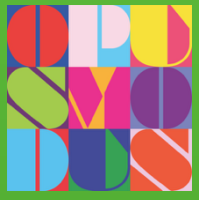
Opusmodus

Music Composition Development Environment

Manual



Janusz Podrazik

Contents

1 Opusmodus Installation

This section covers installation, first launch, and the user folder created by Opusmodus.

1.1 Install Opusmodus (macOS)

1. **Download** the archive **Opusmodus.zip** from the official site.
2. **Extract** the ZIP (Safari may unzip automatically). You should obtain *Opusmodus.app*.
3. **Move** *Opusmodus.app* to the **Applications** folder.
4. **Launch** Opusmodus from Applications or via Spotlight. On first run, if Gatekeeper warns that the app was downloaded from the Internet, use **Control-click → Open** and confirm. Grant requested permissions (e.g., file access) when prompted.

1.2 Install Opusmodus (Windows)

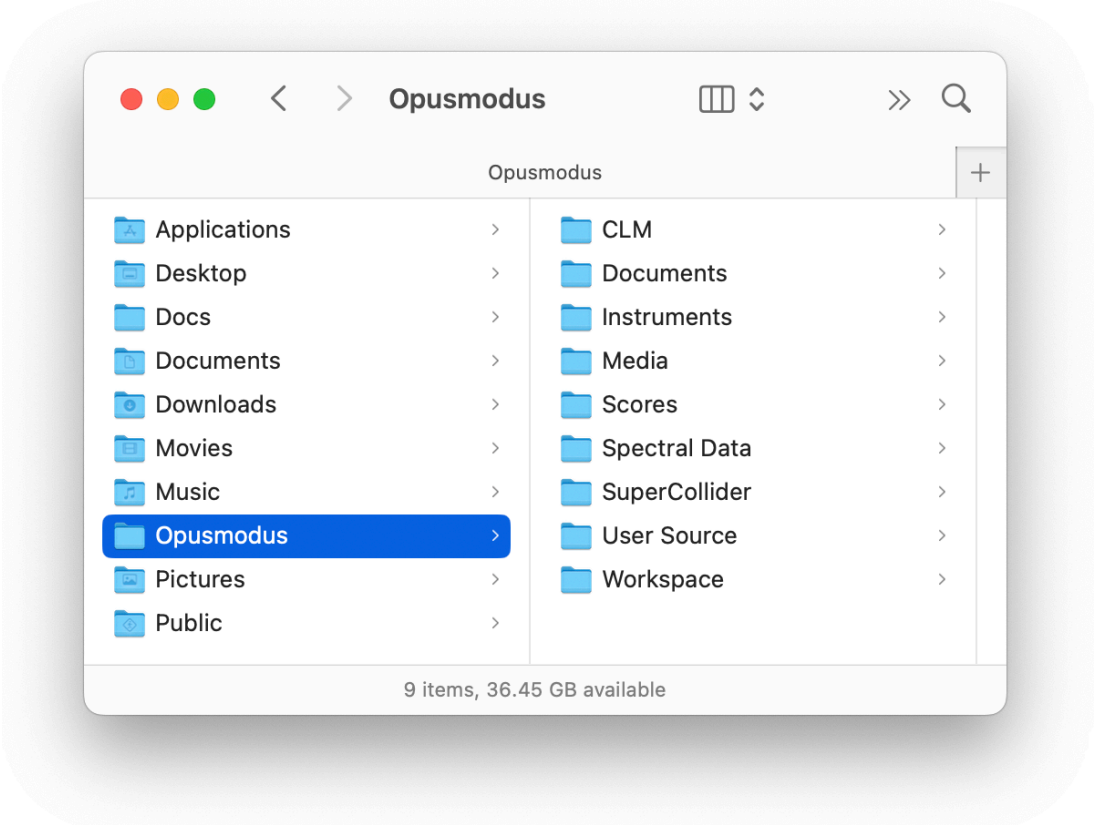
1. **Download** the Windows installer **Opusmodus.exe** from the official site.
2. **Run** the installer and follow the setup wizard. Choose the destination folder (home directory), and allow creation of Start-menu and desktop shortcuts if desired.
3. If Windows **SmartScreen** displays a warning, choose **More info → Run anyway** (verify the publisher is Opusmodus Ltd.).
4. **Launch** Opusmodus from the Start menu. Grant requested permissions when prompted.

1.3 First launch and user folder

On first launch Opusmodus creates the folder `~/Opusmodus` in your home directory and opens the **Quick Start** workspace together with the **Assistant** (documentation and tutorials).

Typical subfolders. *CLM*, *Documents*, *Instruments*, *Media*, *Scores*, *Spectral Data*, *SuperCollider*, *User Source*, *Workspace*.

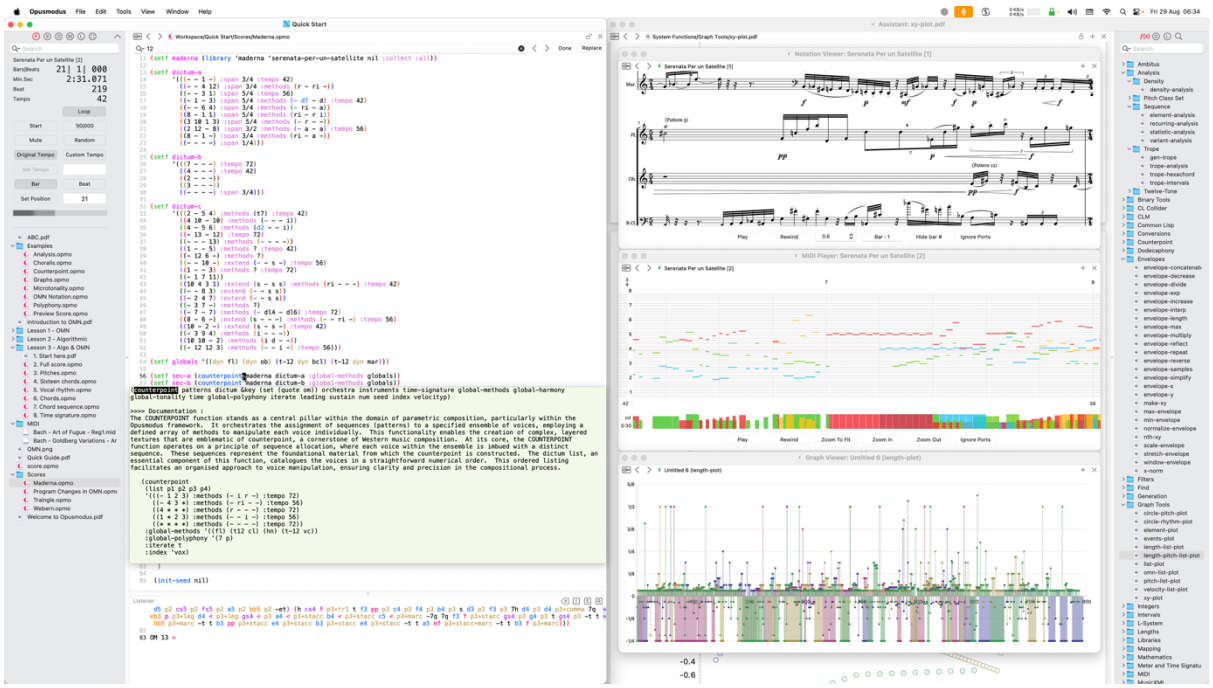
Workspace default. Save new workspaces under `~/Opusmodus/Workspace`. Choose **File → New → Workspace…** Opusmodus **automatically creates** both the **workspace folder** and a **workspace file** of the **same name** (the file can be used to reopen the workspace).



You may create additional folders inside `~/Opusmodus` as required. Do not move or rename the top-level **Opusmodus** folder.

2 Workspace

The principal panels are the **Composer**, **Listener**, **Assistant**, the **Notation**, **MIDI**, and **Graph** viewers, and the **Live Coding Instrument**. Together these provide an efficient environment for code-based composition, audition, and visualisation.



How a composer uses the interface depends on project needs and preferred workflows. Opusmodus is designed to accommodate diverse approaches, from exploratory sketching to structured, library-driven work.

2.1 Global window controls (all panels)

Lock — enables **write-protect** mode for the current document. You may **edit** the buffer and **evaluate/play back** as normal, but **Save** (and any auto-save) will **not overwrite** the backing file. Use **Save As…** to write a new copy.

Unlock — restores normal saving; **Save** writes to the backing file. *(Where applicable: Composer, Assistant items that are editable, and any viewer with an editable backing file.)*

+ — open a **new window** initialised with the **currently displayed item** (score/snippet/MIDI/plot/document). Each window operates independently (transport, navigation, and zoom are per-window).

× — close the **current display** (the top file shown in that window). The underlying file is not deleted and remains accessible via **Last Score**, **recents** (**grid menu**), or the **Navigator**.

Notes

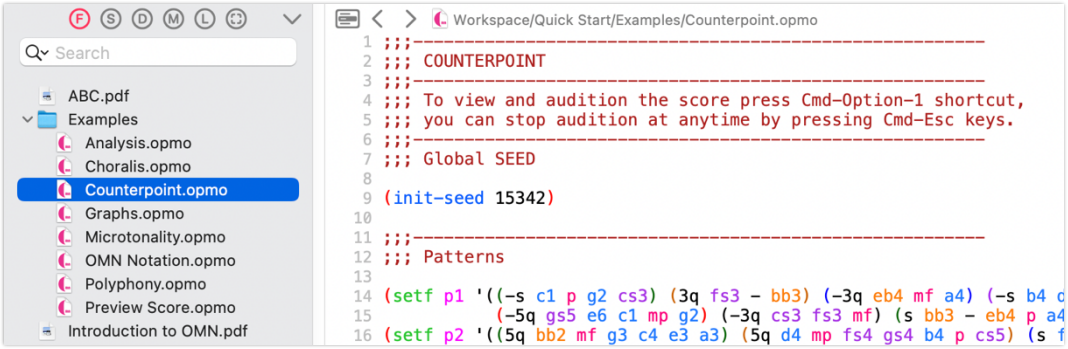
Lock is per window/document and is especially useful when viewing templates, examples, or library files to prevent accidental modification. On close, you will be prompted to **Discard** changes or **Save As…**.

2.2 Workspace Navigator

The Workspace Navigator (on the left) mirrors the contents of the active workspace on disk. All folders and files within the workspace are displayed for quick access to examples, libraries, documents, and your sources. Click an item to open it in the appropriate panel (**Composer**, **Assistant**, **Notation/MIDI**, or **Graph** viewer).

2.2.1 Navigator buttons

- **Finder** — hierarchical view of the active workspace on disk.
- **Scores** — hierarchical view of user score files in `~/Opusmodus/Scores`.
- **Documents** — hierarchical view of user documents in `~/Opusmodus/Documents`.
- **Media** — hierarchical view of user media in `~/Opusmodus/Media` (e.g., MusicXML, MIDI, Audio, Video).
- **Library** — hierarchical view of user libraries and extensions in `~/Opusmodus/User Source`. Use this folder to extend the system with your own functions, instruments, and utilities (e.g., Common Lisp .lisp files). Its contents are loaded automatically at application start into the running Lisp image and are available in the **Composer**, **Assistant**, and tools.
- **Definition** — links to the score script currently active in the **Composer** and enables lookup of the definition of a selected expression.
- **Search** — searches the contents of the active workspace (Finder).
- **Live Coding** — show or hide the **Live Coding Instrument** panel (chevron toggle).



