Denemo. To use these you should review the settings in Edit->Change Preferences MIDI and audio tabs first - change the setting for the backend from “default” to the specific controller you have.

Here too there are commands to change the way the inputs (keyboard, mouse and MIDI) behave.

### 3.1.5 More

Use the **More** menu add commands to Denemo. Extra commands are available (those which not everyone will want) via this menu. More Commands gives commands shipped with Denemo, while My Commands gives ones that you have created locally. Note that the extra commands can also be loaded at the menu where you are looking for them (you right-click on a menu item, and if there are more commands for that menu the More Commands for this Menu item will show),

### 3.1.6 Playback


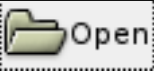
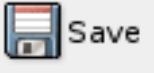








Use the **Playback** menu to listen to the current movement. Denemo lets you hear your score, using an internal synthesizer. There are commands to play through the all the notes or to observe repeats or to play just the chord or notes at the cursor and to play at a shifted pitch.

### 3.1.7 Help

Use the **Help** menu to get help using Denemo. A list of the shortcuts is available (including any you have set yourself), a feature-packed score can be loaded, the excessive help can be tamed, and your browser can be launched on the Denemo chat room for asking questions directly from other users.

## 3.2 Toolbar

Use the Toolbar to access common Denemo commands via a mouse click. The Toolbar contains the following icons:

Icon	Description
	Creates a new document.
	Opens the Open File dialog box.
	Saves the current file
	Runs LilyPond to convert the current file to PDF and sends it to the printer for printing
	Undoes the last action.
	Redoes the previous undo action.
	Cuts the current selection to the clipboard.
	Copies the selected notation to the clipboard.
	Pastes the current clipboard item at the cursor position; to be safe always create blank measures to fit.
	Jumps to the first measure of the movement.
	Jumps to the last measure of the movement.

### 3.3 Playback Controls



The playback starts at the playback start marker (a line in green down the score in the Denemo Display) and stops at the playback end marker (a red line). These lines are not displayed all the time, just once the play has been used at least once. The start/end markers can be altered with the arrow keys on either side of the Play and Record buttons. The arrows to the left adjust the playback start, those to the right the playback end (hover over the arrows for details).

Playing of sections of the music and looping is possible (including editing as the music loops, so that you can listen to different possibilities) as well as recording the audio output (mixed with anything you play on your MIDI keyboard if you have recording set in the MIDI controls).

The master tempo and volume can be set, which will be overridden by tempo and dynamics placed in the music.

Recording the audio output is also possible as well as real-time slow down of the audio playing back (not to be confused with simply setting a slower tempo!). This last is used when transcribing from audio

### 3.4 Midi In Controls

These are the controls for a MIDI keyboard attached to the computer. You should set the MIDI Input Device to your device in the Preferences to be sure all features are working.

The Enharmonic Selection control lets you determine what accidental will be used for the MIDI notes - e.g whether to enter C-sharp or D-flat. As you enter music if you enter an augmented or diminished interval a different instrument is used which helps you avoid pitch-spelling errors. (See Edit->Change Preferences->Audio pitch spelling channel/program to turn this off/change instrument).

**[Appending/Editing]** button: This shows how MIDI in will be treated. Press Control/Shift/Alt keys to modify or click and select Listening to input MIDI, Checking notes in the score against input MIDI or Appending/Editing at the cursor.

**[Switch to Play Along Playback]** button: When you press Play with this option set, the music will not advance past the cursor until you play the note. (Mute the current staff to prevent double sounding of notes)

The **[record]** button allows you to record from your MIDI keyboard while the score is playing back. (Press MIDI record the Play). The MIDI recording will be played back with the score, and can be converted to notation or deleted.

### 3.5 Object Menu Bar

The object menu bar gives a set of menus give ordered by the objects (notes, staves, measures ...) that they relate to. This division cannot be perfect - some activities could be placed in different menus, so you need to be prepared to search for commands using the Command Center.

These are the menus on the Object Menu Bar with a brief description of what they hold.

Menu Item	Description
Score	Settings that apply to the whole score.
Movements	Insert/remove navigate, change the properties of movements.
Staves/Voices	Insert/remove, navigate, swap, change the properties of staves or voices. (Voices in this context mean a staff of music)
Clefs	Insert clef change or set the initial clef.
Keys	Insert a key change or set the initial key
Time Signatures	Insert a change of time signature or set the initial time signature, control the typeset appearance.
Measures	Insert/remove navigate measures.
Chords	Insert/remove/alter notes of a chord. Note that Chord Symbols can created markings on notes (see Notes/Rests menu)
Notes/Rests	Menu items for inserting/changing/deleting note and rests, beaming, ties, slurs, setting the notehead style, grace notes
Directives	These are objects inserted between the notes of a score to place marks or start/stop spanning effects such as text
Notation Magick	Create and manipulate music auto-magick-ally.
Lyrics	Create and lyric verses for the current staff.

### 3.6 Object Menus

Note: Dynamics, Slurs, Trills, Tempo Indications etc are in the Notes/Rests menu (if attached to a note or chord) or Directives menu (if standalone objects) . The advantage of attaching them to a note is that they can be moved around as a unit, but the standalone objects are generally easier to drag in the Print View should that be needed.

### 3.6.1 ScoreMenu

The score menu lets you control things that apply to all the movements of the current score. Also here is the Check Score command which should be your first port of call if your score will not typeset.

The Score Properties command gives access to some built-in properties that affect the whole piece - paper size and display appearance.

With Edit Directives you can edit directives that apply to all the movements, most of the rest of the commands create these Denemo Directives attaching them to the score under several categories (layout, paper, header etc).

Titles submenu: There are two sorts of titles. Book titles have a separate title page with titles for individual movements which can be listed in an automatically generated table of contents. Simple titles give a Title and (optionally) movement titles on the same page.

Comments (critical comments) can be placed on chords in the score and these can be automatically collected into an appendix using the Book Titles->Epilog submenu.

A Table of Contents can be automatically generated from the movement titles by setting Book Titles->Table of Contents Title.

Graphic title pages can also be created.

### 3.6.2 Movements Menu

As with the Score Menu there may be Denemo Directives attached to the movement - e.g titles to be placed at the head of the movement will have a Denemo Directive associated with them. Other controls that can be applied on a per-movement basis include the barline and notehead style, printing of a custos at line ends, the indent before the first system (which can also be set on the score-wide basis).

Various score checking routines work at movement level from this menu.

### 3.6.3 Staff/Voices Menu

Things such as the music crossing to another staff, or Ossia staves are contained here, along with commands for deleting parts of the staff, setting up multi-measure rests for a whole staff, muting the staff during playback

#### The Staff Properties Submenu

This covers such things as the type of staff (e.g. Normal Notation, Tablature, Chord Symbols, Fret Diagrams) as well as the staff appearance and the playback instrument, instrument name or ambitus to be typeset at the start of the staff and more.

#### The Voices Submenu

*Voices* in Denemo are like staves but are typeset on the preceding staff to achieve single-staff polyphony. The Denemo Display is normally set to display any extra voices on a separate staff so as to make it easy to work with - the voices have their clef displayed in yellow and do not show a key signature as these are supplied by the main staff (or “primary voice”). Voices will also normally be assigned to voice numbers 1,2,3,4 using the Initial Voice commands in this menu (this controls the stemming, slur positions etc). Voices can change their voice number during the music - the commands for this are in the Directives->Typesetter->Voices menu as they insert stem change objects in between the notes.

#### The Staff Groupings (Braces) Submenu

The braces at the start of each system are controlled from here.

### 3.6.4 Clefs Menu

### 3.6.5 Keys Menu

### 3.6.6 Time Signatures Menu

### 3.6.7 Measures Menu

### 3.6.8 Chords Menu

### 3.6.9 Notes/Rests

### 3.6.10 Directives

### 3.6.11 Lyrics

To add lyrics to a staff move the cursor onto the staff and choose Add Lyric Verse. You need to have the Lyrics View visible (see View menu). Each verse has a separate tab, with the current verse for the current staff being visible at any one time.

This means that you will see the lyrics for the current staff disappear from the Display Window when you switch staves, which can be disconcerting, but saves space in the display.

Stanza numbers can be entered as well as font-changing commands. Use **Melismata** command to extend syllables over more notes. The lyrics menu also contains a command “Edit Lyric at Cursor” ( Ctrl-l, Ctrl-l) which finds the place in the lyrics corresponding to the cursor position allowing you to edit the syllable that underlies the current note. Verses can be typeset in columns at the end by using command Typeset Verses at End which invokes the external Inkscape (or other) editor.

### 3.6.12 Notation Magick Menu

## 3.7 Palettes

Palettes are collections of commands arranged as a column, row or grid of buttons either in a separate window or docked in the Main Window.

These commands can be executed by clicking with the mouse or typing **p** followed by (part of) the label of the button followed by the **Enter** or **Return** key. (The palette must be selected - you can do this while typing the label or by using a button).

In the View menu you can select palettes of commands to use. Palette items can be invoked from the keyboard (see the Edit Menu) or by clicking on them. Right-clicking on a palette button allows you to edit the button or the entire palette. You can dock the palette in the main display, arrange it as a column, line or block and much more.

You should make palettes your best friend - when you find a command in the menu system that you need right click on it and choose Add to Palette. This way you will build up one or more collections of commands that you need. By right clicking on a palette button you can choose a name for the command that suits you, and even make special notes about its use in the tooltip.

## 3.8 The Command Center

In the View menu you can show the Command Center. This has a list of all the Denemo commands, hidden or not, with their shortcut(s). The box at the top left gives full details of the currently selected command and its location in the menu system. Below that is a search box for finding any command you need by matching words in its description or label. By choosing “Fuzzy” you will make the search less strict - one word could be missing.

The search starts as you type in the first letter of any key words you want, so check after each letter to see if you have found the command you need. To search for further commands matching your keywords press the arrow next to the search box.

The command center is also where you can set one-key shortcuts such as **Ctrl-x** and two-key shortcuts such as **Shift-a** followed by **Shift-a** (which is written as A,A next to the command). You can also lookup which command responds to a given shortcut, and execute the selected command here.

In addition you can save and load whole sets of shortcuts and commands.

## 3.9 Music Snippets

### 3.9.1 Overview

Music snippets are short selections from a voice which can be stored on the Snippet tool bar. They have two main uses: they can act as a repository of “motifs” which can be inserted (**Ctrl-Space**) at the cursor, and they can be used as an extension to the “prevailing duration” idea. In the second use, by selecting a snippet as you enter pitches the durations are assigned from the next step in the snippet. The first case is especially useful when entering a complex rhythm which is repeated many times in the piece. In all cases the snippet can contain all sorts of attributes and markings, slurs, beaming indications, ornaments etc, and these will then all be entered with a single keypress or as you add pitches.

### 3.9.2 Details

The Snippet tool bar has only one button initially: “Create”. This button lets you create a snippet from the current selection. The selection should be a contiguous selection of objects in a single voice. Once created the snippet appears on the snippet tool bar as a button - the label is generated to indicate the content, and this label can be further edited by clicking on the button and choosing “Edit Label”. Other options on clicking a snippet button are to insert the snippet, to select the snippet (in which case as you enter pitches they follow the rhythm of the snippet, including any slurs, trills, or articulations that are in the snippet) etc.

**Ctrl-Space** inserts the selected snippet or (if none is selected) inserts the snippet that you specify by number. **Shift-Space** selects the next snippet (or first if none selected), to unselect a snippet choose a duration (0, 1...) to be used instead for entering pitches.

In the insertion case, you may have two or three snippets that apply to a given score - special bits of text that appear over notes for example - and you can insert them using **Ctrl-Space**, 1 (or 2 or 3 ...) as needed.

In the pattern-following case you will be selecting snippets using **Shift-Space** and then playing in notes (and backing up using backspace in case of error - the pattern backs up for you).

## 4 The Print View Window

The print view window is shown by checking View->Print View. This view shows the typeset score as it will be printed on paper. It also allows you to locate the place in the Denemo Display belonging to a particular note or mark in the score, and allows some graphical editing of the score.

### 4.1 The Buttons

- Print starts your system’s print dialog to send the typeset score to a printer or file.
- PDF to generate a PDF document from the score.
- Typeset. Offers a menu of score layouts to typeset. If only one is available (the Default Score Layout) then it typesets that one.

- **Movement.** Creates a Score Layout comprising the current movement, and then typesets it .
- **Part.** Creates a Score Layout from the current Part and typesets that. A *part* comprises all the staves with the part name of the current staff. (The part name of a staff is shown above the initial clef with a green patch on it).
- **Refresh.** Repeats that last typeset command. For example, if Part was the last then it typesets the current part (which may be different from the one last typeset).
- **Continuous/Manual.** This both indicates the current mode of typesetting and acts as a menu for altering it and choosing how much to typeset in continuous mode. In continuous mode the score is re-typeset every time it is altered - choose the range if the score is large so that it refreshes quickly. For entering music rapidly set this to Manual, and re-typeset on demand.
- **Duplex.** Changes the view to show how two-sided printing will work out for page turns.
- **Next, Previous.** Navigate to the next/previous page of the score. You can scroll or drag as well.

## 4.2 Mouse Controls

The mouse scroll wheel enables vertical panning and with Shift horizontal panning. With Control pressed you can zoom the view, which is very important for re-shaping slurs. Right click for help with tweaking beam angles, slurs etc.

Right clicking in a blank area of the typeset view (normal cursor) gives a menu providing help and allowing you to typeset with the control points for curves and the location points of objects marked as red dots and crosses.

## 4.3 Editing in the Print View

Clicking on the typeset score can locate the place in the Denemo Display belonging to it. For this to work look for the “hand” pointer as you move the mouse over the typeset score. Use a big enough zoom factor to make this easy. The hand pointer indicates a place where if you click the Denemo Cursor will move to the Denemo Object that belongs to that feature.

## 4.4 Editing Positions and Padding

Right clicking on objects in this pane allows you to drag items that LilyPond has positioned badly. Alternatively, you can select an amount of padding to apply to some objects so that they keep their distance from other items in the score. This is generally a better way of indicating that it is too close to something else, since other objects will move away from it as needed - that is, if you later edit something it will behave more intelligently.

## 4.5 Re-shaping Slurs and Ties

Slurs and beaming can also be altered here. When you right-click on a slur or tie the Denemo Cursor moves to the note where the slur starts and you are offered the chance to edit the shape of the curve. Choose a large zoom factor before you begin, and turn on the control points (red dots and crosses) as described Mouse Controls above. Then right click on the curve and choose Edit Shape. You have to click on the reference point first - this is on the center-line of the staff at the horizontal position of the notehead where the curve starts - this is marked with a red dot. Then you click and drag on the four “control-points” that define the shape of the curve. Once the shape looks good, right click in a blank area and click Apply.

With practice you can skip the use of the red dots and crosses as you can guess where they would be.

## 4.6 Adding Line or Page Breaks.

Page or line breaks added by right-clicking on a note at a barline and choosing from the menu. The click positions the Denemo Cursor and then the page or line break command is run just as if you were working directly in the Denemo Display.

# 5 The Object Inspector

The Object Inspector is shown by clicking View->Object Inspector, or just double-clicking on an object in the Denemo Display. It gives comprehensive information about the object at the cursor. This includes directives like ornaments, dynamic markings, fingerings that may be attached to the object. The Object Inspector is sensitive to which note of a chord the cursor is on, so that things like fingerings (which can vary from note to note within a chord) are displayed individually as you move the cursor up and down a chord. You can step through the notes of a chord using the button marked “Inspect the next note in chord” which will work upwards through the notes before wrapping round to start at the lowest note.

For many directives full information about which command created the directive and where it is to be found in the menu system is given, and the description of the command’s behavior.

---

Some older commands don’t have full information but the tag associated with the directive will usually help in tracking it down.

---

The Object Inspector automatically updates itself as you move the cursor around.

The Edit button will launch the Object Editor.

Use the window controls on the Object Inspector’s title bar to dismiss the window.

## 6 Object Editor

The Object Editor allows you to edit the current Denemo Object. It can be launched from the Edit Menu or from the The Object Inspector.

This window is *modal*, that is you cannot interact with Denemo outside of the editor until you have dismissed it. To warn you of this the pointer turns to an X if you try to interact with the Denemo Input Display when the Object Editor is still open. Buttons are provided to move on to editing the next (or previous) object.

As with the Object Inspector the Object Editor is sensitive to the cursor height, with the same navigation control to move through the notes of a chord so as to allow editing the detail of each note.

## 7 The Score Layouts Window

The score layouts window is shown by checking View->Score Layout.

You can use layouts to print the music in different ways. For example, one layout may have the score transposed, or be for just two parts printed together.

You can customize any layout in this window - positioning lyrics above the staff rather than below for example, or setting the staff groupings.

You can make Denemo Directives conditional on the score layout, so that for example a page break only applies when printing the full score, or is just for printing one part. In this way if, for example, one part does not need separate first and second time bars, then these can be marked as not being for that layout. See Directives->Conditional Directives for setting this.

The score layout can only be edited graphically when first created. Once re-loaded from disk it is reduced to its LilyPond text form and can then only be edited in the LilyPond view.

## 8 Score and Movement Button Bars

The View->Titles, Buttons etc" checkbox makes two horizontal button bars visible (if they have buttons on them). What the buttons show are titles/composer etc for the score and for the current movement. The score button bar holds the movement indicator buttons for scores that have several movements - you can click on one of these to move quickly to a different movement. In addition other score-wide settings create buttons on the score titles bar and movement wide settings on the movement titles bar. The menu items that populate these are under Score->Titles etc., and Movement->Titles etc and other menus.

## 9 A Survey of the Input Methods

### 9.1 Introduction

Denemo allows you to explore all the actions (insert/delete/edit/navigate...) using the menus. So, for example, inserting notes can be found under Notes/Rests while inserting a Staff comes under Staffs/Voices. When you have found the action you need hovering over the item will give an explanation of what it does. The keyboard or mouse shortcut to use for the command is in blue on the right. You can set the keyboard shortcut just by pressing the desired key while the menu item is selected. You can right click on the menu item to set a a two-key keyboard shortcut or mouse shortcut to activate the action. In addition you can customize the mouse actions to initiate commands, and set the cursor that will show while certain mouse conditions apply.

Denemo offers several ways of entering music: typing at the keyboard, playing in via a MIDI keyboard (controller), playing in acoustically using a microphone or choosing items from the menu system.

Using the menus for everything would be very slow, but next to each menu item is the keypress that you can use instead. If there is no keypress shown, and you want to use that item often, then right click on the menu item and you can set a key combination for that menu action. Right clicking is also useful for getting a description of what the menu action is.

Typing at the keyboard can be very fast, if you forget which keypress you need you can consult the menu system to find out. This method is good for touch typists.

For entering a lot of already written-out music, playing the music in via a MIDI keyboard or microphone can offer the fastest and most musical method.

One way to do this is enter the music as pure rhythms, ignoring the note names. The durations show as yellow/brown notes. Each duration sounds its own pitch and length as you enter the durations. As you start each measure the bell will sound so that you can keep your eyes on the score you are copying without needing to look up and check that you haven't miss-typed. If you are a reading musician you will find that you tap in the rhythm rhythmically which helps to keep your place in the piece you are entering.

Once you have entered the rhythms for one or two measures you play the notes via MIDI - they automatically fill in the rhythm for you. When you play notes in via MIDI they give their pitches to the rhythm you have notated. By this means you can enter a piece of music in the time taken to play it twice - once to give the rhythm and once to give the pitches. If the piece modulates strongly you may need to shift the set of accidentals used to match the score. If entering pitches via a mic you need to navigate to the start point, but this isn't needed for MIDI in.

## 9.2 Playing Notes into Denemo - MIDI

Denemo can take pitches directly from MIDI input. Select Input->Midi Input. The control panel allows you to choose the enharmonic range, the default centers around the initial key signature (e.g. E-flat to G-sharp for C-major). If you enter a diminished or augmented interval the note will be played in a separate MIDI channel which can be used to alert you to pitch-spelling mistakes (e.g. entering A-flat when G-sharp was meant, you will get a lot more augmented and diminished intervals if you have the enharmonic range set wrongly for your piece). Under Input->MIDI are various "MIDI filters" that allow you to control how the MIDI input is used: with none active the behavior is as if you had entered the notes using the pc-keyboard, but with the following advantages

- The octave, accidental and note name are all entered in one press of the MIDI key. The cursor automatically advances so you can continue to play in notes.
- The duration keys enter the yellow/brown duration-only notes, when you play a MIDI key the cursor automatically moves to the first of these.
- Holding down the **Alt** key (or sustain pedal) lets you enter chords.
- Holding down the **Ctrl** key lets you check the pitches of a piece already entered. The cursor only advances if the note played is the one at the cursor.
- Holding down the **Shift** key lets you listen to the MIDI keyboard without affecting the score.
- The problem of entering the wrong enharmonic is largely avoided by Denemo's simple pitch-spelling strategy. The more extreme intervals are played in a separate MIDI channel so that if, for example you enter F - A-sharp it sounds quite distinct from F - B-flat.

The MIDI filters are scheme scripts, so they can be tailored to do whatever you wish. One example is a filter that enables you to enter Figured Bass figures by playing the notes corresponding to the figures while holding down the bass note. Another is the "AngryDelete" filter. With this filter on notes are entered normally with the cursor advancing automatically, but if you make a mistake and press the wrong note just hit the next one much louder and it will make the correction for you! MIDI filters are found under Input -> MIDI

## 9.3 Playing Notes into Denemo - Audio (Soundcard Mic Input)

Denemo can listen for, and detect the pitch of notes on the mic input of the computer; it doesn't attempt to guess the rhythm - such systems do not work well - but you will find that playing the notes in time will help you to play them in, as well as make playing them in a musical experience rather than a chore.

Playing the notes in can be much quicker than using the keyboard since the note octave and accidental are all given just by playing the note. If you are able to play a musical instrument then this will probably be much faster for you than typing note names, octave shifts and accidentals at the computer keyboard. Using the headphones-out of an electronic keyboard avoids "noises-off" interfering with the pitch detection. Many microphones and pickups benefit from some pre-amplification - it is worth getting the level right before you begin.

When you select the Input->Audio on the Main Menu the Pitch Recognition window pops up. While the mouse pointer is inside the score drawing area the score is sensitive to pitches heard via the microphone input. The background colour of the score changes to show that the notes will be entered into the score.

There are two ways of using the pitch entry - Overlay mode (default) and Insert mode. The button marked Insert causes notes to be entered into the current measure in the prevailing rhythm - the mode is set to Insert for this. The button marked Overlays overlays the notes already present with the pitches you sound. There is a third button, marked Tuning, which is a state-of-the art musical instrument tuner.

Music is entered into the measure which holds the cursor. If you are overlaying a rhythm already entered, then the first un-overlayed note in the measure is overlayed by the note detected. You can delete the overlay using the regular delete keys, or clear them altogether if you want to start over in a measure. If you are in Insert mode then the notes detected will be inserted at the cursor position.

Use the enharmonic shift to select whether Bb or A# should be entered when you enter the given pitch - you can usefully go as far as B# and Fb.

Use the transpose control to shift up or down by octaves.

Most of the other settings would require study of the Aubio documentation to understand, but the one marked threshold may be useful to make the detection less sensitive to ambient noises if using a microphone with an acoustic instrument.

The best set-up is to plug the headphones-out socket of an electronic keyboard into the mic input, and choose a piano setting on the electronic keyboard.

If you don't have any musical instrument that you can plug directly into the mic in, then you can use an acoustic instrument with a microphone, in which case move your microphone closer or further from your instrument to get reliable detection. Too close and you get double detections, too far and you get missed ones. To check for good detection open a piece of music, set Overlay mode and put the cursor in the first measure and play the piece in - the notes should all turn blue if you have perfect detection. It is worth while getting perfect detection - more than one or two miss-detects per piece of music and you may want to use the Insert rather than the Overlay method.