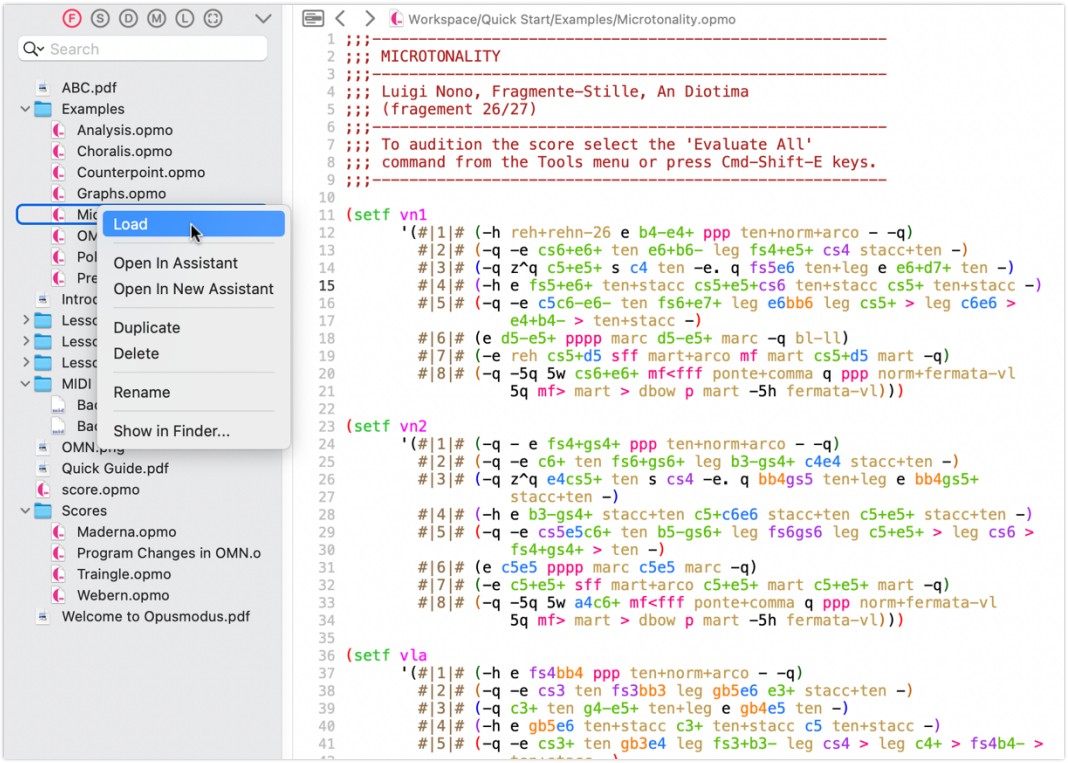
2.2.2 Navigator contextual menu

Provides file operations and alternate open targets for items in the **Workspace Navigator**. Use when you wish to open a file in a specific panel or manage files without leaving Opusmodus.

Commands

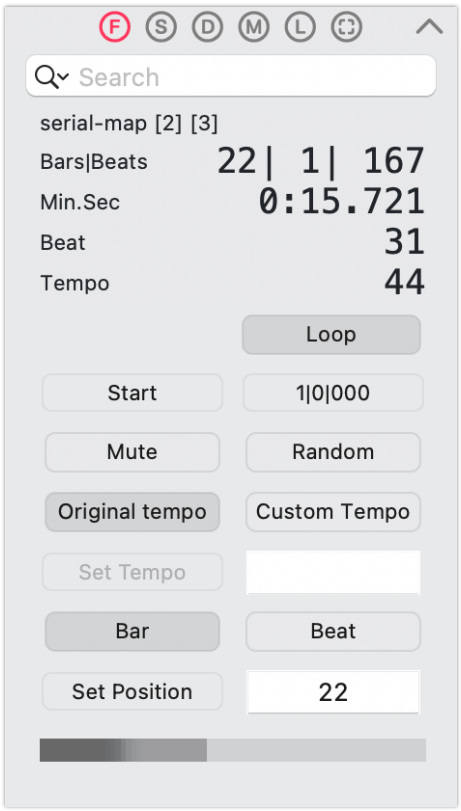
- **Load** — load the selected file into the running Opusmodus image without opening it (useful for libraries/utilities).
- **Open In Assistant** — display the item in the current Assistant window.
- **Open In New Assistant** — display the item in a new Assistant window.
- **Duplicate** — create a copy of the selected item in the same folder.
- **Delete** — remove the item (moves to the system Trash/Recycle Bin).
- **Rename** — rename the item in place.
- **Show in Finder** — reveal the item in the operating system's file manager.

Tip: a single click on an item opens it in the default panel (Composer/Assistant/Viewer). Use the contextual menu when you require a different destination or a file operation.



2.3 Live Coding Instrument (LCI)

The **Live Coding Instrument** (**LCI**) provides a real-time control surface for interactive performance with an evaluated score. It supports incremental changes from the panel while remaining in synchrony with the running music. Open the panel. Use **Tools → Live Coding** (or the **Live Coding** toggle in the Workspace Navigator).



The **LCI** operates on the current score (via **Evaluate Score → Live Coding**) or the last score (via **Last Score → Live Coding**).

2.3.1 Displays

- **Bars|Beats** — bar, beat, and tick counters.
- **Min.Sec** — elapsed clock time.
- **Beat** — quarter-beat indicator; coincides with **Bars|Beats**.
- **Tempo** — current tempo display.
- **Loop** — loop state (enabled/disabled).

2.3.2 Transport and muting

- **Start** — begin/continue playback from the current position.
- **1|0|000** — reset counters and restart at **Bar** 1.
- **Mute** — silence output without stopping transport (useful for drop-ins).

2.3.3 Randomisation

- **Random** — select bars or beats at random for controlled variation while transport continues.

2.3.4 Tempo control

- **Original tempo** — restore the tempo defined by the score.
- **Custom Tempo** — user tempo control.
- **Set Tempo** — commit the queued tempo.
- **Tempo Numeral** — set the target tempo value.

This separation allows preparation of a new tempo without disturbing the current one; use **Set Tempo** to apply at a musically appropriate moment.

2.3.5 Synchronisation

Controls and the horizontal (grey) sync indicator work together to align actions:

- **Sync to Bar / Sync to Beat** — schedule the next change for the start of the next **Bar** or **Beat**.
- **Set Tempo / Numeral coupling** — mirrors the tempo workflow above: choose the target (numeral), then apply (set tempo) at the selected sync boundary.

2.3.6 Typical workflow

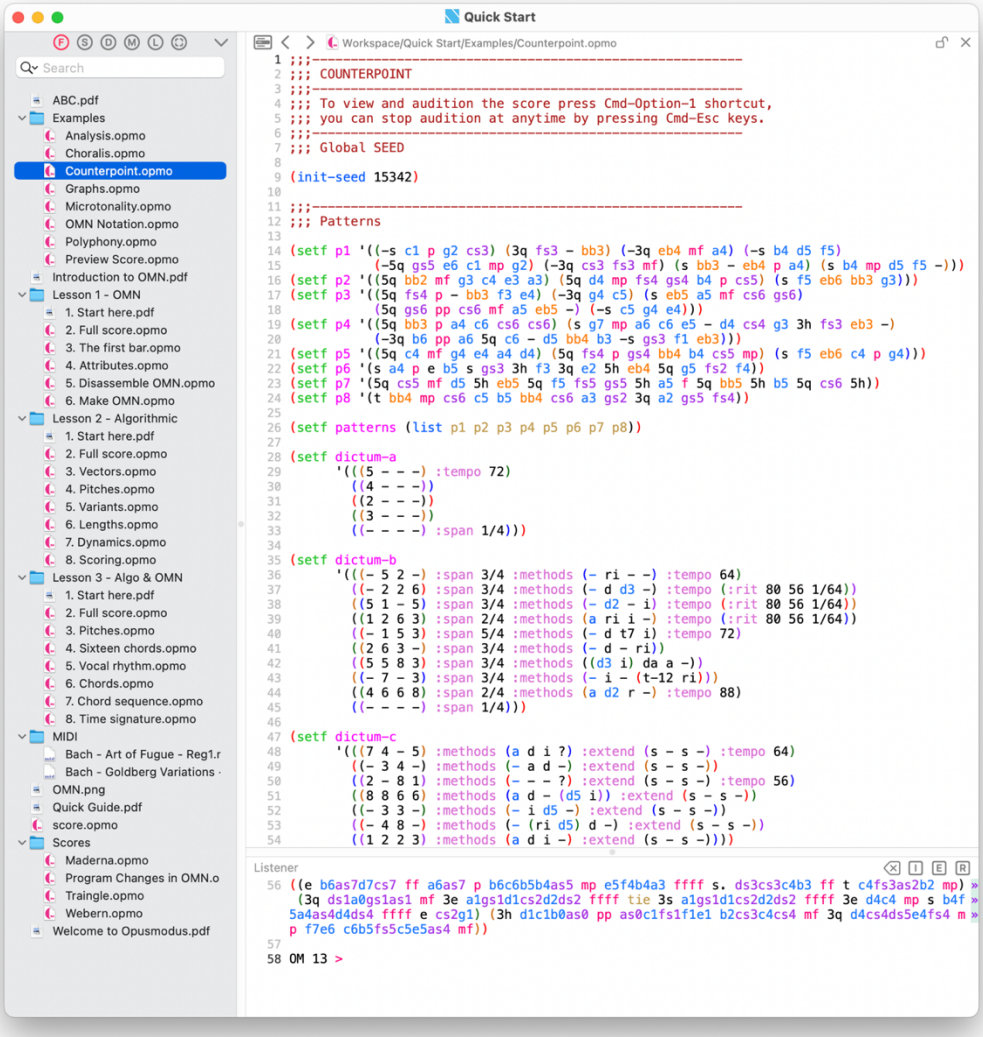
1. Evaluate the score and open **Live Coding**.
2. Let playback run; observe **Bars|Beats** and **Min.Sec**.
3. Arm **Custom Tempo**, set **Tempo Numeral**, choose sync to **Bar** or sync to **Beat**, then press **Set Tempo**.
4. Use **Mute** for silent passages and **Random** for stochastic variation.
5. Use **1|0|000** reset to return to **Bar** 1.

2.3.7 Multiple workspaces and concurrent LCIs

- Opusmodus can keep several workspaces open simultaneously. Each workspace provides its own **LCI**, bound to that workspace's **Composer/Listener** context and current/last score.
- **Independence.** Editing and evaluation are workspace-local: evaluating code in one workspace updates only that workspace's **LCI** and views.
- **Real-time control.** Transport, counters, loop state, and tempo are **per-LCI**; actions in one **LCI** do not affect the others.
- **Continuous revision.** Score code in each workspace can be continually edited and re-evaluated while performance proceeds.
- **Practical note.** For simultaneous playback across **LCIs**, assign distinct MIDI ports/channels or audio outputs to avoid contention.

3 Composer

The **Composer** (Editor) is the source editor in which you write Common Lisp/**OMN** code that defines materials and scores. From here you can evaluate expressions, audition results, and open notation or analytical views. Evaluations send output and diagnostics to the **Listener**.



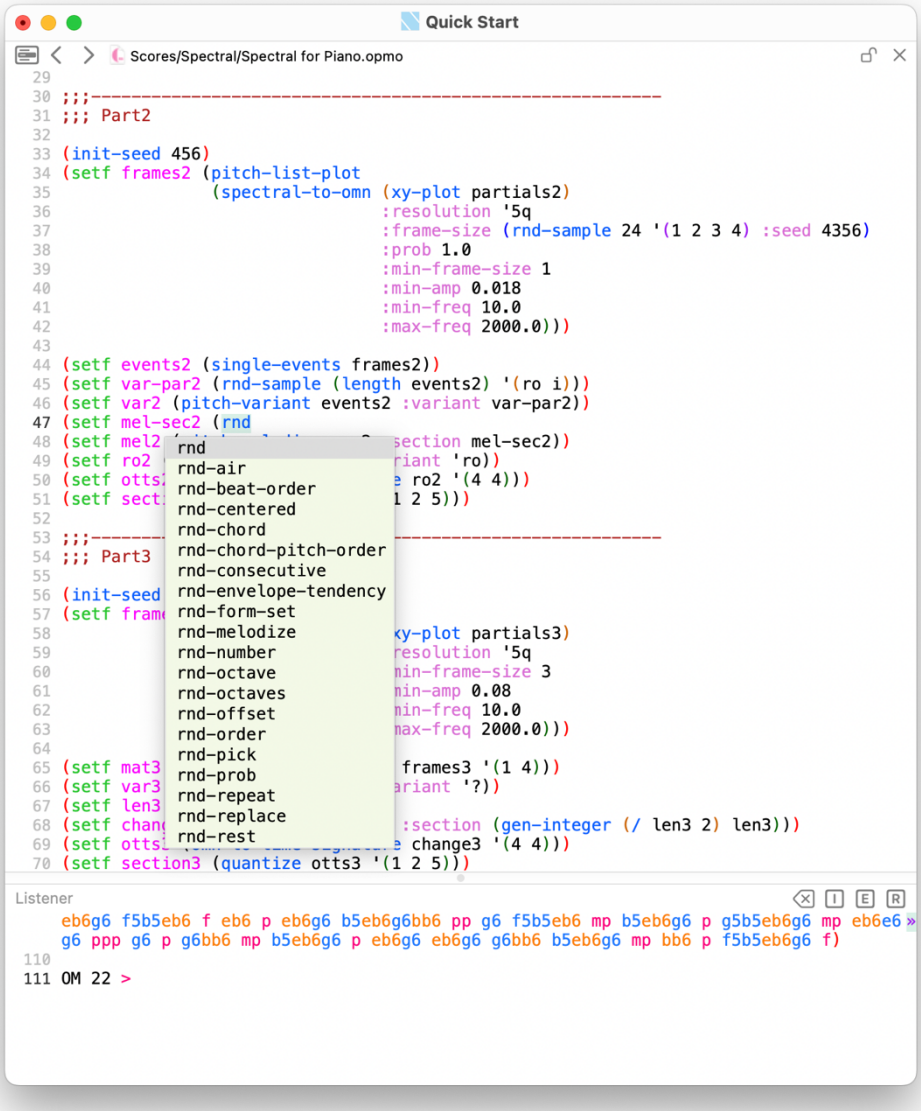
Writing in the **Composer** resembles working on paper in that early ideas can be rough and iterative. The difference is that results are immediately testable, audible, notated, or plotted—while the **Workspace Navigator** keeps related files organised.