The Audio Input button introduces a special entry mode where the pitches you play in will overlay the rhythm, appearing as blue notes. You can delete any wrong pitches using the usual delete keys, without deleting the rhythm. In fact if you have an "interloper" (an extra spurious note) you can delete it and the other pitches will all move along to their correct places.

Another method of playing music in acoustically doesn't involve entering the rhythm separately. For this select Input->Audio and then choose Insert instead of Overlays on the Pitch Recognition Panel that pops up. With Insert the sounded notes are entered as in the prevailing rhythm. The same applies if you have MIDI - by choosing a rhythm (e.g. half-note, quarter-note, or a custom rhythm pattern) and playing in the pitches you can enter the music into a blank score.

## 9.4 Using the Denemo From the PC keyboard

### 9.4.1 Note Entry

Basic Note entry is to type a number key 0-6 for the duration of the note followed by a letter key a-g for the note name. (The shorter duration notes are available, but the keys 7,8,9 are used for starting and extending/reducing slurs). If the duration is the same as the last entered note then just the note name is needed. The duration can be dotted with the period key, and the octave adjusted with comma (for down) or apostrophe (for up).

When you append music by pressing a duration key with a MIDI controller active, you get a brown note, indicating a duration which has not been given a pitch. Once you have pressed a note-name it becomes that note printed in black. Prior to that it is a pure rhythm and will play back as a drum beat and will print as a space.

When not appending music, you need to use the **shift** key to insert a new note, or set the **Caps Lock**. So **Shift-a**, **Shift-a** inserts the note A etc. Plain **a** edits the note at the cursor to be an A. Likewise with the numeric keypad the **Shift** key can be used to edit a note duration, while the plain number keys insert the duration of the note at the cursor. (If you do not have a numeric keypad, you have to use the number keys - best to plug in an additional USB keyboard with numeric keypad, they are very cheap nowadays).

Rests are entered with **Alt-0** **Alt-1** ...

Spacer rests (that is non-printing rests as seen in keyboard polyphony) have shortcuts too **Mod4-0** etc, where **Mod4** is the key with a flag on it. However, on Windows machines you will need to reclaim the Windows **Mod-4** key from Microsoft or re-define the shortcuts, e.g. to **Shift-Alt-0** etc.

### 9.4.2 Chord Entry

The standard keybinding for entering a note in a chord is the **Insert** key, **Ctrl-Insert** or **Del,Del** deletes. The is inserted at the cursor height, use **-** and **+** to flatten/sharpen the note before you enter it (avoiding hearing both notes if you flatten/sharpen afterwards). There are also commands to enter a named note **Alt-a**, **Alt-b**... are the standard shortcuts for this.

## 10 Transcribing from Facsimile or Hand-written Score

If you have a PDF of music you wish to transcribe with Denemo you can display it within Denemo and create links between the source material and your transcripion. This means that when you later review your work you can quickly locate which point in the original manuscript belongs to which point in your transcription

Select the **File->Open** menu and the item **Open a Source for Transcribing**. Select the PDF containing the music to be transcribed from. The PDF is opened in a **Source View** window. Right-click in this window at the point that your transcription will start. This will create a link - a Denemo Directive Object - in your score at the Denemo cursor position, shown as arrow icon. If you later use a right-click on this Denemo Directive the source file will be re-opened with the blue highlight marker on that point in the score. (Depending on the position of the top of the page you may need to scroll the **Source View** window up or down to bring it into view).

When you re-open the score to check or edit at some point you can click on a nearby link and the source file will be opened and the position highlighted once again.

## 11 Using a Proof-Read PDF

If you send the PDF output of your score to someone to proof-read they can make PDF annotations to it. In the **File->Open** menu choose **Open Proof Read PDF** to open a proof-read PDF. It will open at the first annotation, and, by clicking on the note the annotation applies to, the annotation is transferred as a comment into the score before the note chosen. When you have done all the annotations on the first page advance to the next page with annotations using the button "Next".

There are limitations - the PDF must be created from your score with point-and-click (the default) and you must not alter the score in a way that would invalidate the point-and-click. (Basically, don't edit again until the proof-reading is done, always good practice!) Also, enter the annotations in order from the start and don't edit before you have entered all the annotations, as this could invalidate the point-and-click.

## 12 Score Setup

Denemo provides different ways to set up scores:

- Create template with the special name "default.denemo". If this is placed at the top level of your templates directory (that is `~/denemo/templates/default.denemo`), then it will be opened each time you start a new score.



- Create an init.denemo score with everything you want to start up with, and any actions you want to be taken before you begin. This will be opened when you first start Denemo (that is once per Denemo session). See 14.16 on page 30 for details on what actions you can have Denemo perform before you start.
- Use a Template
- Use shortcut keys to add staves and set attributes
- Use Menus

## 12.1 Using Templates to Setup Scores

Denemo comes with a few preinstalled templates. You can also create your own templates as well, in fact you can use any Denemo file as a template, just open it using File->Open->Open Custom Template. (The only difference is that, opened this way, the score is a new i.e. untitled score)

1. Navigate to File, Open Template. The Open dialog box appears.
2. Double click on a template name from the list of templates. The dialog box closes and the template appears.
3. Adjust the clef, key and time signatures as needed.

# Advanced Topics

## 13 Customizing Denemo

### 13.1 Adding and Editing Key/Mouse Shortcuts

Denemo allows you to choose which keys activate which commands (shortcuts). It also allows you to choose Mouse press/release gestures (in combination with keys such as **Shift**, **NumLock**, **CapsLock** etc).

See The Command Center for setting shortcuts. The following method can also be used. To choose a shortcut for a menu item, select the item (it becomes highlighted) and press the key that you want to become the shortcut. For mouse shortcuts, right-click the menu item and choose the Create Mouse Shortcut, to change the pointer shape during mouse operations invoke the Command Center with the Edit Shortcuts option. The Set Mouse Shortcut Dialog requires you to set which button and action you want to use and then to hold/lock the keyboard modifier while clicking on the big button. The setting you have chosen is shown, and you click ok to accept it.

Mouse Shortcuts are tricky to set - you are able to control what happens on mouse button press and release, which can be used to do things like drag notes up and down. But they are tricky to set up!

If your choice of keyboard shortcut already belongs to another command, you are warned about this, and given the chance to change your mind or steal the shortcut. Also, you can choose whether to distinguish between keypresses with, say the **NumLock** down and those without or to ignore it. If the **NumLock** is set but no command for that key specifically requires it then the setting of **NumLock** is ignored and the keypress is treated as if the **NumLock** was off.

The menu item that activates the command shows the shortcuts that exist in bright blue lettering. **PrsL-CapsLoc**, for example, is the Press action on the Left mouse key while the **CapsLock** light is on. In the default command set this is linked to the **BeginSlur** command, so that together with the **RlsL-CapsLoc** for the release of the left mouse button, you can "draw" slurs by using the mouse. Similarly, **Shift** with left mouse button (**MveL-Shift**) is set to allow you to drag notes up and down the staff. (This is using Command **MoveNoteToCursor**).

### 13.2 More Commands

#### Adding More Features

It is possible to add more commands to the set that Denemo ships with. Some of these are shipped with Denemo, but not added automatically so as to keep the menu sizes manageable. Generally, you right-click to find more menu items to add; once added you can set a keyboard/mouse shortcut and keep the command in your default setup. Scheme to be executed at startup can be placed in the `denemo.scm` file.

**An example - quicker dynamics selection** It can be slow to choose dynamics (**Ctrl-D**) because the list is quite long to move down (or move to the mouse). You can define a list of the dynamics you actually want to be readily available as for example: (define DenemoDynamicList '(("f" "127" "Forte") ("p" "127" "Piano"))) This makes forte available just by pressing Return and piano available by pressing down arrow Return. (The others via More of course). If you place this in (home directory)/.denemo-x.x.x/actions/denemo.scm then it will be defined when you start Denemo.

#### 13.2.1 Recording a Denemo Macro

You can also get Denemo to record a frequently used sequence of actions - it can be installed in the menu system and given a keyboard shortcut just like any other command.

To do this choose View->Scheme Script. (You do not have to understand what a Scheme Script is to use this!). In the window that pops up check Record Scheme Script and then do the set of steps you wish Denemo to record.

As a simple example, suppose you wanted a command to delete the next note (there are commands to delete the previous note and the current note, so why not?). You would enter some notes, put on Record Scheme Script and then move the cursor right two steps and delete the previous note (with Backspace if that is your shortcut). (Each time you use a command you will see the Scheme syntax for that command entered into the Scheme window). Now turn off Record Scheme Script and you can experiment with your new command by pressing Execute in the Scheme Script window. To save this new command for future use, you right click on a menu item (in the menu where you would like the command to appear) and choose "Save Script as New Menu Item", this will ask you to make up a name for your new command as well as a label for the menu item etc.

You can save the command in your default sequence of commands (via Edit->Customize Commands...->Manage Command Set->Save as Default Command Set). If not you will be asked if you want to save your new commands when you exit. Otherwise you can re-load via the More->My Commands menu item.

#### 13.2.2 Editing a LilyPond Tweak

There is another way of adding your own favorite LilyPond tweaks, which is by modifying a tweak that has already been done. This can be done by using the text-edit dialog on an existing Denemo Directive (usually found under the Advanced button provided by

an edit script). Here you can see the actual LilyPond text that will be inserted, and there is a button to create a script to generate whatever you choose to enter. You can also enter the name of graphic images (.png files) that are to be used to represent your item in the display, and say where it should be positioned.

Once you have created the script, it can be saved in the menu system by right clicking on an item in the menu where you want to place the command, and choosing "Save Script as New Menu Item" as above.

## 14 Writing Scheme Scripts

If you are a programmer you will have guessed that you can edit the Scheme Script window to create any command you want. Even if you are not familiar with Scheme you may find that you can adapt other scripts to do what you want.

A good example of this is a script to insert a particular LilyPond directive into the score. This is all that many scripts do: it is easy to see the piece of LilyPond in the Scheme Script window, and by changing it you can create a new command.

For example, from the More menu select the command /menus/ObjectMenu/Instruments/Orchestral/RehearsalMark (the files are laid out in folders/directories in the same way as the menu system itself). Then choose Get Script from the right click menu. With this command its script is appended to the Scheme Script window. Here you can see the part that says

```
(d-DirectivePut-standalone-postfix "RehearsalMark" " "\\mark \\default" )
```

which is inserting the LilyPond directive "\\mark \\default" which inserts the default rehearsal mark. (The extra backslashes are needed to tell Scheme that you literally mean a \ sign). You can change this to insert any other LilyPond that you need - always doubling the \ signs. The you can save as a new menu item, or use Save Script to customize the script you started with.

More ambitious programmers will need to know all the commands available. Besides the complete Denemo command set (the list is given in the Command Center window) there are the following additional Scheme procedures defined.

### 14.1 Commands Getting User Input

All these commands are invoked from scheme as (d-Command args...)

- GetChar returns a string containing a single character from the user (blocks waiting for a keypress)
- GetKeypress returns a string representing a keypress from the user (blocks waiting for a keypress) (e.g. Up for the up arrow key etc.)
- GetCommand returns a string containing the command name for the user's keypress (blocks waiting for a keypress)
- GetCommandFromUser
- GetUserInput takes three strings, displays and returns the user's response as a string.
- RadioBoxMenu takes an arbitrary number of arguments, each argument is a pair, presents a menu of the first items in the pairs and returns the second item as chosen.
- RadioBoxMenuList like RadioBoxMenu but takes a list
- GetUserInput takes three arguments (title, prompt, suggested value) and returns the string typed by the user in the pop-up dialog or #f if Cancelled.

### 14.2 Commands Getting Information about Object at Cursor

All these commands are invoked from scheme as (d-COMMAND)

- GetType returns a scheme string indicating the type of the current object
- GetNoteName returns a scheme string, giving the note name a-g of the current note
- GetNote returns a scheme string, the note name, accidental and octave of the current note (in LilyPond notation)
- GetNotes returns a scheme string, the notes of a chord separated by spaces (in LilyPond notation)

### 14.3 Predicates

Testing what is true at the cursor position

- Music?
- Note?
- Rest?