3.1 Editing tools in the Composer

Users can employ several built-in aids, each activated by a simple command.

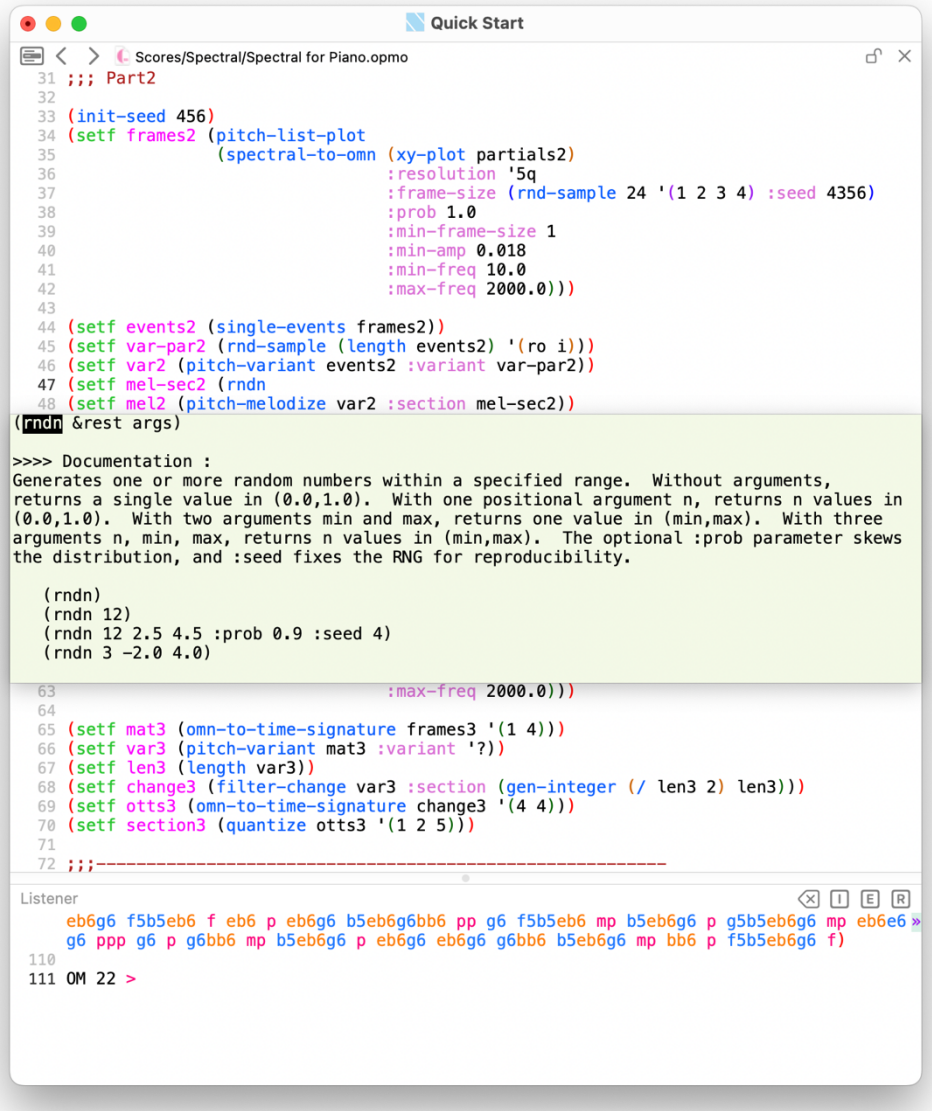
3.1.1 Autocomplete

Type an opening parenthesis and the first letter of a symbol, then press **Tab** to invoke the completion menu. Use the arrow keys to navigate; Return inserts the selection.



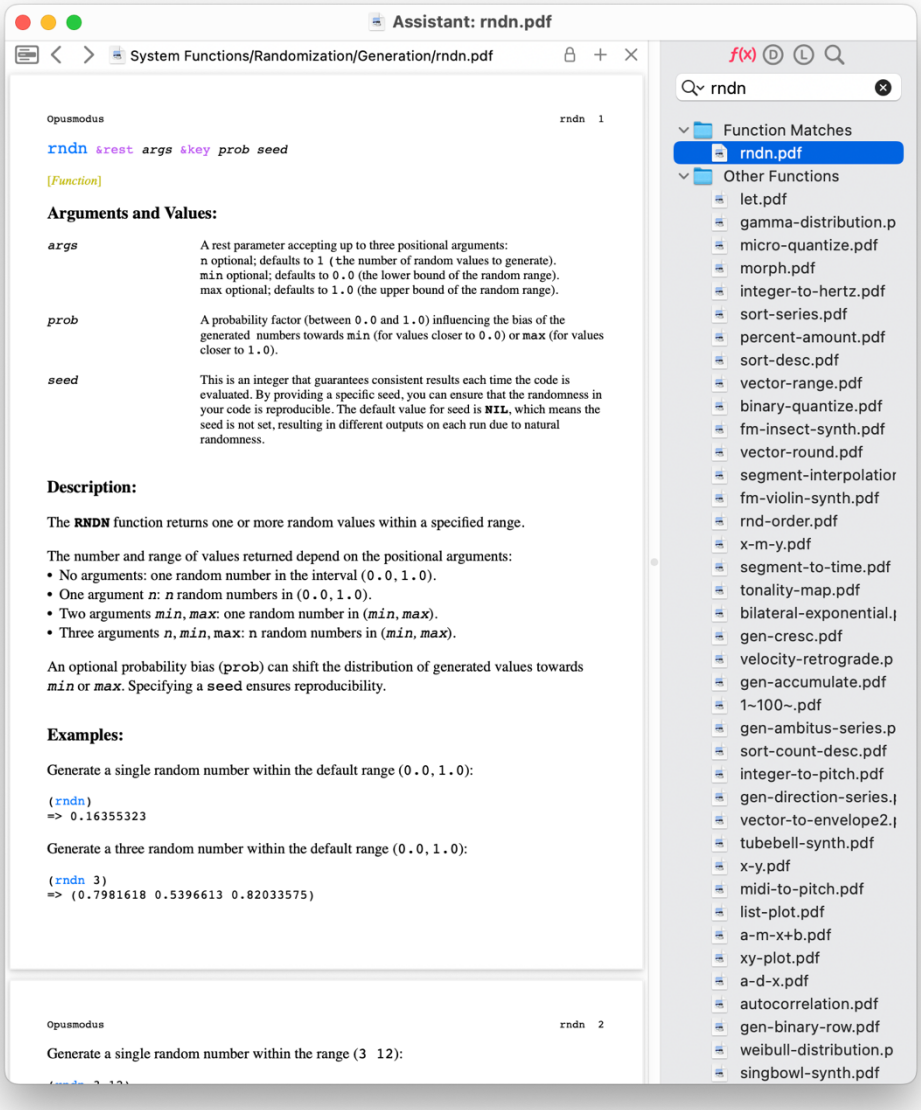
3.1.2 Inline documentation

Press **Ctrl-Y** to display a brief documentation pane for the function under the cursor.



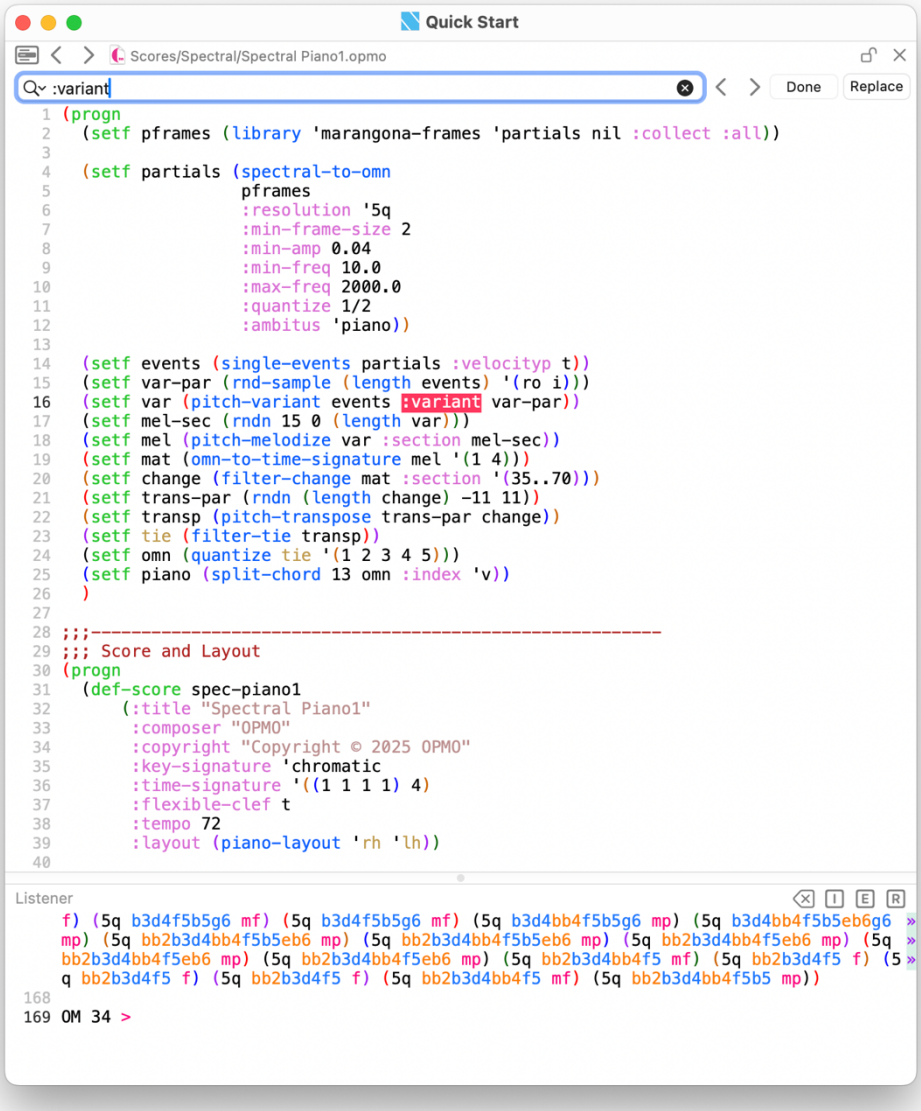
3.1.3 Full documentation

With the cursor on a function name, press **Cmd-D** to open the full function documentation in the **Assistant** panel.



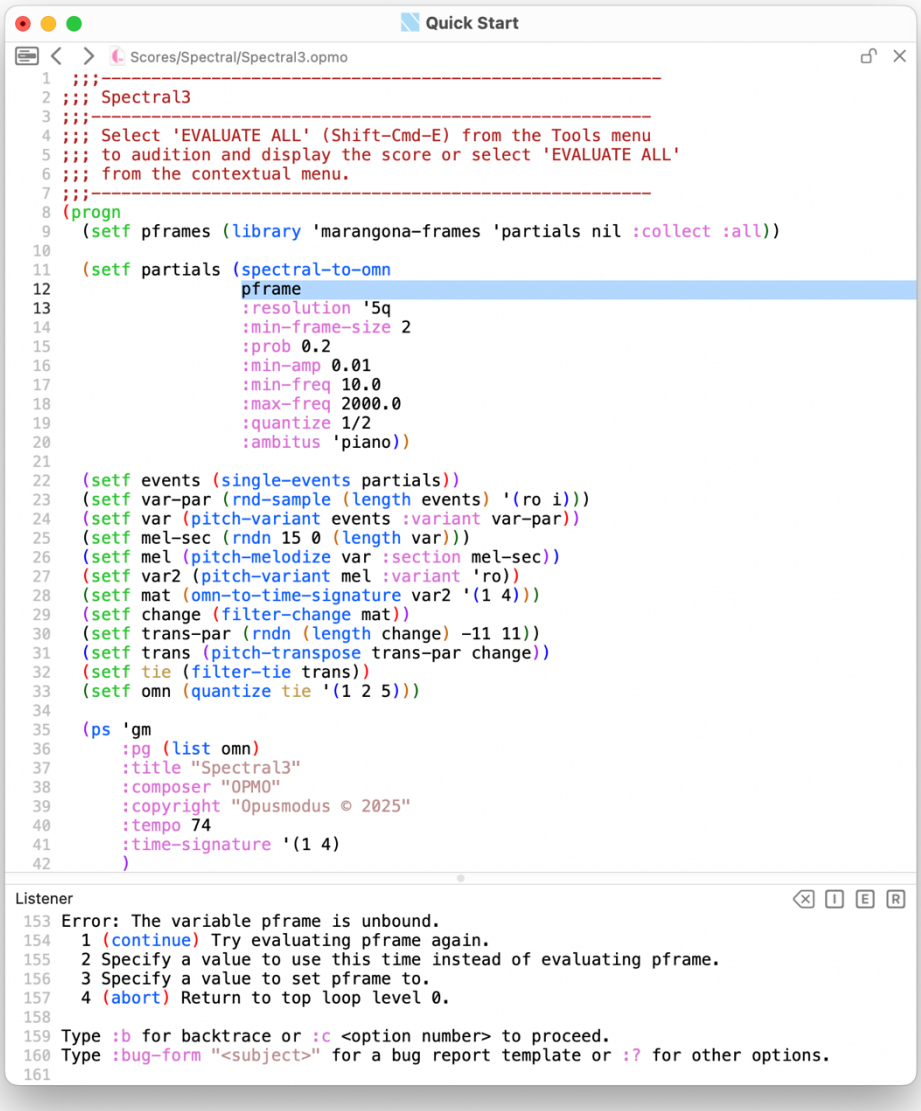
3.1.4 Find

Press **Cmd-F** to open the **search bar** at the top of the **Composer**.



3.1.5 Error highlighting

If evaluation produces an error (Evaluate All, Defun, or Region), the **Listener** reports the condition and Opusmodus highlights the error location in the **Composer** for rapid correction.

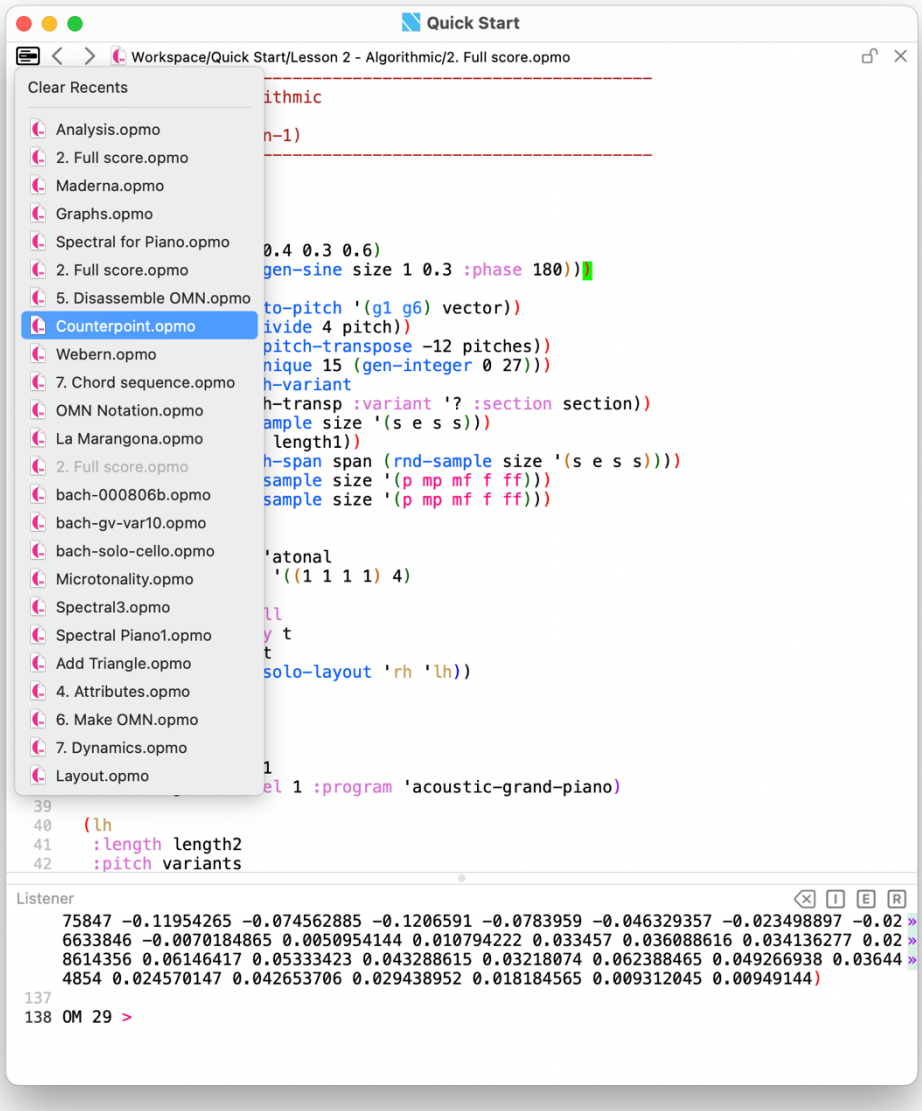


3.1.6 Contextual menu

A shortcut to the principal **Tools** commands at the point of editing. Open it with **right-click** (or **Control-click**) in the **Composer**.

3.1.7 Files Grid menu

Rapid navigation among **recently opened source files** without leaving the current Composer window.

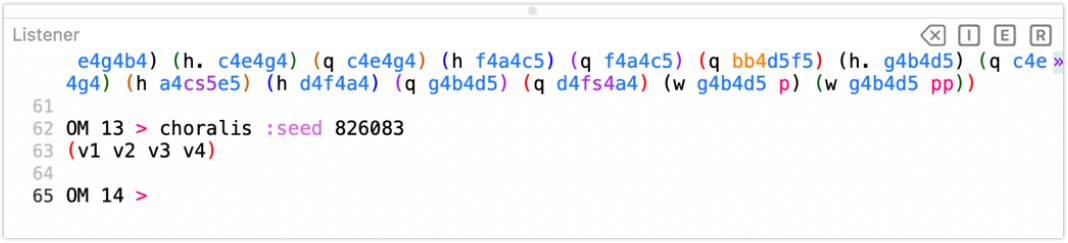


Behaviour

- The menu lists **recent files** (most recent at the top).
- **Select** an item to open it in the **Composer**.
- **Clear Recents** removes the list for the current window/session.

4 Listener

The **Listener** is the interactive REPL and message console. It receives all evaluation output (values, diagnostics, and logs), and accepts expressions for immediate execution in the **same Lisp image** as the **Composer**.



Relationship to other panels

- **Composer → Listener.** Evaluating a defun/region/buffer prints results and messages to the **Listener**. On error, the Listener reports the condition while the **Composer highlights the error location** for rapid correction (see *Settings → Debugger* for unwind behaviour).
- **Assistant → Listener.** When the Assistant displays source/score, it uses its **own Listener** pane for output; both share the same image.

What appears in the Listener

- **Values and summaries** of evaluated expressions (including OMN, integers, intervals etc.).
- **Score-related prints** from **Evaluate Score** and **Last Score** commands (e.g., **PPrint Score**, **PPrint Instruments**).
- **Pretty-printed structures** from **Tools → PPrint Expression**.
- **Warnings and errors**, including debugger entries and stack information when enabled.

4.1 Listener — window controls

These controls mirror **Tools → Listener** and act on the running REPL.

- **× (Clear)** — clear the Listener's visible contents. This does **not** reset the Lisp image or variables; it only empties the console buffer.
- **I (Interrupt)** — break the current evaluation and enter the debugger. Use when a computation runs longer than intended.
- **E (Exit Debug Level)** — unwind one debugger level and resume at the previous continuation.
- **R (Return to Top Level)** — abort all nested debug levels and return to the REPL prompt.

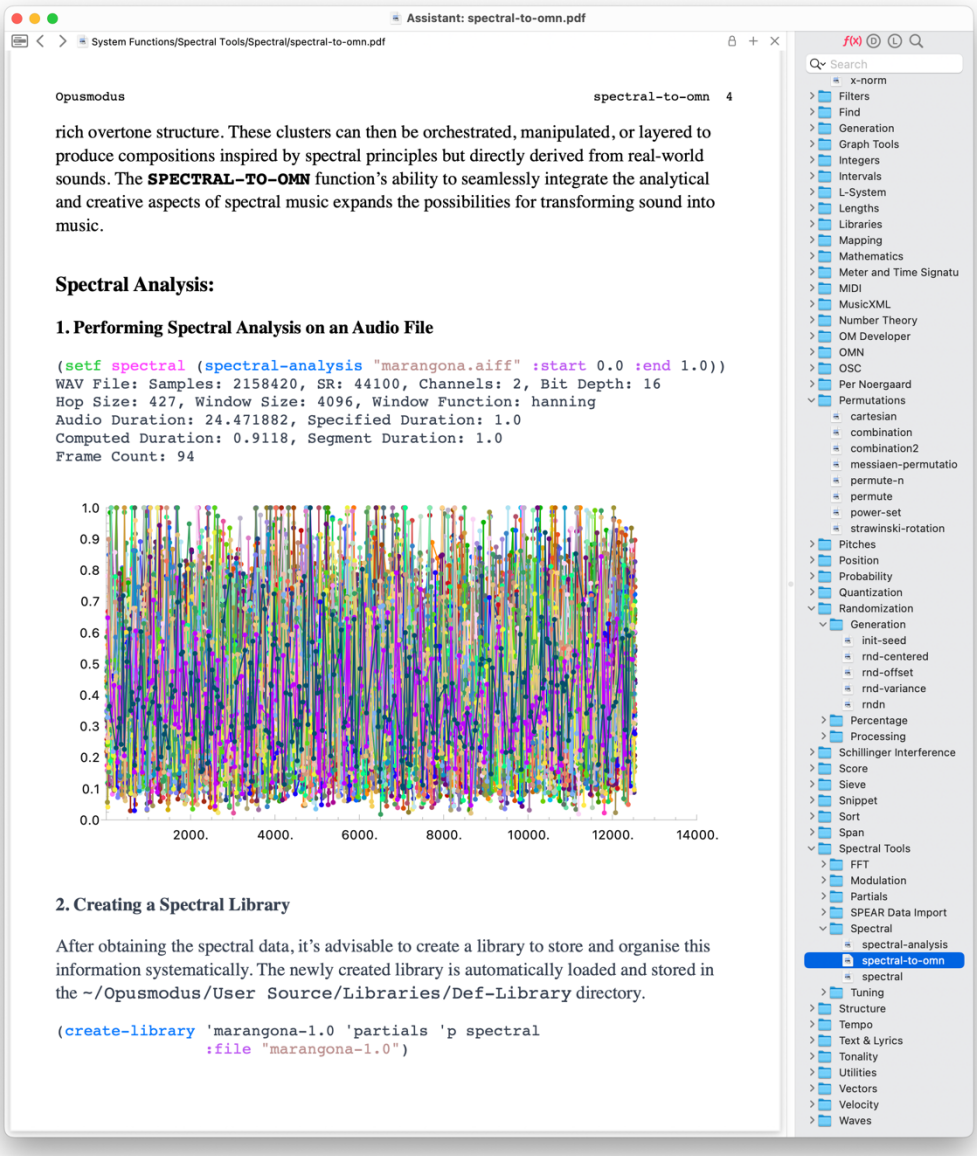
Notes

I/E/R are meaningful when a break/debugger is active (e.g., after an error or explicit interrupt).

Clearing the Listener does not affect **Last Score**, **open viewers**, or **workspace** state.

5 Assistant

The **Assistant** is the integrated documentation, examples, and library browser. Use the search field or the topic index to locate function documentation, score examples, documents, and installed libraries. Clicking an item opens it in the **Assistant**.

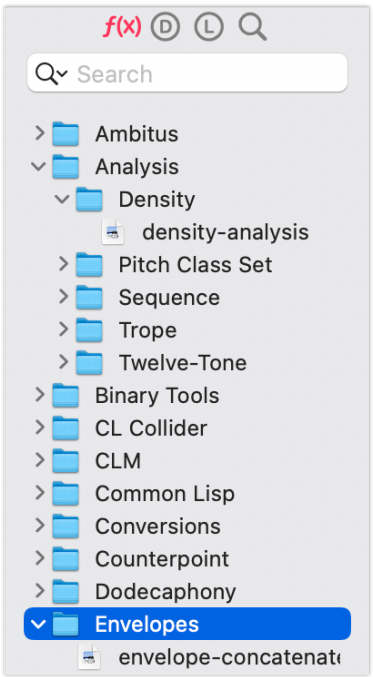


5.1 Assistant Navigator

Navigator (on the right)**:** a hierarchical tree of functions, examples, documents and libraries. This area is view-only (locked).

5.1.1. Navigator buttons

- **Functions** — hierarchical index of Opusmodus functions by topic (e.g., *Analysis*, *Counterpoint*, *Envelopes*). Click a function to open its documentation in the Assistant.
- **Documents** — curated manuals and guides, including *OMN The Language*, *How-to*, and score examples. Click to open PDFs and example files.
- **Library** — bundled resource libraries: Default-Instrument-Sets, **Default-Library**, **Default-Sound-Sets**, **Default-Unfold-Sets**. These are read-only reference materials used by examples and templates.
- **Search** — full-text search across Assistant content (functions and documents). Results are grouped (e.g., **Function Matches, Other Functions**); click any item to open its document.