- Chord?
- Singlenote?
- Directive?
- Timesignature?
- Keysignature?
- Clef?
- Tupletmarker?
- TupletOpen?
- TupletClose?
- StemDirective?
- None?
- MovementEmpty?
- MeasureEnd?
- MeasureBeginning?
- LastMovement?
- FirstMovement?

## 14.4 Iterators

The parameters are either a string (script) which will be evaluated after moving the cursor to step of the iteration or a scheme procedure (a thunk).

- (ForAllMovements script)
- (ForAllStaffs script)
- (ForAllMovementsExecute proc)
- (ForAllStaffsExecute proc)
- (ForAllObjectsInStaffExecute proc)
- (ForAllObjectsInScoreExecute proc)
- (ForAllNotesInChordExecute proc)

## 14.5 Message Bar

These commands put a message on the status bar to the right hand side. They are defined in `actions/denemo-modules/helpsystem.scm`. Help messages are pushed paired with a symbol (e.g. `(cons ('mytag "mymessage"))`) and can simply be Popped or removed using the tag.

- (Help::Push pair)
- (Help::Pop)
- (Help::RemoveTag tag) ; Remove all messages with this 'tag symbol
- (Help::ClearQueue) ; Clear the entire queue
- (Help::UpdateWriteStatus)

## 14.6 Move and Search

- (FindNextObjectAllStaffs test?)
- (PrevDirectiveOfTag tag)
- (NextDirectiveOfTag tag)
- (NextDirectiveOfTagInMeasure tag)
- (PrevDirectiveOfTagInMeasure tag)
- (GoToMeasureEnd)
- (GoToMeasureBeginning)
- (MoveToColumnStart)
- (MoveToColumnEnd)
- GetPosition ;use result with (apply d-GoToPosition position)) to go to the position gotten
- (PositionEqual? position1 position2)
- (Probe test moveinstruction)
- (ProbePosition test movement staff measure horizontalposition)
- (ProbePreviousMeasure test)
- (ProbeNextMeasure test)
- (ProbeNextObject test)
- (ProbePreviousObject test)
- (ProbeNextNote test)
- (ProbePreviousNote test)
- (MoveDownStaffOrVoice)
- (MoveUpStaffOrVoice)

## 14.7 More ...

This is a comprehensive listing of Scheme commands that are built-in but not including those in the menus. Call these via (d-HideMenus *params* ...)

- HideMenus. Hides all the menus
- HideButtons. Hides Score buttons or shows them if passed #f
- DestroyButtons. Removes Score buttons
- HideWindow. Hides the Denemo.project or shows it if passed #f
- ScriptCallback. Takes the the name of a scripted command. Runs the script stored for that command. Scripts which invoke other scripted commands use this (implicitly?)
- GetOption. create a dialog with the options & return the one chosen, of #f if the user cancels
- GetTextSelection. Returns the text on the clipboard
- GetPadding. Returns the padding that has been set by dragging in the Print view window
- GetRelativeFontSize. Deprecated - gets an integer from the user via a dialog
- InitializeScript. Takes a command name. called by a script if it requires initialization the initialization script is expected to be in init.scm in the menupath of the command passed in.
- LoadCommand. pass in a path (from below menus) to a command script. Loads the command from .denemo or system if it can be found. It is used at startup in .denemo files like ReadingNoteNames.denemo which executes (d-LoadCommand \"MainMenu/Educational/ReadingNoteNames\") to ensure that the command it needs is in the command set.

- `ActivateMenuItem`. Takes a string, a menu path (from below menus). It executes the command for that menu item. Returns `#f` for no menu item.
- `LocateDotDenemo`. Returns the directory holding the user's preferences
- `GetType`. Returns the name of the type of object at the cursor
- `GetLilyPond`. Returns the lilypond typesetting text for object at the cursor or `#f` if the object has not yet been typeset
- `GetTuplet`. Returns a string numerator/denominator for a tuplet open object or `#f` if cursor not on a tuplet open
- `SetTuplet`. Set passed string as numerator/denominator for a tuplet open at cursor
- `SetBackground`. Set passed 24 bit number as RGB color of background.
- `GetClipObjType`. Takes a staff number `m` and a object number `n`. Returns the type of object at the `(m, n)`th position on the Denemo Clipboard or `#f` if none.
- `GetClipObjects`. Takes a staff number `m`, Returns the number of objects in the `m`th staff on the Denemo Clipboard or `#f` if none.
- `PutClipObj`. Takes a staff number `m` and a object number `n`. Inserts the `(m, n)`th Denemo Object from Denemo Clipboard into the staff at the cursor position
- `ClearClipboard`. Clears the Denemo Music Clipboard
- `GetStaffsInClipboard`. Gives the number of staffs in the Denemo Music Clipboard
- `GetMeasuresInStaff`. Gives the number of measures in the current staff
- `GetStaffsInMovement`. Gives the number of staffs in the current movement
- `StaffToVoice`. Makes the current staff a voice belonging to the staff above
- `VoiceToStaff`. Makes the current voice a independent staff
- `IsVoice`. Returns `#f` if the current staff is not a voice else true
- `AdjustXes`. Adjusts the horizontal (x-) positioning of notes etc after paste
- `HighlightCursor`. Turn highlighting of cursor off/on returning `#t`, or given a boolean parameter sets the highlighting returning the previous value
- `GetNonprinting`. Returns `#t` if there is an object at the cursor which has any printing behavior it may have overridden
- `SetNonprinting`. Sets the Non Printing attribute of a chord (or note/rest) at the cursor. For a rest this makes a non printing rest, for a note it makes it ia pure rhythm (which will not print, but can be assigned pitch, e.g. via a MIDI keyboard. Pass in `#f` to unset the attribute
- `IsGrace`. Returns `#t` if there is a grace note/chord at cursor, else `#f`
- `IsTied`. Returns `#t` if there is a tied note/chord at cursor, else `#f`
- `IsSlurStart`. Returns `#t` if there is a chord with slur starting at cursor, else `#f`
- `IsSlurEnd`. Returns `#t` if there is a chord with slur ending at cursor, else `#f`
- `IsCrescStart`. Returns `#t` if there is a chord with crescendo starting at cursor, else `#f`
- `IsCrescEnd`. Returns `#t` if there is a chord with crescendo ending at cursor, else `#f`
- `IsDimStart`. Returns `#t` if there is a chord with diminuendo starting at cursor, else `#f`
- `IsDimEnd`. Returns `#t` if there is a chord with diminuendo ending at cursor, else `#f`
- `IsInSelection`. Returns `#t` if the cursor is in the selection area, else `#f`
- `HasSelection`. Returns `#t` if there is a selection, else `#f`
- `IsAppending`. Returns `#t` if the cursor is in the appending position, else `#f`
- `ShiftCursor`. Shifts the cursor up or down by the integer amount passed in
- `GetMovement`. Returns the movement number counting from 1
- `GetVoiceIdentifier`. Returns the LilyPond identifier for the current voice

- **GetStaff.** Returns the staff/voice number counting from 1
- **StaffHidden.** With parameter `#t` or `#f` makes the staff hidden/visible in the display, returns the hidden status. Typesetting is unaffected
- **GetMeasure.** Returns the measure number counting from 1
- **SetObjectDisplayWidth.** Sets the display width of the object at the cursor to the value passed (in pixels)
- **GetHorizontalPosition.** Returns the cursor horizontal position in current measure.\n 1 = first position in measure, n+1 is appending position where n is the number of objects in current measure
- **GetCursorNote.** Returns the note name for the line or space where the cursor is
- **GetCursorNoteWithOctave.** Returns the note name and octave in LilyPond notation for the line or space where the cursor is
- **DebugObject.** Prints out information about the object at the cursor
- **DisplayObject.** Displays information about the object at the cursor position.
- **GetEditingTime.** Prints out the cumulative time spent editing this score.\n The time counts any period between starting to edit and saving to disk\n The time is accumulated over different editing sessions.
- **DestroySchemeInit.** Remove the user's customized buttons and other scheme startup stuff created by the user in actions/denemo.scm
- **GetNoteName.** Returns the name of the (highest) note in any chord at the cursor position, or `#f` if none
- **InsertRest.** Insert a rest at the cursor in the prevailing duration, or if given a integer, in that duration, setting the prevailing duration. If MIDI in is active, the cursor stays on the rest after inserting it, else it moves right.
- **PutWholeMeasureRests.** Insert rests at the cursor to the value of the one whole measure in the key signature and return the number of rests inserted
- **GetNote.** Takes optional integer parameter `n = 1...`, returns LilyPond representation of the `n`th note of the chord at the cursor counting from the lowest, or `#f` if none
- **GetNoteFromTop.** Takes optional integer parameter `n = 1...`, returns LilyPond representation of the `n`th note of the chord at the cursor counting from the highest, or `#f` if none
- **GetNoteFromTopAsMidi.** Takes optional integer parameter `n = 1...`, returns MIDI key for the `n`th note of the chord at the cursor counting from the highest, or `#f` if none
- **GetNotes.** Returns a space separated string of LilyPond notes for the chord at the cursor position or `#f` if none
- **GetNoteAtCursor.** Returns LilyPond note at the cursor position or `#f` if none
- **GetDots.** Returns the number of dots on the note at the cursor, or `#f` if no note
- **GetNoteBaseDuration.** Returns the base duration of the note at the cursor number=0, 1, 2 for whole half quarter note etc, or `#f` if none
- **GetNoteDuration.** Returns the duration in LilyPond syntax of the note at the cursor, or `#f` if none
- **GetOnsetTime.** Returns start time for the object at the cursor, or `#f` if it has not been calculated
- **SetDurationInTicks.** Takes an integer, Sets the number of ticks (PPQN) for the object at the cursor, returns `#f` if none; if the object is a chord it is set undotted
- **GetRecordedMidiTempo.** Takes an index, returns the time in seconds, time signature and tempo in seconds per quarter note of the index'th MIDI tempo event in the recorded MIDI stream.
- **GetImportedMidiTrack.** Takes an track number 1,2 ..., makes that MIDI track of the loaded MIDI stream the current recorded track.
- **DeleteImportedMidi.** Delete the current imported/recorded MIDI track fails if playing, returning `#f`.
- **GetCurrentMidiTrack.** Returns the MIDI track number of the current imported track.
- **GetImportedMidiTracks.** Returns the number of MIDI tracks of the loaded/recorded MIDI.
- **GetRecordedMidiDuration.** Returns the duration in seconds of the recorded MIDI track or `#f` if none
- **GetDurationInTicks.** Returns the number of ticks (PPQN) for the object at the cursor, or `#f` if none

- `GetBaseDurationInTicks`. Returns the number of ticks (PPQN) for the chord without dots or triplet effects at the cursor, or `#f` if not a chord. The value is -ve for special durations (i.e. non-standard notes)
- `GetEndTick`. Returns the tick count (PPQN) for the end of the object at the cursor, or `#f` if none
- `GetStartTick`. Returns the tick count (PPQN) for the start of the object at the cursor, or `#f` if none
- `GetMeasureNumber`. Returns the measure number at cursor position.
- `CursorToNote`. Takes LilyPond note name string. Moves the cursor to the line or space
- `CursorToNthNoteHeight`. Takes a number 1 ... n. Moves the cursor to the nth note from the bottom of the chord at the cursor, returning `#f` if it fails.
- `CursorToNextNoteHeight`. Moves the cursor up to the next higher note of the chord at the cursor, returning `#f` if it fails.
- `GetPrevailingKeysig`. Returns the prevailing key signature at the cursor
- `GetPrevailingTimesig`. Returns the prevailing time signature at the cursor
- `GetPrevailingClef`. Returns the prevailing clef at the cursor. Note that non-builtin clefs like drum are not handled yet.
- `GetPrevailingClefAsLilyPond`. Returns the LilyPond typesetting syntax for prevailing clef at the cursor.
- `GetPrevailingKeysigAsLilyPond`. Returns the LilyPond typesetting syntax for prevailing key signature at the cursor.
- `GetPrevailingTimesigAsLilyPond`. Returns the LilyPond typesetting syntax for prevailing time signature at the cursor.
- `GetPrevailingDuration`. Returns the prevailing duration, ie duration which will be used for the next inserted note, with a parameter 0 ... 8 sets the prevailing duration.
- `IncrementInitialKeysig`. Makes the initial key signature sharper/flatter
- `IncrementKeysig`. Makes the key signature sharper/flatter, affects key signature change when cursor is on one or appending after one, otherwise affects initial key signature
- `AddMovement`. Appends a new movement without copying staff structure.
- `ChangeChordNotes`. Takes a string of LilyPond note names. Replaces the notes of the chord at the cursor with these notes, preserving other attributes
- `PutNoteName`. Takes a LilyPond note name, and changes the note at the cursor to that note
- `SetAccidental`. Takes a LilyPond note name, changes the note at the cursor to have the accidental passed in either LilyPond string or integer -2..+2. Returns `#f` if cursor is not on a note position.
- `PutRest`. Inserts a rest at the cursor; either passed in duration or if none passed the prevailing duration.
- `PutNote`. Inserts a note at the cursor; either passed in duration or if none passed the prevailing duration.
- `InsertNoteInChord`. Takes a LilyPond note name, and adds that note to the chord
- `DiatonicShift`. Moves the note at the cursor by the number of diatonic steps passed in
- `NextObject`. Moves the cursor to the right returning `#t` if this was possible
- `PrevObject`. Moves the cursor to the left returning `#t` if the cursor moved
- `NextObjectInMeasure`. Moves the cursor to the next object in the current measure, returning `#f` if there were no more objects to the left in the current measure
- `PrevObjectInMeasure`. Moves the cursor to the previous object in the current measure, returning `#f` if the cursor was on the first object
- `NextSelectedObject`. Moves the cursor to the next object in the selection. Returns `#t` if the cursor moved
- `PrevSelectedObject`. Moves the cursor to the previous object in the selection. Returns `#t` if the cursor moved
- `NextChord`. Moves the cursor to the next object of type CHORD in the current staff. Returns `#f` if the cursor did not move
- `PrevChord`. Moves the cursor to the previous object of type CHORD in the current staff. Returns `#f` if the cursor did not move
- `NextChordInMeasure`. Moves the cursor to the next object of type CHORD in the current measure. Returns `#f` if the cursor did not move



- `PrevChordInMeasure`. Moves the cursor the the previous object of type `CHORD` in the current measure. Returns `#f` if the cursor did not move
- `NextNote`. Moves the cursor the next object of type `CHORD` which is not a rest in the current staff. Returns `#f` if the cursor did not move
- `PrevNote`. Moves the cursor the previous object of type `CHORD` which is not a rest in the current staff. Returns `#f` if the cursor did not move
- `CreateSnippetFromObject`. Creates a music Snippet comprising the object at the cursor Returns `#f` if not possible, otherwise an identifier for that snippet
- `SelectSnippet`. Selects music Snippet from passed id Returns `#f` if not possible
- `InsertSnippet`. Inserts music Snippet from passed id Returns `#f` if not possible, a second boolean parameter determines if the snippet becomes selected.
- `NextStandaloneDirective`. Moves the cursor the next object that is a Denemo Directive in the current staff. Returns `#f` if the cursor did not move
- `PrevStandaloneDirective`. Moves the cursor the previous object that is a Denemo Directive in the current staff. Returns `#f` if the cursor did not move
- `NextStandaloneDirectiveInMeasure`. Moves the cursor within the current measure to the next object that is a Denemo Directive in the current staff. Returns `#f` if the cursor did not move
- `PrevStandaloneDirectiveInMeasure`. Moves the cursor within the current measure to the previous object that is a Denemo Directive in the current staff. Returns `#f` if the cursor did not move
- `Chordize`. Enforces the treatment of the note at the cursor as a chord in LilyPond
- `SetPrefs`. Takes xml representation of a preference and adds it to the Denemo preferences
- `GetBooleanPref`. Takes string name of a boolean-valued preference and returns the current value. Non-existent prefs return `#f`, ensure that the preference name is correct before using.
- `GetIntPref`. Takes string name of an int-valued preference and returns the current value. Non-existent prefs return `#f`
- `GetStringPref`. Takes string name of a string-valued preference and returns the current value. Non-existent prefs return `#f`
- `AttachQuitCallback`. Takes a script as a string, which will be stored. All the callbacks are called when the musical score is closed
- `DetachQuitCallback`. Removes a callback from the current musical score
- `GetInputSource`. Returns `DENEMO_INPUTMIDI`, `DENEMO_INPUTKEYBOARD`, `DENEMO_INPUTAUDIO` depending on the source of input to Denemo.
- `PopupMenu`. Pops up a menu given by the list of pairs in the argument. Each pair should be a label string and an expression, the expression for the chosen label is returned. Alternatively the label string can be replaced by a pair of strings, label . tooltip. The third syntax is just a list of string labels, the chosen string is returned.
- `GetTargetInfo`. Returns a list of the target type and grob (if a directive). Target is set by clicking on the typeset version of the score at a link that LilyPond has inserted.
- `GetNewTarget`. Interactively sets a target (a click on a LilyPond link in the printview window) from the user
- `GetNewPoint`. Interactively sets a point in the printview window from the user
- `GetReferencePoint`. Interactively sets a reference point (a click on a point in the printview window) from the user showing a cross hairs prompt
- `GetOffset`. Interactively gets an offset from the user in the print view window. The offset is from the last clicked object in the print view window. Returns pair of numbers x is positive to the right, y is positive upwards.
- `GetControlPoint`. Interactively sets a control point for a curve in the print view window. Takes one parameter the number 1-4 of the control point to set.
- `GetCurve`. Interactively gets a curve from the user in the print view window. Returns a list of pairs of numbers, the control points of the curve.
- `GetPositions`. Interactively gets two positions from the user in the print view window. Returns pair of pairs numbers.
- `HTTP`. Takes 4 parameters and makes http transaction with `www.denemo.org`

- **GoToPosition.** Move to given Movement, voice measure and object position. Takes 4 parameters integers starting from 1, use #f for no change. Returns #f if it fails
- **CreatePaletteButton.** Takes a palette name, label, tooltip and script
- **SetPaletteShape.** Takes a palette name, boolean, and limit
- **ShowPalettes.** Hides/Un-hides a palette. Pass a palette name (or #t to choose a palette) with second parameter #f hides the palette otherwise show.
- **SelectPalette.** Returns the current palette name. The palette status is not changed - it may be hidden. Pass a palette name to become the current palette or pass #t to choose a palette as the current palette.
- **ActivatePaletteButton.** Allows the user to type a label to activate a palette button.
- **GetUserInput.** Takes up to three strings, title, prompt and initial value. Shows these to the user and returns the user's string. Fourth parameter makes the dialog not block waiting for input
- **GetUserInputWithSnippets.** Takes up to three strings, title, prompt and initial value. Shows these to the user with a text editor for the user to return a string. Buttons are present to insert snippets which are bracketed with section characters. Fourth parameter makes the dialog not block waiting for input. Returns a pair comprising the user's text and formatted LilyPond syntax.
- **SelectFont.** Allows the user to select a font returns a string describing the font. Takes an optional title.
- **SelectColor.** Allows the user to select a color returns a list of r g b values between 0-255.\nTakes an optional title.
- **WarningDialog.** Takes a message as a string. Pops up the message for the user to take note of as a warning
- **InfoDialog.** Takes a message as a string and boolean noblock parameter. Pops up the message for the user to take note of as a informative message, blocks if noblock is #f
- **ProgressBar.** Takes a message as a string. Pops up the message inside of a pulsing progressbar
- **ProgressBarStop.** If running, Stops the ProgressBar.
- **TypesetForScript.** Typesets the score. Takes a script which will be called when Refresh is performed on the typeset window.
- **PrintTypesetPDF.** Prints from the PDF file generated by TypesetForScript.
- **GetChar.** Intercepts the next keypress and returns a string containing the character. Returns #f if keyboard interception was not possible.
- **GetKeypress.** Intercepts the next keypress and returns a string containing the name of the keypress (the shortcut name). Returns #f if keyboard interception was not possible.
- **GetCommandKeypress.** Returns the last keypress that successfully invoked a command
- **GetCommand.** Intercepts the next keypress and returns the name of the command invoked, before invoking the command. Returns #f if the keypress is not a shortcut for any command
- **GetCommandFromUser.** Intercepts the next keyboard shortcut and returns the name of the command invoked, before invoking the command. Returns #f if the keypress(es) are not a shortcut for any command
- **LockDirective.** Locks the standalone directive at the cursor so that it runs its delete action when deleted. The tag should be the name of a command that responds to the delete parameter.
- **SetDirectiveTagActionScript.** Sets an \"action script\" on the directive of the given tag
- **PutStandaloneDirective.** Inserts a Denemo Directive of the given tag, even if one already exists at the cursor, a pixel width can be passed as well
- **DirectiveChangeTag.** Changes the tag of the Denemo Directive at the cursor
- **DirectiveTextEdit-standalone.** Start a low-level edit of the standalone directive at the cursor
- **PutTextClipboard.** The passed string is placed on the system clipboard
- **GetUserName.** Asks the user for a user name which is returned
- **GetPassword.** Asks the user for a password which is returned
- **GetKeyboardState.** Returns an integer value, a set of bitfields representing the keyboard state, e.g. GDK\_SHIFT\_MASK etc
- **SetMidiThru.** Routes the MIDI in to MIDI out if it is not intercepted by d-GetMidi

- **GetRecordedMidiOnTick.** Returns the ticks of the next event on the recorded MIDI track -ve if it is a NOTEOFF or #f if none. Advances to the next note.
- **GetNoteForMidiKey.** Returns the LilyPond representation of the passed MIDI key number, using the current enharmonic set.
- **GetRecordedMidiNote.** Returns the ticks of the next event on the recorded MIDI track -ve if it is a NOTEOFF or #f if none
- **RewindRecordedMidi.** Rewinds the recorded MIDI track returns #f if no MIDI track recorded
- **GetMidi.** Intercepts a MIDI event and returns it as a 4 byte number
- **SetMidiCapture.** Takes one bool parameter - MIDI events will be captured/not captured depending on the value passed in, returns previous value.
- **TogglePlayAlong.** Switches to playalong playback. When playing or recording playback will not advance beyond the cursor position unless then mouse is moved or the next note is played in via MIDI in.
- **ToggleConduct.** Switches to mouse conducting playback. Playback will not advance beyond the cursor position unless then mouse is moved in the drawing area.
- **MidiRecord.** Starts playback and synchronously records from MIDI in. any script passed in is run at the end of the recording. The recording will play back with future play until deleted. The recording is not saved with the score - convert to notation first,
- **ComputeMidiNoteDurations.** Computes durations for recorded/imported MIDI notes based on tempo and timing of note off from previous note off or start.
- **GetMarkedMidiNote.** Gets the marked recorded midi note as LilyPond
- **GetMarkedMidiNoteSeconds.** Gets the time in seconds of marked recorded midi note or #f if none
- **AdvanceMarkedMidi.** Advances the marked recorded midi note can take an integer for number of steps to advance, or #f to clear the mark. Returns #f if no more marks.
- **InsertMarkedMidiNote.** Inserts the marked recorded or imported MIDI note using the duration guessed from the note length. Returns #f if nothing marked.
- **CreateTimebase.** Generates the MIDI timings for the music of the current movement. Returns TRUE if the MIDI was re-computed else FALSE (call was unnecessary).
- **PutMidi.** Takes and int as MIDI data and simulates a midi event, avoiding capturing of midi by scripts. Value 0 is special and is received by scripts.
- **OutputMidi.** Takes and int as MIDI data and sends it directly to the MIDI out backend
- **OutputMidiBytes.** Takes a string of space-separated bytes. The \$ char stands for the current channel. Sends the passed bytes to the MIDI out
- **PlayMidiKey.** Deprecated - takes an integer which is decomposed into a MIDI note played for 100ms
- **PendingMidi.** Takes a midi note key and plays it with next rhythm effect
- **PlayMidiNote.** Takes midi key number, volume 0-255, duration in ms and channel 0-15 and plays the note on midi out.
- **OneShotTimer.** Takes duration and executable scheme script. Executes the passed scheme code after the passed duration milliseconds
- **Timer.** Takes a duration and scheme script, starts a timer that tries to execute the script after every duration ms. It returns a timer id which must be passed back to destroy the timer
- **KillTimer.** Takes a timer id and destroys the timer
- **HasFigures.** Returns #f if the current staff has no figures (or will not print out figured bass. See d-ShowFiguredBass)
- **BassFigure.** Returns a string for the bass figure for the two MIDI keys passed in
- **SpellCheckMidiChord.** returns #t if the passed list of MIDI keys fails the pitch spellcheck
- **GetCursorNoteAsMidi.** Gets the MIDI key number for the note-position where the cursor is
- **GetNoteAsMidi.** Returns the MIDI key number for the note at the cursor, or 0 if none
- **RefreshDisplay.** Re-draws the Denemo display, which can have side effects on the data
- **SetSaved.** Sets the status of the current musical score to saved, or unsaved if passed #f