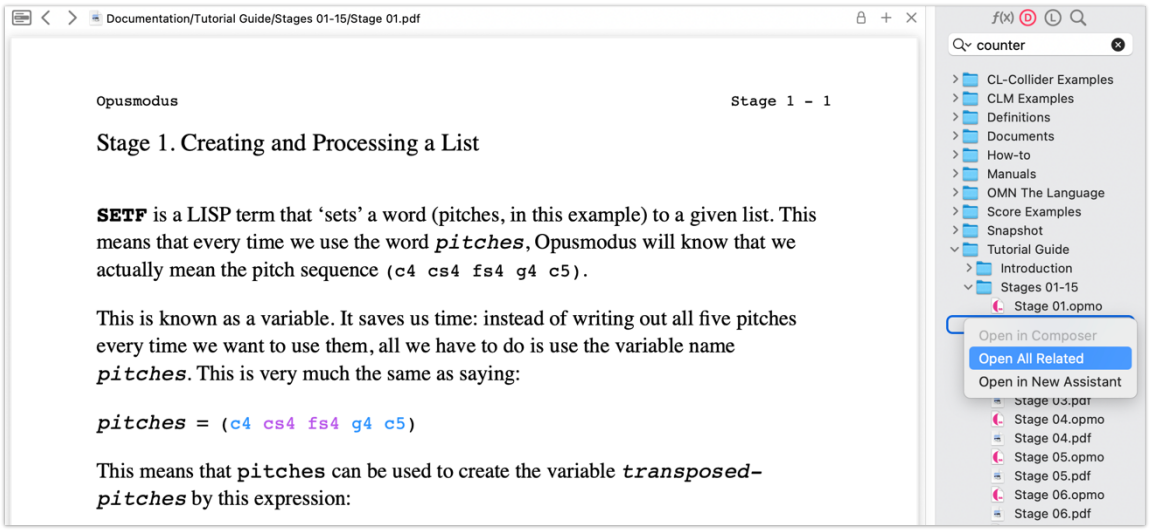
5.1.2 Assistant — Open All Related

When items share the **same basename** but have **different extensions** (e.g., Stage 01.opmo, Stage 01.pdf, Stage 01.png), the Assistant's contextual menu offers **Open All Related**.

Behaviour

- **Open All Related** — opens every file with the same basename in their appropriate panels: score sources (e.g., .opmo) open in the **Composer**,
- documents (e.g., .pdf) and images open in the **Assistant**,
- other supported media open in their respective **Assistant** viewers.
- **Open in Composer** — opens only the selected source in the **Composer**.
- **Open in New Assistant** — opens the selected document in a new **Assistant** window.

Use case. Keep tutorial text, example score, and reference image in sync by naming them identically; a single **Open All Related** command prepares the full context for study or demonstration.



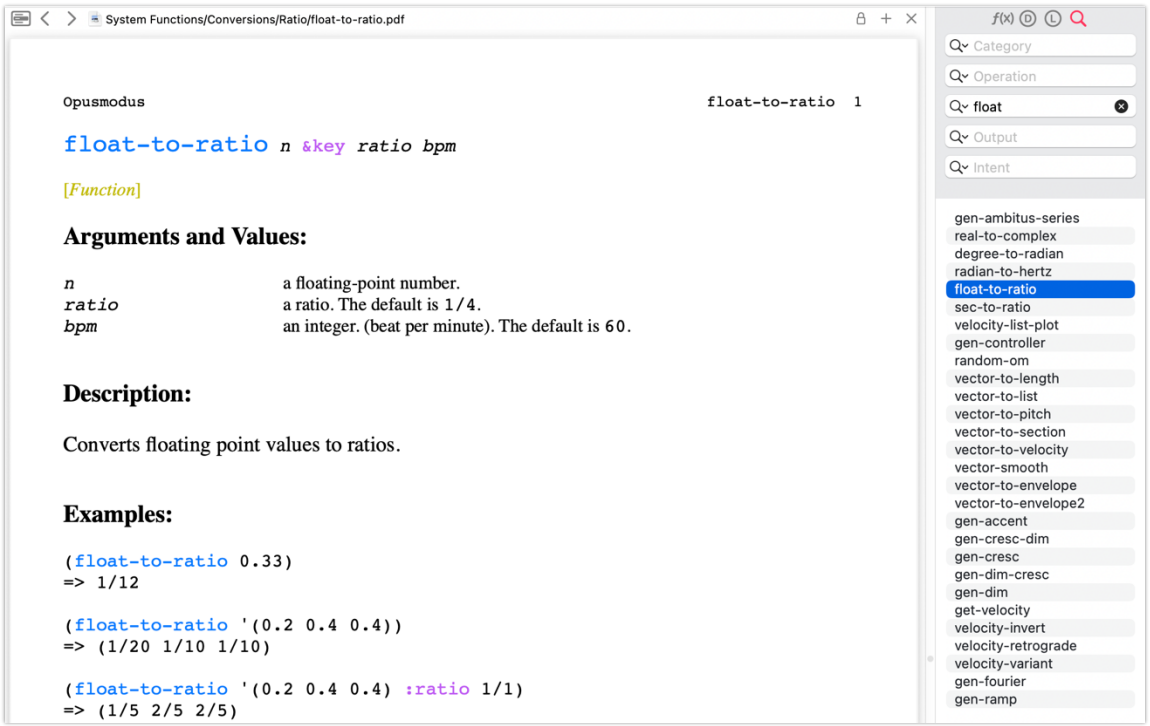
5.1.3 Assistant — Properties Search

Locate functions by **semantic properties** rather than by name. The search pane exposes five fields; entering one or more terms **narrows** the result (logical **AND** across fields).

Fields

- **Category** — topic or domain (e.g., analysis, filter).
- **Operation** — action performed (e.g., generate, assemble).
- **Input** — required input type(s) (e.g., float, pitch, omn).
- **Output** — principal return type (e.g., float, array, omn).
- **Intent** — verb-like descriptor of purpose (e.g., accumulate, align).

Type a term in any field; matching functions are listed on the right. **Click** a result to open its documentation in the **Assistant**.



To discover admissible values for each property, query the system:

```
(function-property-values :category)
=> (ambitus analysis array attribute binary conversion …)

(function-property-values :operation)
=> (align analyse assemble assign calculate …)

(function-property-values :input)
=> (ambitus array attribute bar binary bpm cent chord …)

(function-property-values :output)
=> (alias analysis array articulation assignment …)

(function-property-values :intent)
=> (above abs accent accidentals accumulate add …)
```

5.1.3.1 Direct use in the Composer

Direct use in the Composer. You can run **FUNCTION-SEARCH** directly in the **Composer**; results print to the **Listener**. This is equivalent to the Assistant search. Place the cursor on any returned function name and press **Cmd-D** to open its full documentation in the **Assistant**.

Generate + float output**:**

```
(function-search :operation 'generate :output 'float)
=> (random-om gen-gaussian-noise gen-noise gen-pink-noise
    gen-accent gen-cresc-dim gen-cresc gen-dim-cresc gen-dim
    gen-fourier gen-ramp gen-sawtooth gen-sine gen-square
    gen-triangle)
```

All functions in the filter category:

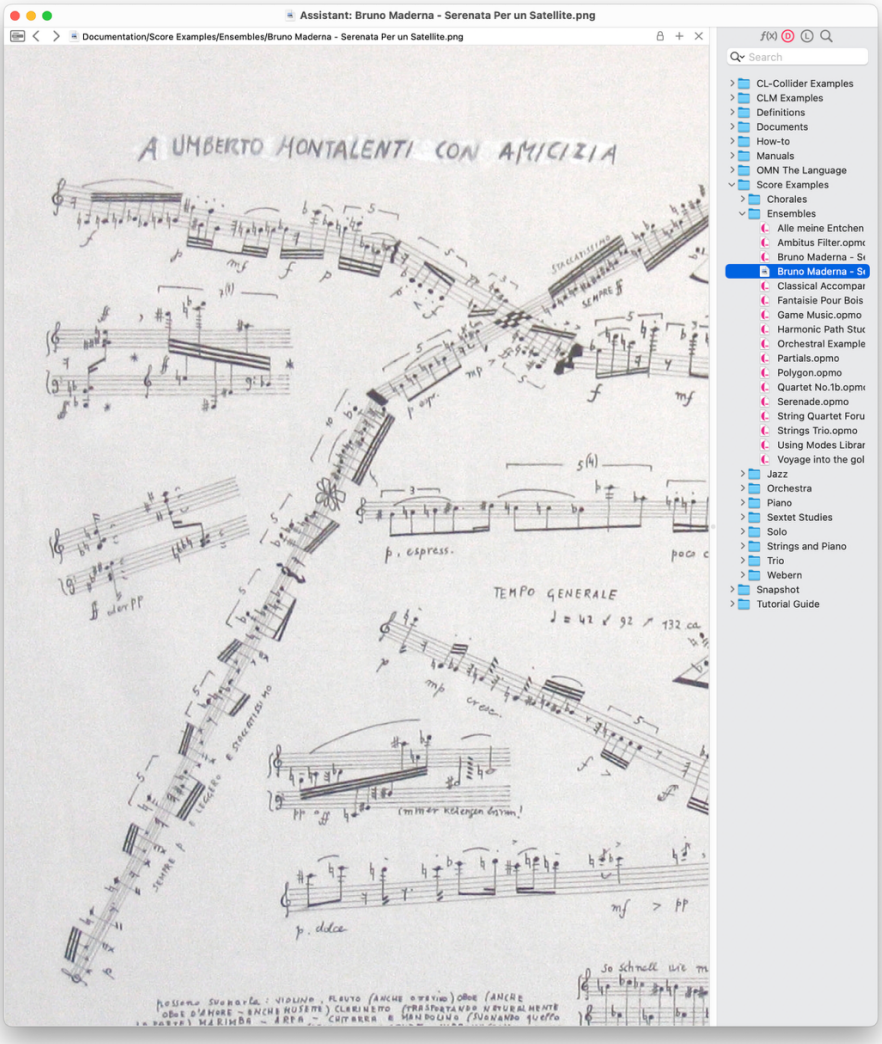
```
(function-search :category 'filter)
=> (gen-filter-ambitus gen-filter-anacrusis gen-filter-change
    gen-filter-euclidean gen-filter-remove gen-filter-rnd
    filter-anacrusis filter-change filter-chromatic filter-exclude
    filter-first filter-last filter-preserve filter-remove
    filter-repeat filter-tie)
```

5.2 Assistant — Content area

A read-only viewer for documentation and source. When the Assistant displays a source/score, it can audition and display results with its own **Listener**, analogous to the **Composer's Listener**.

5.2.1 Assistant — displaying images

The Assistant can display **image files** embedded in the documentation tree (e.g., within *Score Examples*). Supported formats include **.**png, .jpeg, and .tif. **Click** an image entry to open it; use the contextual menu to **zoom in/out** or **fit to window**, and the navigation arrows to move **back/forward** between viewed items. Images are **read-only** and intended for reference alongside code and scores.