- **GetSaved.** Gets the saved status of the current musical score
- **MarkStatus.** Returns `#f` if mark is not set
- **GetHelp.** Takes a command name and returns the tooltip or `#f` if none
- **LoadKeybindings.** Takes a file name, loads keybindings from actions/menus returns `#f` if it fails
- **SaveKeybindings.** Takes a file name, saves keybindings from actions/menus returns `#f` if it fails
- **ClearKeybindings.** Clears all keybindings returns `#t`
- **LoadCommandset.** Takes a file name for xml format commandset, loads commands, returns `#f` if it fails
- **Zoom.** Takes a double or string and scales the display; return `#f` for invalid value else the value set. With no parameter returns the current value.
- **MasterTempo.** Takes a double or string and scales the tempo; returns the tempo set. With no parameter returns the current master tempo
- **MovementTempo.** Takes an integer or string number of beats (quarter notes) per minute as the tempo for the current movement; returns the tempo set
- **MasterVolume.** Takes a double or string and scales the volume; returns the volume set
- **StaffMasterVolume.** Takes a double 0-1 and sets the staff master volume for the current staff, returns the value.  
With no parameter returns the current value or zero if staff is muted.  
Pass `#f` to mute the current staff and `#t` to unmute, leaving master volume unchanged.
- **SetEnharmonicPosition.** Takes a integer sets the enharmonic range to use 0 = E-flat to G-sharp
- **GetMidiTuning.** Return a string of tuning bytes (offsets from 64) for MIDI tuning message
- **GetFlattest.** Return name of flattest degree of current temperament
- **GetSharpest.** Return name of sharpest degree of current temperament
- **GetTemperament.** Return name of current temperament
- **RewindMidi.** Rewind the MIDI generated for the current movement. Given a time in seconds it tries to rewind to there.
- **NextMidiNotes.** Takes an interval, returns a pair, a list of the next note-on events that occur within that interval and the time of these events.
- **RestartPlay.** Restart midi play, cancelling any pause
- **GetMidiOnTime.** Return a number, the midi time in seconds for the start of the object at the cursor; return `#f` if none
- **GetMidiOffTime.** Return a number, the midi time in seconds for the end of the object at the cursor; return `#f` if none
- **MidiInListening.** Set the MIDI in controller to listening mode. All signals are directed straight to the output.
- **MidiInChecking.** Set the MIDI in controller to checking mode. The cursor will advance and the note sounded only if it is the (lowest) note at the cursor.
- **MidiInAppendEdit.** Set the MIDI in controller to append/edit mode. The MIDI key sounded will be inserted in score, or appended if in appending position. MIDI signals can be filtered by scheme scripts in this mode.
- **SetPlaybackInterval.** Set start and/or end time for playback to the passed numbers/strings in seconds. Use `#t` if a value is not to be changed. Returns `#f` for bad parameters
- **AdjustPlaybackStart.** Adjust start time for playback by passed number of seconds. Returns `#f` for bad parameter
- **AdjustPlaybackEnd.** Adjust end time for playback by passed number of seconds. Returns `#f` for bad parameter
- **UserScreenshot.** Takes a parameter `#t` or `#f` and optional position: Get a screenshot from the user and append or insert it in a list (one per measure) either applying across the staves or to the current staff.
- **DeleteScreenshot.** Takes a parameter `#t` or `#f`: Delete a screenshot for the current measure, either across staves or for current staff.
- **PushClipboard.** Pushes the Denemo clipboard (cut/copy buffer) onto a stack; Use `d-PopClipboard` to retrieve it.
- **PopClipboard.** Pops the Denemo clipboard (cut/copy buffer) from a stack created by `d-PushClipboard`. Returns `#f` if nothing on stack, else `#t`.

- `DeleteSelection`. Deletes all objects in the selection Returns `#f` if no selection else `#t`.
- `SetThumbnailSelection`. Sets the selection to be used for a thumbnail. Returns `#f` if no selection or selection not in first movement else `#t`.
- `CreateThumbnail`. Creates a thumbnail for the current score. With no argument it waits for the thumbnail to complete, freezing any display. With `#t` it generates the thumbnail asynchronously. It does not report on completion.
- `Exit`. Exits Denemo without saving history, prefs etc.
- `TakeSnapshot`. Snapshots the current movement putting it in the undo queue returns `#f` if no snapshot was taken because of a guard
- `SelectDefaultLayout`. Creates the default layout.
- `CreateLayout`. Creates a custom layout from the currently selected (standard). Uses the passed name for the new layout. Returns `#f` if nothing happened.
- `DeleteLayout`. Deletes a custom layout of the passed name. Returns `#f` if no layout with passed name.
- `GetLayoutId`. Returns the id of the currently selected score layout (see View->Score Layout). Returns `#f` if no layout is selected.
- `GetCurrentStaffLayoutId`. Returns the id of a score layout for typesetting the part for the current staff. Returns `#f` if not a primary voice.
- `SelectLayoutId`. Selects the score layout with the passed id. Returns `#f` if there is no such layout.
- `LilyPondForPart`. Generates LilyPond layout for the current part (ie staves with the name of the staff with the cursor), all movements and staves with that staff name are generated.
- `TypesetPart`. Typesets the current part (ie the staff with the cursor), all movements and staves with that staff name are typeset.
- `ReduceLayoutToLilyPond`. Converts the current score layout to editable LilyPond text. After this the score layout is only affected by editing the LilyPond syntax.
- `GetLayoutName`. Returns the name of the currently selected score layout (see View->Score Layout). Returns `#f` if no layout is selected.
- `SelectNextLayout`. Selects the next score layout. If the current layout is the last, returns `#f` otherwise `#t`.
- `SelectFirstLayout`. Selects the first score layout.
- `SelectNextCustomLayout`. Selects the next custom score layout. If the current layout is the last, returns `#f` otherwise `#t`.
- `SelectFirstCustomLayout`. Selects the first custom score layout.
- `GetFilename`. Returns the full path to the currently opened Denemo score or `#f` if it does not have a disk file yet.
- `PathFromFilename`. Returns the directory component of the passed filename.
- `FileExists`. Returns the `#t` if file passed in exists.
- `FilenameFromPath`. Returns the filename component of the passed path.
- `ChooseFile`. Gives dialog to choose a file. Takes a title, start directory and list of extensions. Returns a string or `#f` if user cancels
- `OpenSource`. Follows a link to a source file of form string `"filename:x:y:page\"`. It opens the file and places a marker there.
- `EditGraphics`. Takes an optional filename and optional new name. Opens an encapsulated postscript file for editing. Returns the filename (without extension) if successful. `\n` Starts the graphics editor on the passed in filename or one from a dialog. `\n` The returned .eps file may not exist when this procedure returns, an editor is open on it. With no filename parameter allows the user to choose, `\n` copying to the project directory or the users graphics templates (if a new name is given)
- `OpenProofReadFile`. Opens a PDF file previously generated by Denemo which has proof reading annotations. The notes in the file can be clicked on to locate the music in the Denemo display
- `ExportRecordedAudio`. Converts the recorded audio to user chosen audio file.
- `OpenSourceFile`. Opens a source file for transcribing from. Links to this source file can be placed by shift-clicking on its contents
- `OpenSourceAudioFile`. Opens a source audio file for transcribing from. Returns the number of seconds of audio successfully opened or `#f` if failed.

- `CloseSourceAudio`. Closes a source audio attached to the current movement.
- `StartAudioPlay`. Plays audio allowing timings to be entered via keypresses if passed `#t` as parameter.
- `StopAudioPlay`. Stops audio playback
- `SetAudioLeadIn`. Takes a number of seconds to be used as lead-in for audio. If negative clips that much from the start of the audio.
- `AudioIsPlaying`. returns `#f` if audio is not playing else `#t`
- `NextAudioTiming`. Returns the next in the list of timings registered by the user during audio play.
- `IncreaseGuard`. Stop collecting undo information. Call `DecreaseGuard` when finished. Returns `#f` if already guarded, `#t` if this call is stopping the undo collection
- `DecreaseGuard`. Drop one guard against collecting undo information. Returns `#t` if there are no more guards \n(undo information will be collected) \nor `#f` if there are still guards in place.
- `Undo`. Undoes the actions performed by the script so far, starts another undo stage for the subsequent actions of the script. Note this command has the same name as the built-in Undo command, to override it when called from a script. Returns `#t`
- `NewWindow`. Creates a new tab. Note this command has the same name as the built-in NewWindow command, to override it when called from a script. Returns `#t`
- `StageForUndo`. Undo normally undoes all the actions performed by a script. This puts a stage at the point in a script where it is called, so that a user-invoked undo will stop at this point, continuing when a further undo is invoked. Returns `#t`
- `GetLastChange`. return a string giving the latest step available for Undo
- `GetMenuPath`. Takes a command name and returns the menu path to that command or `#f` if none
- `GetChecksum`. Takes a string and returns a string representing an MD5 checksum for the passed string.
- `SetNewbie`. Sets the newbie status to the passed value
- `GetVerse`. Gets the current verse of the current staff or `#f` if none, with an integer parameter, gets the nth verse
- `SynchronizeLyricCursor`. Moves the lyric cursor to match the current Denemo Cursor position, switching the keyboard input to the lyrics pane
- `InsertTextInVerse`. Inserts passed text at the lyric cursor in the lyrics pane, returns `#f` if no verse at cursor
- `PutVerse`. Puts the passed string as the current verse of the current staff
- `AppendToVerse`. Appends the passed string to the current verse of the current staff
- `GetId`. Takes a command name and returns and id for it or `#f` if no command of that name exists
- `AddKeybinding`. Takes a command name or command id and binding name and sets that binding on that command returns the command id that previously had the binding or `#f` if none
- `GetLabel`. Takes a command name and returns the label for the menu item that executes the command or `#f` if none
- `GetMenuPosition`. Takes a non-built-in command name and returns position in the menu system for he command or `#f` if none
- `GetLilyVersion`. Returns the installed LilyPond version
- `CheckLilyVersion`. Returns a boolean if the installed version of LilyPond is greater than or equal to the passed in version string
- `InputFilterNames`. Takes a string putting it on the scheme-controlled status bar as a list of active filters
- `WriteStatus`. Takes a string putting the scheme controlled status bar; with no argument it hides this status bar
- `Debug`. Display a debug message
- `Info`. Display an info message
- `Message`. Display a regular message
- `Warning`. Display a warning message
- `Critical`. Display a critical message
- `Error`. Display an error message and abort

## 14.8 ... and More

This is a hand-made list most of which already appear in the comprehensive list above, but with hand-written comments.

- `d-PutNoteName` takes a string argument, a note in LilyPond notation. Changes the note at the cursor. Not for use with multi-note chords.
- `d-NextObject` moves cursor to next object, returning TRUE if current object has changed
- `d-NextChord` as `d-NextObject`, but skipping non-chord objects. chords includes rests and chords with 1 or more notes.
- `d-NextNote` as `d-NextChord` but skipping rests (i.e. chords with 0 notes).
- `d-NextStandaloneDirective` as `NextObject`, stopping on a standalone directive. Using `d-DirectiveGet-standalone` tag a directive of a particular tag can be found.
- `(d-Directive-type? optional-tag)` where type is one of score, scoreheader, movementcontrol, header, paper, layout, clef, timesig, keysig, staff, voice, standalone, chord or note. This returns `#t` if the cursor is on a directive of type (with tag optional-tag if optional-tag is present) else `#f`
- `(d-DirectivePut-standalone tag)` inserts a standalone directive with the given tag at the current cursor position and places the cursor on it.
- `(d-DirectiveGetTag-type)` where type is one of score, scoreheader, movementcontrol, header, paper, layout, clef, timesig, keysig, staff, voice, standalone, chord or note. This returns the tag if the cursor is on a directive of type else `#f`
- `(d-DirectiveGetTagForTag-type tag)` returns the tag passed in if the cursor is on directive of type with that tag, else it returns the tag of the first directive of type that is present at the cursor, else `#f`
- `d-WarningDialog` Pass a string argument to pop up a warning.
- `d-GetOption` (parameter string of options). Takes a null separated set of options and pops up a dialog offering them to the user. Returns the one chosen or `#f` if the user cancels.
- `d-GetMidi`
- `d-PutMidi`
- `d-PlayMidiKey`
- `d-BassFigure`
- `d-GetNoteAsMidi`
- `d-RefreshDisplay`
- `d-InputFilterNames` sets the status bar
- `d-Chordize` Ensure that even a single note is treated as a chord - needed for some LilyPond constructs (e.g. fingerings)

Standard Denemo commands that pop up dialogs will work as usual if no argument is passed to them from Scheme. In general if a string consisting of strings of the form "name=value" is passed these will be used and no popup will occur. The field "name" will be given the value "value". The `\0` is a NULL character that separates the assignment strings. As a shorthand if there is only one value being passed you can just pass the value. The names of the fields depend on the action being called. So for example:

```
(d-Open "filename=myfile.denemo")
```

will open the file "myfile.denemo".

This feature is being rolled out, and currently works for:

- `d-InsertLilyDirective` (directive, display. minpixels) (deprecated function)
- `d-AttachLilyToChord` (prefix, postfix, display) deprecated see `d-DirectiveGet-chord-*` and `d-DirectivePut-chord-*` below
- `d-AttachLilyToNote` (prefix, postfix, display) deprecated see `d-DirectiveGet-note*` and `Put` below
- `d-StaffProperties` understands a couple of property=value settings
- `d-InitialClef`
- `d-InsertClef`
- `d-InitialKey`
- `d-InsertKey`
- `d-Open` filename
- `d-ScoreProperties` (fontsize= size of font to be used for score)

### 14.8.1 Denemo Directives

Denemo directives are things like special barlines, rehearsal marks that appear amongst the notes, as well things like trills that are attached to notes and modifications to other things in the score to give them a different appearance or behavior. If they are objects in their own right ("Standalone Directives"), like the first time bar marker then they appear between notes and the cursor can be placed on them. If they are attached to other things (such as a page break attached to a movement) then they are modified with the command used to create them.

Double clicking on Standalone Directives, gives information about the directive, while right-clicking will edit it.

## 14.9 LilyPond Editing

**Introduction** From version 0.7.8 we have the ability to edit the LilyPond output within Denemo. This approach immediately makes Denemo able to do many more things (e.g. multiple verses for songs) with the music still editable from within Denemo. The gallery of examples and the standard templates contain examples which you can use.

These can be used without knowing the LilyPond language (provided a suitable template or example file exists). Alternatively, with a general idea of how a LilyPond file works tweaks from the LilyPond documentation can be inserted into the LilyPond output and stored with the Denemo, leaving open the possibility of further editing of the notes within Denemo without the need to re-apply tweaks or keep separate LilyPond files.

### 14.9.1 Using the LilyPond Window

Under the View menu is a Show LilyPond item which pops up a window with the LilyPond output in it. The text is interspersed with buttons which enable you to hide or show the various sections or to create custom versions. The text in bold can be altered and the alterations are kept in the Denemo file.

The two windows are kept in sync, so you can move back and forth between editing textually and editing in Denemo.

Right clicking on the text gives a menu for actions on the LilyPond text. LilyPond text can be inserted between notes, and the final section (the score layout) can be turned into editable text (see Score Layout). There is one for moving the cursor to the LilyPond text for the current Denemo object. Moving the cursor in the text window with the arrow keys causes the Denemo window cursor to move in synchronism.

This menu also includes a Print command that operates on the visible LilyPond text in the window. This means you can open specific custom score layouts and print from them, or even make a temporary edit for just one print. If you save the score with custom layouts then these are remembered. So when you reload and the custom score layout will be printed.

**Detail** Several custom layouts can be kept, selecting them in the Score Layout view enables the same Denemo file to print a variety of things from the same music input.

For example, a full score or a set of parts, or several voices on group of staves or even a piano reduction.

The various Voices/Staves for the different Movements are separated by buttons. These are labelled by enumerating the movements and staves in order, so the first voice in the first movement has the music defined as "MvmntIVoiceIMusic" and so on. Within these music blocks you can insert arbitrary LilyPond text between notes, (the insertion points are marked by grey blocks) and the text inserted will appear as a LilyPond directives in the main Denemo window.

The main Denemo window also moves its cursor to correspond with where you are editing. (It should of course move the cursor immediately you click on a point in the music, but, as yet, it only moves the cursor when you press a key just before the note name - one of the arrow keys will do).

Custom layout blocks can be created by right-clicking on the Standard score layout and selecting create custom score layout. The Score Layout window allows creation of customized layouts via a GUI. If you print a single part from all movements, you get the standard scoreblock for this and can add it to your custom score block so that a single print command prints, e.g. both a full score and parts. Using the \book {} block you can put these into separate files (stored in the folder .denemo in your home folder).

The music defined by MvmntIVoiceIMusic is then used in the score blocks at the end of the LilyPond window, by the expression \MvmntIVoiceIMusic. This means that the same music can be output in several different ways, so that the same Denemo file can contain custom score blocks to output the music as a Piano Score with several voices or separate parts, for example.

The definitions for MvmntIVoiceIMusic actually look like this:

- `MvmntIVoiceIProlog = {\MvmntIVoiceITimeSig \MvmntIVoiceIKeySig \MvmntIVoiceIClef}`
- `MvmntIVoiceIMusic = {\MvmntIVoiceIProlog \MvmntIVoiceI}`

Where \MvmntIVoiceI is the actual block of notes you have written, while the other definitions hold the time signature, clef etc. By using these, you can print the same music with different clefs, still maintaining the ability to edit the notes in Denemo.

Note that the normal Denemo Print commands are still operate as they do if you never look at the LilyPond window. What happens is that if you select the Print Current Part menu item then this creates a tailored standard scoreblock for that part. However, if you use the File->Print command then it prints the first custom scoreblock that is open (visible) (or the standard scoreblock if you do not have any custom scoreblocks).

Clicking (as opposed to using the arrow keys) in the LilyPond text does not move the cursor in the Denemo window. When you delete a LilyPond directive textually you have to move the cursor to start re-inserting it.



## 14.10 More Features

### 14.10.1 Piano Stuffs, Orchestral Scores etc

A piano staff can be added using the Staff->Add Staff menu. In addition, piano staves, and staff groups such as choir staff can be created using the Staff Groupings menu under the staff menu. You can set a piano staff within a staff group by setting successively the StartPiano and StartGroup contexts on a single staff, or more generally setting StartPiano, EndPiano on adjacent staves within a staff group.

### 14.11 Single Staff Polyphony

Use the Staves/Voices->Voices menu for placing more than one voice on a staff. You can set the initial voice number from this menu (voices 1 and 3 are stem up with slurs and ties etc adjusting to suit, voices 2 and 4 are stem down). Directives can be placed in the music to change voice, see Directives->Typesetter->Voices menu.

You will also need spacer rests for voices that are silent, and commands to displace rests vertically and horizontally (see Notes/Rests->Rest Insertion menu), and commands from the Voices menu under the Staves/Voices and Directives->Typesetter menus.

### 14.12 Entering Figured Bass

To enter figures choose Notes/Rests->Markings->Figured Bass. There are some shortcuts that enable all the work to be done with the numeric keypad. A brief summary is given by right clicking the option. The conventions are described in the LilyPond docs.

### 14.13 Fret Diagrams

Fret Diagrams can be placed on the score using the command Fret Diagram in the ObjectMenu->Directives->Markings menu. In addition by assigning a Denemo staff to display as fret diagrams, chords can be entered in standard notation which will then be displayed as fret diagrams. See ObjectMenu->StaffMenu->StaffPropertiesMenu->FretDiagrams for this.

### 14.14 Tablature

Music can be displayed in tablature - the default is for standard guitar tuning but others can be set. The menus to explore are:

ObjectMenu->StaffMenu->StaffPropertiesMenu->Tablature

ObjectMenu->Directives->Typesetter->Tablature

It is possible to display the same music as both notation and tablature, and example of how to do this will be found under File->Open->Open Example.

### 14.15 Entering Chord Symbols

To enter chord symbols choose Notes/Rests->Markings->Chord Symbols. A brief summary is given by right clicking the option. The conventions are described in the LilyPond documentation.

An alternative is to set a staff to display not the chords in regular notation but the chord symbol that represents the chord. These can then be arranged above or below the melody or other representation of the piece.

Chord Charts can also be created - there is a Chord Charts palette for these.

### 14.16 Musical Scores that Do Things!

By saving a Denemo score with a script defined (in the script window) you can create music lessons, automatic midi player... the possibilities are endless. When you open such a score, the script is run - it can take user input and manipulate the score, or do other actions as your fancy takes you.

There is a special score init.denemo that is run on startup. By editing this you can startup with whatever template and whatever actions you wish to be performed. If you set it to do something that quits Denemo, you may need to delete the file before using Denemo normally again. Your local init.denemo is stored in the directory .denemo/actions in your home directory. To create it put the script you want in the script window and use SaveAs selecting ~/.denemo/actions/init.denemo as your file to save to.

#### 14.16.1 What Happens at Startup

On starting the scheme script ~/.denemo-(version number)/actions/denemo.scm is executed (where ~ means your home directory and version number is 1.1.2 or later).

A denemo file called init.denemo is/was loaded, but this is deprecated.

In addition, on startup a set of keyboard and mouse shortcuts, and a selection of optional menu items are loaded. Other sets are available via the Edit->Customize Commands ...->Manage Command Set dialog.

### 14.17 Starting Denemo - Command Line Options

Denemo -help shows the options at startup.

## Part IV

# Technical Reference - Denemo Directives

## 15 Denemo Objects

Denemo Objects are all the things that are placed in the measures of the staves in the Denemo Display. These are Chords, Notes (single note chords), Rests, Clef Changes, Key Changes, Time Signature Changes, Voice Changes (stemming control), Tuplet Starts, Tuplet Ends and Denemo Directives. The Denemo cursor can be stepped through every Denemo Object in the bar and by double clicking the object can be inspected and edited in detail.

## 16 Denemo Directives

Denemo Directives give attributes to objects that are not built-in but can be changed by the user.

Denemo Directives can be attached at almost every level of a Denemo score and can modify the behavior of the element concerned. They contain fields to describe how the element's properties should be modified, either in the display or in the printing. Elements, such as clefs notes etc have their own built-in display and print properties; Denemo Directives allow you (or scripts you invoke) to modify them for many more purposes than the built-in set allows. This means Denemo can grow - you can add features - without getting a new version.

For example the drum clef is not built-in to Denemo. Instead a directive attached to the clef has a field (graphic) set to an image of the drum clef, and another field (postfix) set to the LilyPond syntax for a drum clef, while another field (override) is set to indicate that these values should replace the normal ones, rather than adding to them.