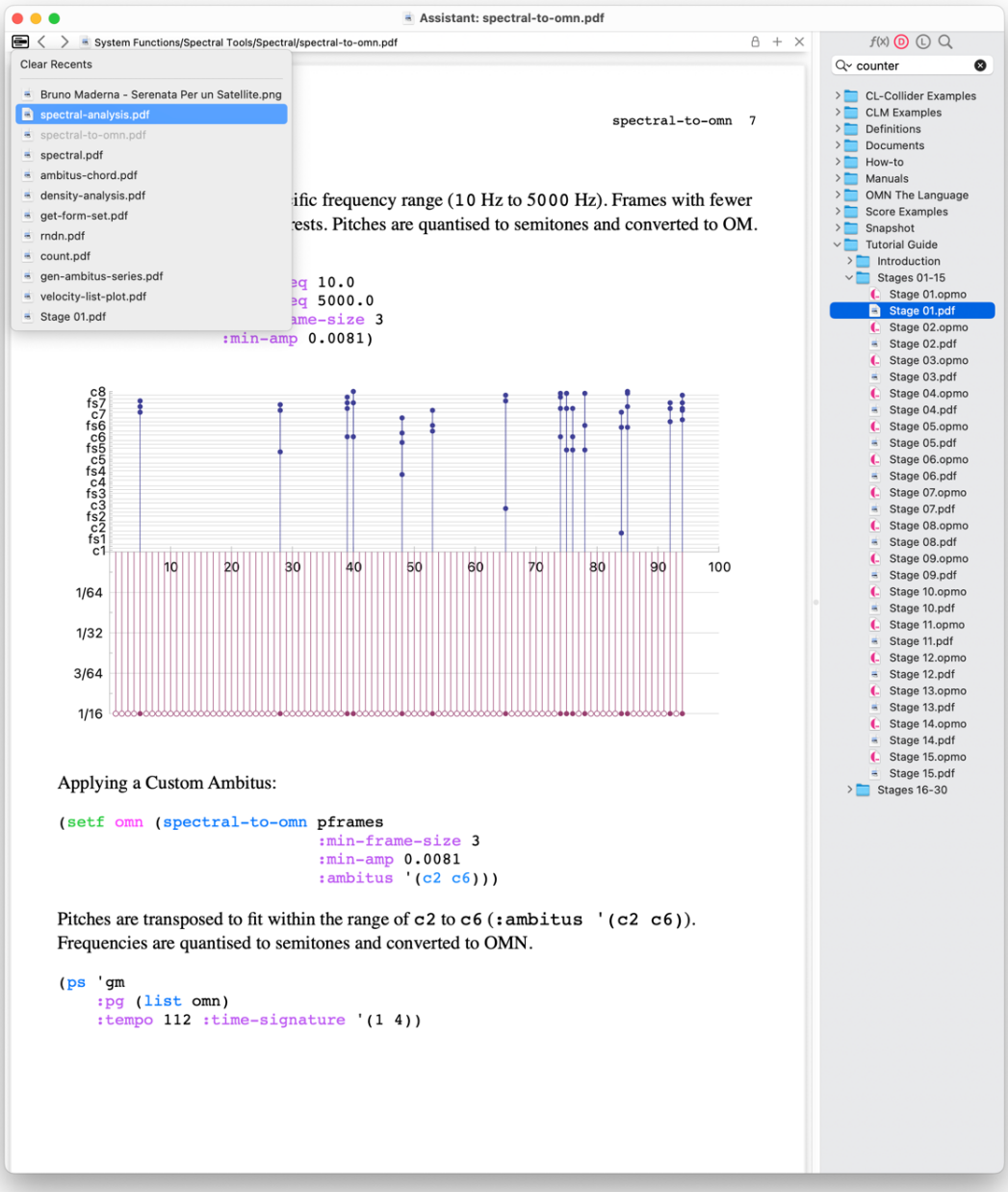


5.2.2 Assistant — Files Grid menu

Quickly navigate among recently opened **Assistant** documents without leaving the current window. Click the **grid icon** in the **Assistant's** title bar to open the drop-down list.

Behaviour

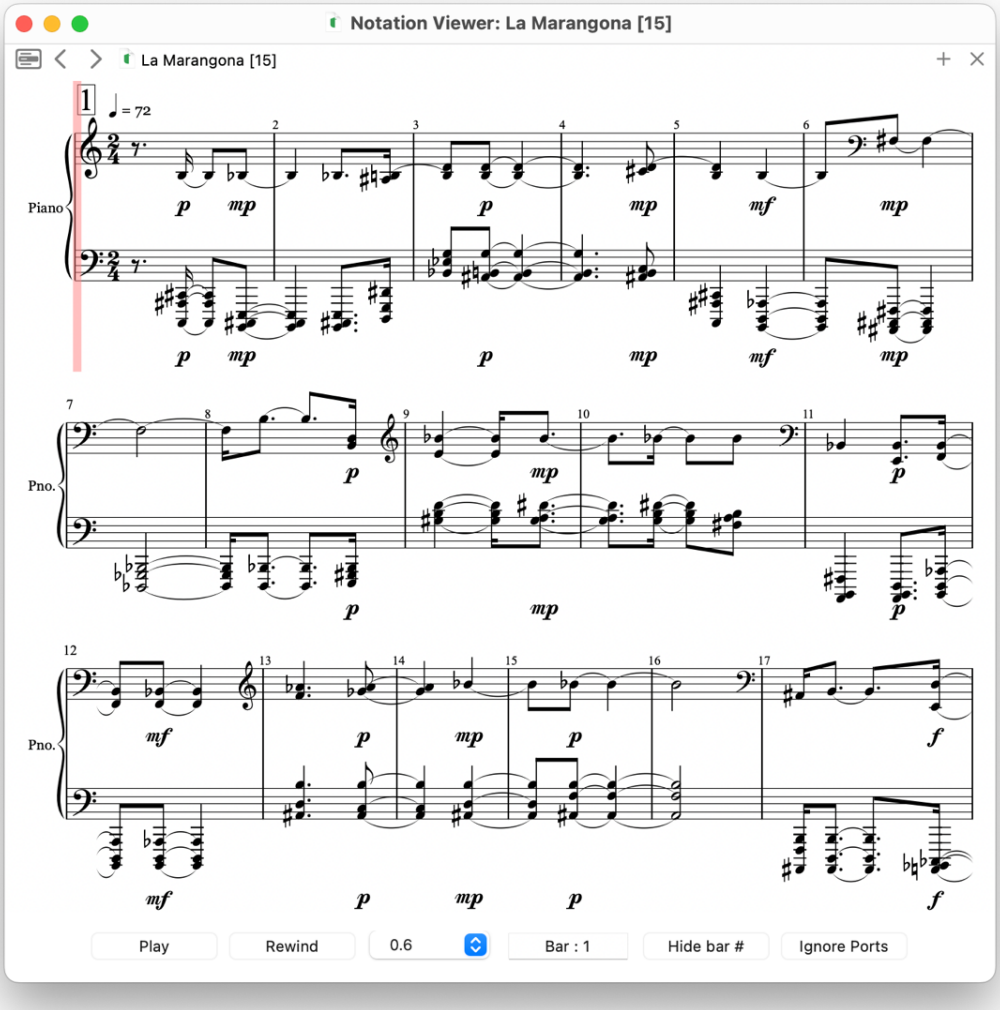
- Lists **recent documents** (most recent at the top).
- **Select** an entry to display it in the current **Assistant** window.
- **Clear Recents** removes the list for the current window/session.



6 Notation Viewer

Render and audition notated results. The Notation Viewer displays scores produced by **Evaluate Score**, **Last Score**, and **Snippet**, as well as **MusicXML** files opened from the Media folder. It provides transport, display controls, and quick access to conversion and export operations.

The viewer is created automatically when you choose **Evaluate Score → Notation** or **Last Score → Notation**, or when a **Snippet** is rendered in notation. MusicXML files open from the **Media** tree in the **Workspace/Navigator**.

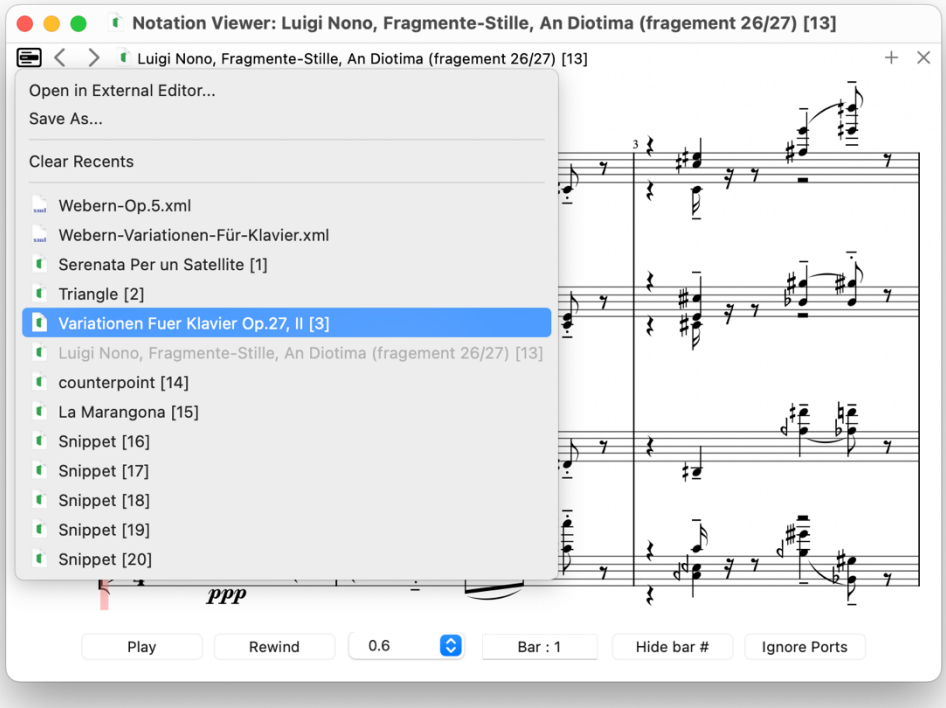


6.1 Transport and display controls

- **Play / Rewind** — audition from the current bar; rewind to the start.
- **Score scale** — display scaling factor for the notation (e.g., **0.6** renders at **60%** size).
- **Bar selector** — jump to a bar before playback.
- **Hide bar #** — toggle bar-number display.
- **Ignore Ports** when enabled, playback is routed to the **built-in General MIDI** device regardless of per-part port assignments; useful for quick checks or when external ports are unavailable.

6.2 Notation Viewer — Files grid menu

Click the **grid icon** to reveal recent items for the viewer. Select to reopen; **Clear Recents** removes the list for the window/session.

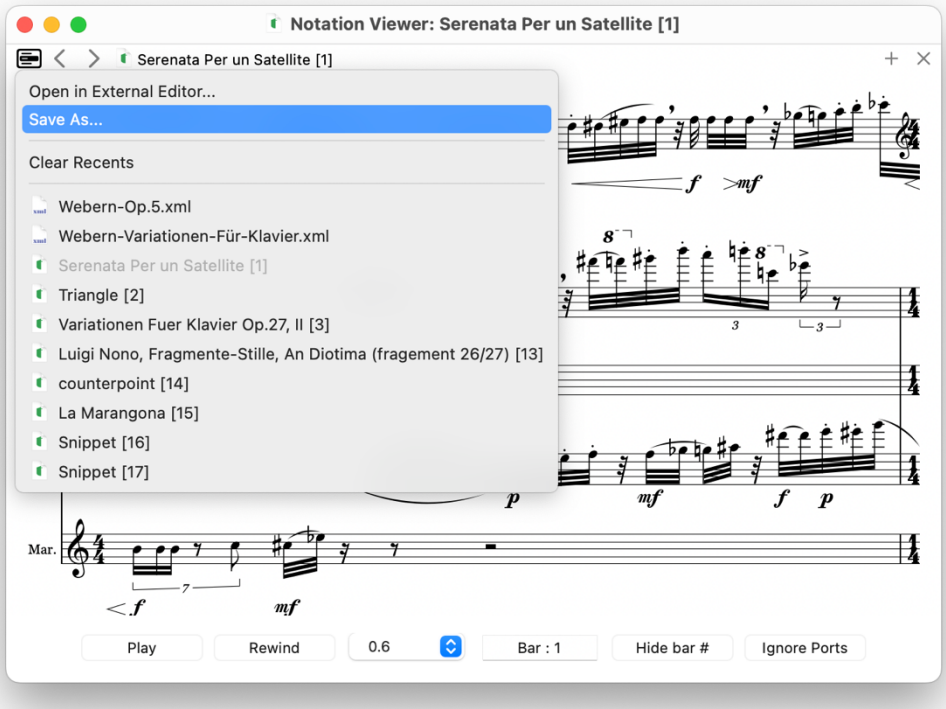


6.3 Notation Viewer — export to MusicXML

Rendered **scores** and **snippets** can be exported as **MusicXML** for use in external notation editors.

- **Command:** from the **Notation Viewer** title bar grid menu, choose **Save As…**.
- **Format:** uncompressed **.xml** (MusicXML).
- **Scope:** exports the **current rendering** (instrument names, layout, key/time signatures, tempo marks, dynamics and articulations as supported).
- **Destination:** choose any folder; the default filename derives from the score/snippet title.

The exported file can be opened directly in editors such as **MuseScore**, **Dorico**, **Sibelius**, or **Finale**.