The elements that can be modified in this way are the following:

- score: the LilyPond fields (prefix and postfix) are placed at the start of the score and just before each movement. The display field is shown at the top of the display.
- scoreheader: Attached to the score. The postfix field is put inside a `\header{}` block at the start of the score.
- movementcontrol: Attached to a movement. The prefix field is placed before the movements `\score{}` block, the postfix after it.
- header: Attached to a movement. As scoreheader but for `\header[]` blocks inside the movement's score block.
- paper: Attached to the score. The postfix is placed inside a `\paper{}` block.
- layout: Attached to a movement. The postfix is placed inside a `\layout{}` block in the movement's scoreblock.
- clef: Attached to a clef or clef change. The graphic holds the displayed icon, gx,gy its position. The postfix field is put into the music at the point where the clef is found, replacing the normal text if the override is set.
- timesig: Attached to a time signature or time signature change. The graphic holds the displayed icon, gx,gy its position. The postfix field is put into the music at the point where the time signature is found, replacing the normal text if the override is set.
- keysig: as timesig but for key signatures. (e.g. used to suppress key signatures in drum clef).
- staff: The postfix field modifies the whole staff context, with the display field printed at the start of the staff
- voice: The postfix field modifies the voice context, with the display field printed at the start of the staff containing the voice
- standalone: A directive not attached to an music element - it comes with the music and is used for things like repeat bars etc.
- chord: The prefix field is emitted before the LilyPond for the chord and the postfix after it.
- note: The prefix field is emitted before the LilyPond for the note and the postfix after it. Examples are fingerings attached to notes etc. Again the display and graphic fields are placed in the display positioned relative to the note via the coordinate fields gx, gy (for the graphic) and tx, ty for the display text.

The Directives are sufficiently important to have their own commands.

`d-DirectivePut-type-field` where type is one of score, scoreheader, movementcontrol, header, paper, layout, clef, timesig, keysig, staff, voice, standalone, chord or note and field is one of display, tx, ty, gx, gy, graphic, prefix, postfix, override, midibytes. These commands take two arguments, a tag (string) and a value to set. For example:

```
(d-DirectivePut-note-postfix "LHFinger" "3")
```

will put the fingering 3 on (after) the note at the cursor.

`d-DirectiveGet-type-field` type is one of score, scoreheader, movementcontrol, header, paper, layout, clef, timesig, keysig, staff, voice, standalone, chord or note and field is one of display, tx, ty, gx, gy, graphic, prefix, postfix, override, midibytes. This function returns the value in the field or `#f` if there is no directive with the given tag at the cursor.

`d-DirectiveDelete-type` type is one of score, scoreheader, movementcontrol, header, paper, layout, clef, timesig, keysig, staff, voice, standalone, chord or note. This function returns `#t` or `#f` if a directive with the given tag was deleted.

## 16.1 The Directive Fields

The fields of the Denemo Directive can control the Denemo Display and the LilyPond output.

The fields in d-DirectiveGet/Put have the following meanings:

- postfix - A fragment of LilyPond to be output (after the LilyPond for any object the directive is attached to).
- prefix - A fragment of LilyPond to be output (before the LilyPond for any object the directive is attached to).
- display - text to be shown in the Denemo Display
- tx,ty - where to show the text in the Denemo Display
- graphic - For directives that are in the music this is a .png image to be shown in the Denemo Display (the directory bitmaps holds these). The graphic can be saved for a command using the right-click -> Save Graphic command, after selecting a portion of the print preview as the image required. For directives attached to the score, movement etc the string set here will be displayed on the button in the button box for that sort of directive (see Show Score Titles etc in view menu for showing this button box).
- gx,gy - where to show thegraphic in the Denemo Display
- minpixels - how much space to leave for this item in the Denemo Display
- override - Contains bits to determine whether the LilyPond contained in the Directive (postfix and/or prefix fields) should override the normal LilyPond output, and whether the Graphic should replace the normal Denemo display for the item. A further tranche of bits controls MIDI output for the directive, which can override the normal MIDI interpretation of the music and provide additional information not explicit in the music notation (e.g. the tempo of an Adagio marking).

midibytes - a string of numbers (in hexadecimal format) whose interpretation is given by the MIDI bits in the override field

The override field contains the following bits:

DENEMO\_OVERRIDE\_LILYPOND: override the LilyPond output normally used at this point, rather than adding to the normal output. The text used is in the prefix and postfix fields.

DENEMO\_OVERRIDE\_GRAPHIC: overrides what Denemo would normally show in the display with the image named in the graphic field of the directive

The MIDI bits in the override field are as follows

DENEMO\_OVERRIDE\_VOLUME: the MIDI velocity to use

DENEMO\_OVERRIDE\_DURATION: affects the duration of a note. not yet implemented

DENEMO\_OVERRIDE\_REPEAT: indicates that an earlier passage should be repeated (from a directive of the same tag). not yet implemented

DENEMO\_OVERRIDE\_CHANNEL: midibytes field gives the MIDI channel to use (? implemented?)

DENEMO\_OVERRIDE\_TEMPO: midibytes field gives the tempo to use.

The interpretation of these flags is modified by the following flags:

DENEMO\_OVERRIDE\_ONCE: the value in midibytes is to be used just for the item the directive is attached to. not yet implemented

DENEMO\_OVERRIDE\_STEP: the value in midibytes is to be used from this point on

DENEMO\_OVERRIDE\_RAMP: the value in midibytes is to be used as a starting value, a corresponding directive (i.e. with the same tag) gives the final value, values are then interpolated between these. not yet implemented

DENEMO\_OVERRIDE\_RELATIVE: value in midibytes is used relative to the current value (otherwise it is an absolute value; e.g. an absolute velocity etc)

DENEMO\_OVERRIDE\_PERCENT: value in midibytes is interpreted as percentage value. not yet implemented

These flags are combined together to get the combination required for the directive using the scheme procedure called logior.

Here is an example, making a step-change in volume of 0x40 in the MIDI output, and printing "più mosso" in the output score

```
;;;;;;;;;; piu mosso
(d-DirectivePut-standalone-minpixels "StepTempo" 20)
(d-CursorLeft)
(d-DirectivePut-standalone-override "StepTempo" (logior DENEMO_OVERRIDE_TEMPO DENEMO_OVERRIDE_STEP DENEMO_OVERRIDE_REPEAT))
(d-DirectivePut-standalone-midibytes "StepTempo" "40")
(d-DirectivePut-standalone-display "StepTempo" "piu mosso")
(d-DirectivePut-standalone-postfix "StepTempo" "~\\markup {\\bold \\italic \"più mosso\"}")
(d-RefreshDisplay)
```

In this example the directive is a standalone directive. MIDI commands are being extended to apply to chords and notes. Implementation of the commands for Voices, Movements, Score etc are not yet done.

## 16.2 Directive Edit Scripts

**Introduction** Score and movement directives can define a value for their graphic - a button then appears at the top of the score which can be used to edit the directive. Likewise Staff and Voice directives show as a properties icon before the staff to which they apply (staff directives above, voice directives below). Clicking on these lets you edit the directive.

The command `EditDirective` can be used when the cursor is on a Denemo Directive object, or an object with a Denemo Directive attached to it. What happens then is determined by a script named after the "tag", or name of the directive. For example the command `RehearsalMark` creates a Directive with tag "RehearsalMark" and `EditDirective` runs a script called `RehearsalMark.scm`.

There is also a low-level editing dialog which is invoked if no editing script exists or directly from scheme using (`d-DirectiveTextEdit-<tagname>`). This allows you to edit and delete a directive directly. The other directive editing commands are `EditScoreDirective` `EditMovementDirective` `EditStaffDirective` `EditVoiceDirective` `EditClefDirective` `EditKeysigDirective` `EditTimesigDirective` for directives attached to the relevant objects.

The low level edit of directives from scheme uses the following command:

```
(d-DirectiveTextEdit-<field> <tagname>)
```

Where `<tagname>` specifies the directive to be edited and `<field>` is one of `score`, `scoreheader`, `movementcontrol`, `header`, `paper`, `layout`, `clef`, `timesig`, `keysig`, `staff`, `voice`, `standalone`, `chord` or `note`.

For example,

```
(d-DirectiveTextEdit-paper "PrintAllHeaders")
```

gives low-level access to the directive setting the print all headers command in the paper block of the LilyPond output.

### 16.2.1 Initialization Scripts

Each menu can have an initialization script, which can contain procedures that would be too time consuming to define every time they are needed. These scripts are guaranteed to be run before any menu item is activated within the menu. They can be read/written using the right click menu on any scripted menu item.

### 16.2.2 Edit Scripts

Each directive has a tag field, so that it can be recognized by the scripts that manipulate it. In particular for each tag there can be an edit script, for editing a directive of that tag.

Edit scripts are kept in a directory `actions/editscripts` parallel with the directory `actions/menus` where the commands themselves are kept.

You can read and write edit scripts by using the low level editing dialog on a directive with the tag you are writing for.

The low level editing dialog is the one that is presented if there is no edit script. Edit scripts can themselves give access to this dialog by including an option `cue-Advanced` which calls (`d-DirectiveTextEdit-field tag`) for the appropriate field and tag. For an example see the edit script for Instruments -> Orchestra -> RehearsalMark.

They have available functions to enable editing of directives which are defined in `actions//denemo.scm` executed at startup. The functions and variables for script editing are:

- `d-GetOption` followed by a nul separated list of options, offers the options to the user and returns one, or `#f` if the user cancels.
- `Extra-Offset` tag type context. Takes three string options: `tag` is the name of the directive to be edited, which must be the name of a LilyPond object, `type` is the type of directive (note, chord, standalone, staff, voice or score), `context` is the LilyPond context of the object. Only `tag` is required. Shifts the object in the LilyPond output.
- `SetPadding` tag type context. As `Extra-Offset`, it sets the space left around the item in the LilyPond engraving process.
- `SetRelativeFontSize` tag type context. As `Extra-Offset`, it shifts the font size of the following text in the LilyPond engraving process.
- `CreateButton` tag label this is just a convenience function to put a button with the passed in label onto a button box at the top of the screen. Scripts can attach actions to such tagged buttons.
- `d-SetDirectiveTagActionScript` tag scheme-actions. This command sets scheme-actions as the actions to be performed when the button of the given tag is clicked. The default action is to run any editscript associated with tag, and if none to run the `d-DirectiveTextEdit-score` on the directive that displays the button.
- `d-DirectiveGetForTag-field tag`. Useful variables defined:

- (define stop "\0")
- (define cue-Advanced "Advanced")
- (define cue-PlaceAbove "Place above staff")
- (define cue-PlaceBelow "Place below staff")
- (define cue-SetRelativeFontSize "Set Relative Font Size")
- (define cue-OffsetPositionAll "Offset Position (All)")
- (define cue-OffsetPositionOne "Offset Position (One)")
- (define cue-EditText "Edit Text")
- (define cue-SetPadding "Set Padding")
- (define cue-Delete "Delete")

## Part V

# Obtaining and Installing Denemo



Denemo is available from the Downloads page of the Denemo website <http://denemo.org>, where more up-to-date information will be found.

Denemo is available from a variety of sources for different distributions. The latest stable release (tar.gz and .deb formats) is available for download from <http://denemo.org/downloads-page/>. You can install Denemo from the Debian unstable repositories, using the command `apt-get install denemo`. Builds for Macintosh are available from the Gnu-Darwin project. The Denemo development branch can be downloaded using anonymous CVS or Git.

Anonymous Git checkout:

```
git clone git://git.savannah.gnu.org/denemo.git
```

Anonymous CVS checkout:

```
cvs -d:pserver:anonymous@pserver.git.sv.gnu.org:/denemo.git co -d denemo master
```

**Dependencies** To build Denemo from a source package, please see the website for an up-to-date list of dependencies. Remember to install the development packages as well (check your distribution for the specific package name):

For Debian Users: Type `apt-cache showsrc denemo` at a command line to determine what are the package names for Denemo dependencies. Type `apt-get build-dep denemo` to build the dependent files or use `apt-get` to install the packages individually by copying and pasting their names to the command line.

**Installing Denemo from Source Code** Denemo is available in a variety of formats. The current stable release is available either as source code or in binary format. The Development branch of Denemo is available as a GIT source tree.

### .1 To install from source code:

1. Open a terminal window.
2. Change directory to the directory to which you downloaded the Denemo source package.
3. Uncompress the source package using standard Linux tools (tar and gunzip).
4. Change directory to the uncompressed source directory.

## **.2 Generating a Configure Script**

Git does not come with a Configure script; generate one by typing and pressing Enter after the line:

```
./autogen.sh
```

To build from source, press Enter after each line:

```
./configure  
make  
make install
```

If you are not logged in as root user, for the last step type su and your root password, or alternately type sudo make install.