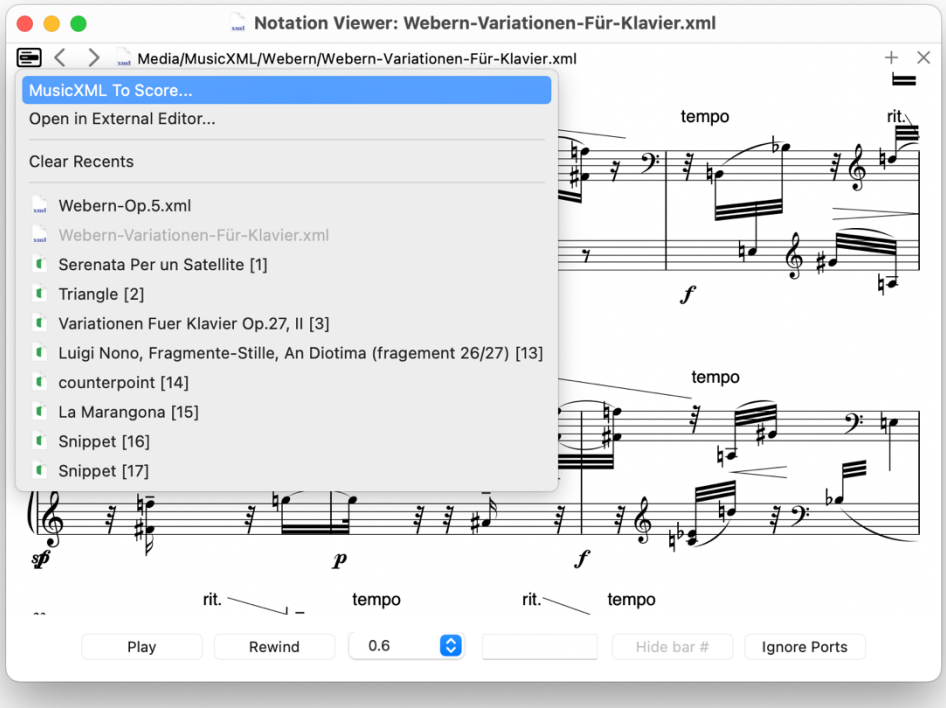
7 MusicXML integration

7.1 MusicXML To Score

Parse a **MusicXML** file (.xml, musicxml or mxl); and generate an Opusmodus **DEF–SCORE** as a **.opmo** file. The definition includes metadata, `instrument layouts`, `tempo`, `time/key signatures`, and OMN `event sequences` for each part. Output is saved in the project's **Scores** directory unless a different `:name` is provided.

- **Header.** A documentation header with date, version, and optional user comment is inserted.
- **Existing files.** Behaviour is governed by `:if–exists` or `:new–index`.
- **De-voicing.** Each notated voice is **separated into its own instrument part**. For example, a two-voice piano staff becomes four voices (RH1/RH2, LH1/LH2). This facilitates independent analysis, re-orchestration, and algorithmic transformation.

In-viewer command. From the Notation Viewer's grid menu on a MusicXML document choose **MusicXML To Score...** to run the conversion and open the generated .opmo.



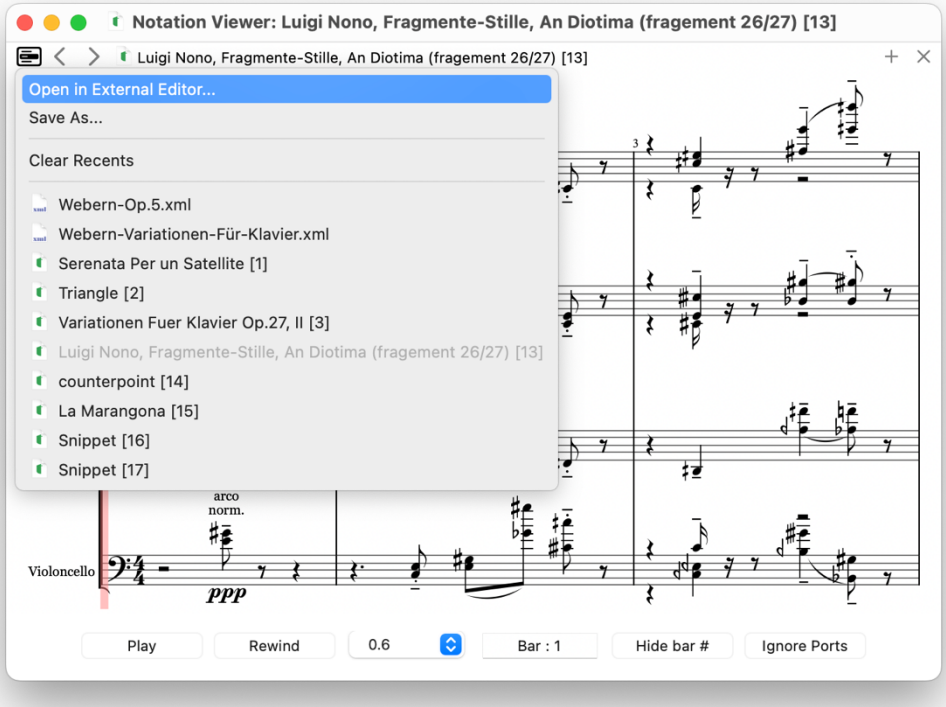
The new score appears in the **Workspace Navigator** and can be opened/edited like any other score.

The generated **.opmo** file is written to the active workspace's **Scores** folder (or to the name you supply via `:name`).

7.2 External notation editor

Open a MusicXML file, or the **most recently compiled score** (from **DEF–SCORE**, **PS**, or **Snippet**), in an external notation editor such as **MuseScore**, **Dorico**, **Sibelius**, or **Finale**.

In-viewer command. Use the grid menu **Open in External Editor…** to export and open the current score.



Default editor

```
(defparameter *default-notation-editor* "MuseScore 4.app")
```

Custom editor

Put the definition in your user **start-up file**, e.g. `~/Opusmodus/User Source/Extensions/User Function.lisp`

```
(defparameter *default-notation-editor* "Dorico 6.app")
(defparameter *default-notation-editor* "Sibelius.app")
```

7.3 Composer — MUSICXML-TO-SCORE

MUSICXML-TO-SCORE imports scores authored in standard notation software (Finale, Sibelius, Dorico, MuseScore) via **MusicXML** and generates an Opusmodus **DEF-SCORE** as a *.opmo file. Voices are **de-voiced** (each notated voice becomes its own part). The resulting score is written to the workspace Scores folder and appears in the **Workspace Navigator** for immediate editing.

Examples

```
;; Default name in Scores
(musicxml-to-score "bach-air.xml")

;; Custom output name
(musicxml-to-score "giant-steps.xml" :name "Coltrane-steps")

;; With a note in the header
(musicxml-to-score "quartet.xml" :note "First draft")
```

Notes

- Expects MusicXML (.xml, musicxml or mxl); paths may be absolute or resolved against your default `Media/MusicXML` directory.
- Overwrite behaviour is controlled by `:if-exists` (or `:new-index`).
- Status messages are printed to the **Listener** on evaluation.

7.4 Composer — MUSICXML-TO-OMN

Return **OMN sequences** from a **MusicXML file** for direct analytical or generative work. The result is **printed to the Listener**.

Examples

```
;; All instrument/voice sequences
(musicxml-to-omn "giant-steps.xml")

;; Only the first instrument/voice
(musicxml-to-omn "giant-steps.xml" :inst 1)

;; Measures 33–48 (inclusive) of the first instrument
(musicxml-to-omn "giant-steps.xml" :inst 1 :start 33 :end 48)
```

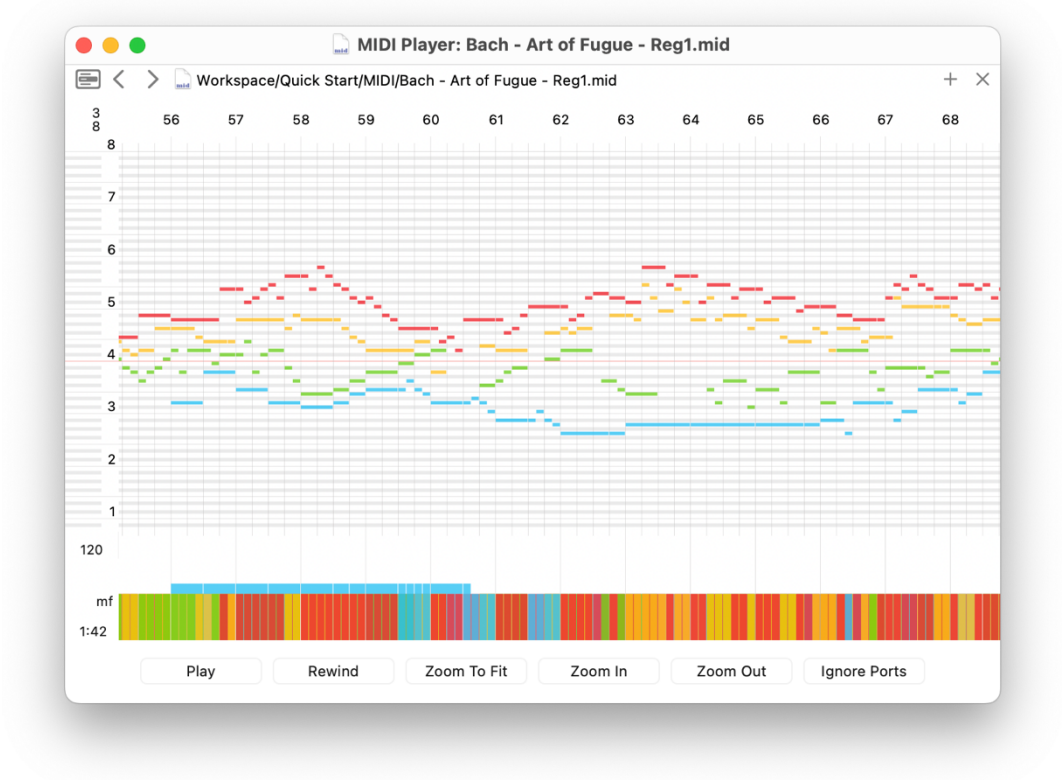
Immediate visualisation

Cmd-1 — open **Notation** (single instrument) for the current **OMN**.
Cmd-2 — open **Voices Notation** (separate voices) for the current **OMN**.

8 MIDI Player

Visualise and audition MIDI as a piano-roll. The **MIDI Player** displays results from **Evaluate Score → MIDI Player**, **Last Score → MIDI Player**, **Snippet → MIDI Player**, and external **.mid** files opened from Media. It offers transport and zoom controls, plus conversion/export via the title bar menu.

Opening. Created automatically from Tools commands, or by clicking a MIDI file in the Media tree.



8.1 Transport and display

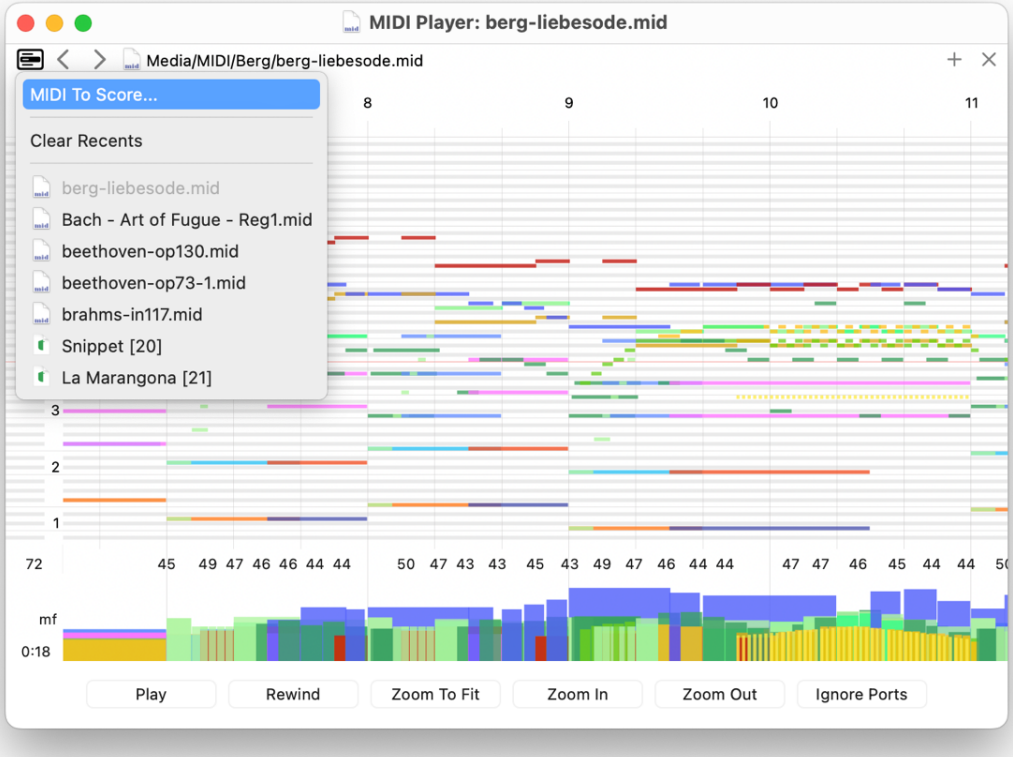
- **Play / Rewind** — audition from the current position; return to the start.
- **Zoom To Fit / Zoom In / Zoom Out** — adjust the time scale of the piano-roll.
- **Ignore Ports** — route playback to the **built-in General MIDI** device regardless of any per-part port assignments (useful for quick checks or when external ports are unavailable).

The vertical axis shows **pitch**; the horizontal axis shows **time**. A compact time strip at the bottom summarises activity.

8.2 Files grid menu (title bar)

Click the **grid icon** to reveal recent items and context actions.

- **MIDI To Score…** — convert the current **Standard MIDI File** into an Opusmodus **score file** (**.opmo**). Output is saved in the project's **Scores** directory.
- **Save As…** — export the **current rendering** as a **.mid** file.
- **Clear Recents** — clear the per-window list.
- **Recent files** — select any listed item to reopen it.

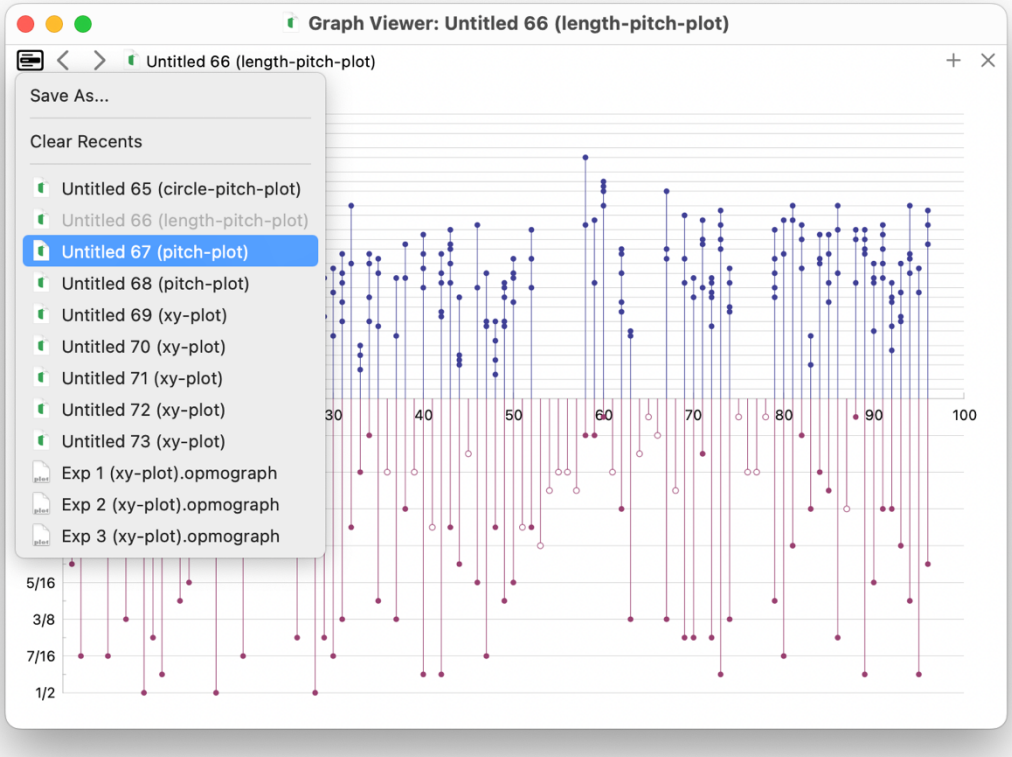


9 Graph Viewer

Display analytical plots generated by **Tools → Graph** or **Last Score → (Length/Pitch/Velocity Graph**). Supported plot types include numbers `list-plot`, `OMN list-plot`, `length-plot`, `pitch-plot`, `velocity-plot`, `circle-pitch`, `circle-rhythm`, and XY.

Opening. Created automatically when a graph command is invoked (from **Composer** or **Snippet**). Multiple plots can be open concurrently.

Display. Axes and legends reflect the selected domain (e.g., `pitches` vs. indices; `lengths` with `rests` shown as negatives; XY pairs). Plots are static analytical views intended for inspection and comparison.



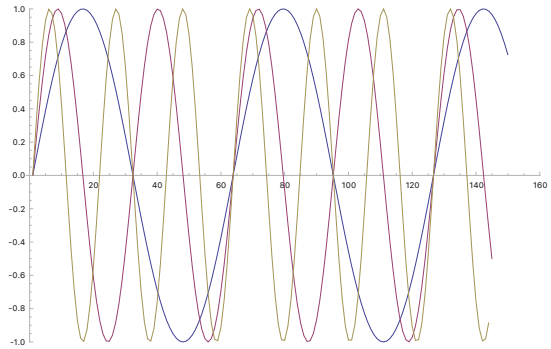
9.1 Files grid menu (title bar)

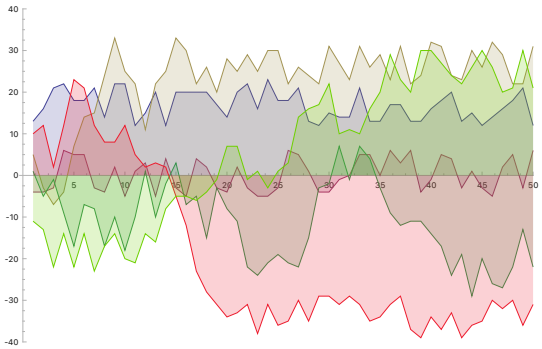
- Click the **grid icon** to access recent plots and file commands.
- **Save As…** — export the current plot as an **.opmograph** file (self-contained graph document).
- **Clear Recents** — clear the per-window list.
- **Recent items** — select any entry to reopen that plot in the current viewer.

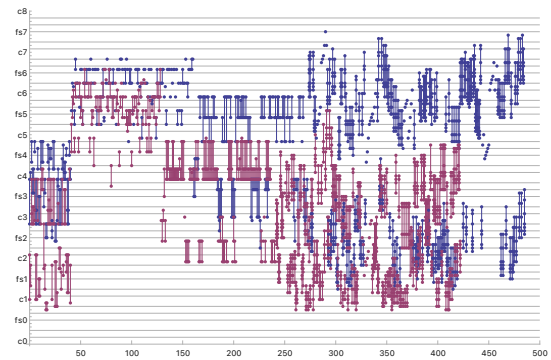
Notes

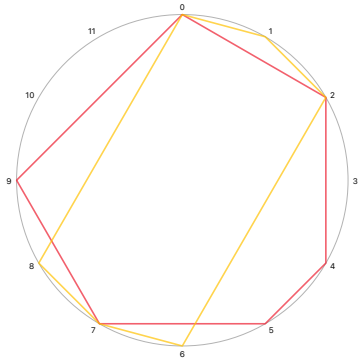
Saved **.opmograph** files can be archived and reopened via the grid menu or Navigator. To regenerate a plot programmatically, keep the corresponding plotting expression in the **Composer**.

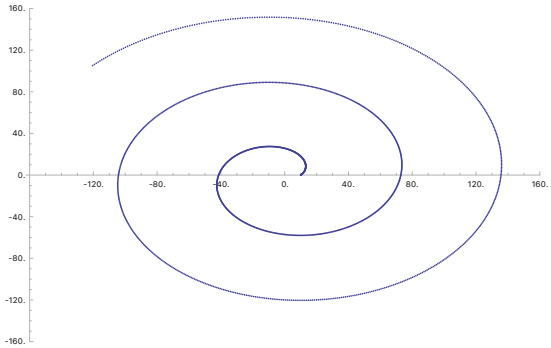
9.2 Graph examples

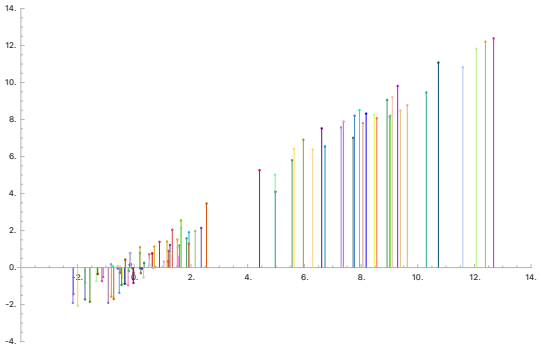


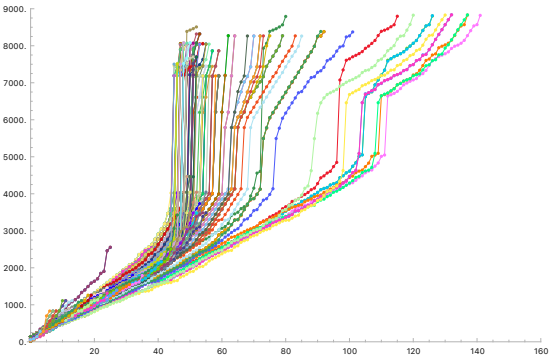


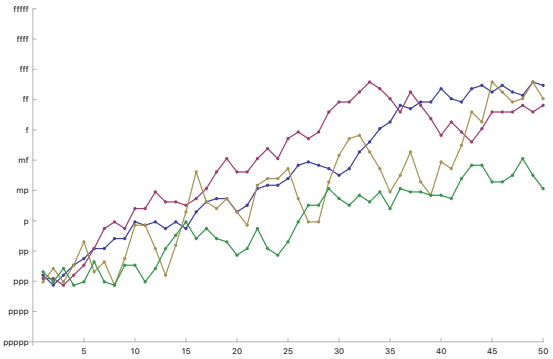


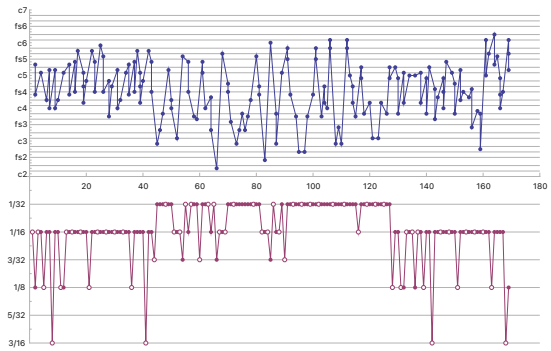


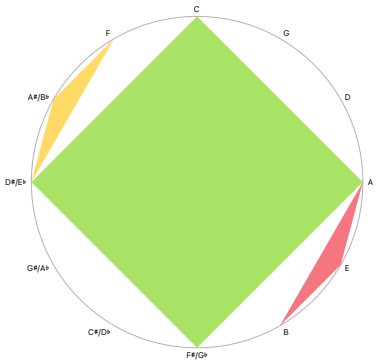

160.
120.
80.
40.
0.
-40.
-80.
-120.
-160.
-120.
-80.
-40.
0.
40.
80.
120.
160.

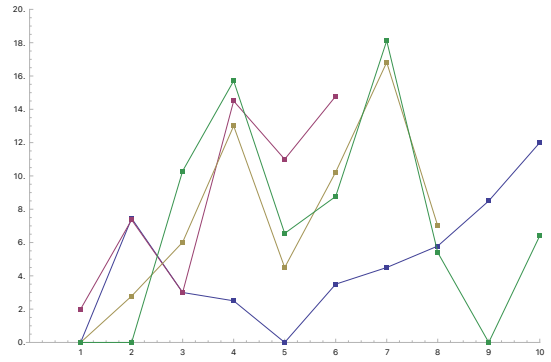

14.
12.
10.
8.
6.
4.
2.
0.
-2.
-4.
-2.
2.
4.
6.
8.
10.
12.
14.

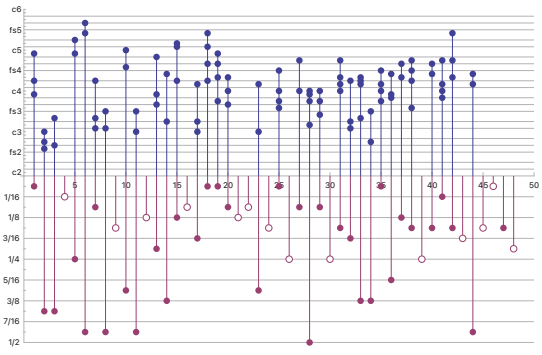

9000.
8000.
7000.
6000.
5000.
4000.
3000.
2000.
1000.
0.
20
40
60
80
100
120
140
160


fffff
ffff
fff
ff
f
mf
mp
p
pp
ppp
pppp
ppppp
5
10
15
20
25
30
35
40
45
50


c7
fs6
c6
fs5
c5
fs4
c4
fs3
c3
fs2
c2
20
40
60
80
100
120
140
160
180
1/32
1/16
3/32
1/8
5/32
3/16


C
G
D
A
E
B
F♯/G♭
C♯/D♭
G♯/A♭
D♯/E♭
A♯/B♭
F


20.
18.
16.
14.
12.
10.
8.
6.
4.
2.
0.
1
2
3
4
5
6
7
8
9
10


c6
fs5
c5
fs4
c4
fs3
c3
fs2
c2
5
10
15
20
25
30
35
40
45
50
1/16
1/8
3/16
1/4
5/16
3/8
7/16
1/2

10. Copy as PDF

You can capture high-quality figures directly from the viewers for use in papers, slides, or reports.

Command. Click inside the viewer to focus it, then press **Cmd-C**.

The **clipboard receives a vector PDF image** of the current view, ready to paste into word processors, layout tools, or graphics applications. (Raster fallbacks may also be provided by the system.)

What is copied

- **Notation Viewer** — copies the **entire score** (**all pages**) as a vector **PDF** at the **current score scale**; the on-screen region does not constrain what is copied.
- **MIDI Player** — the **current piano-roll time window** as displayed.
- **Graph Viewer** — the **current plot canvas**.

Tips for publication-quality output

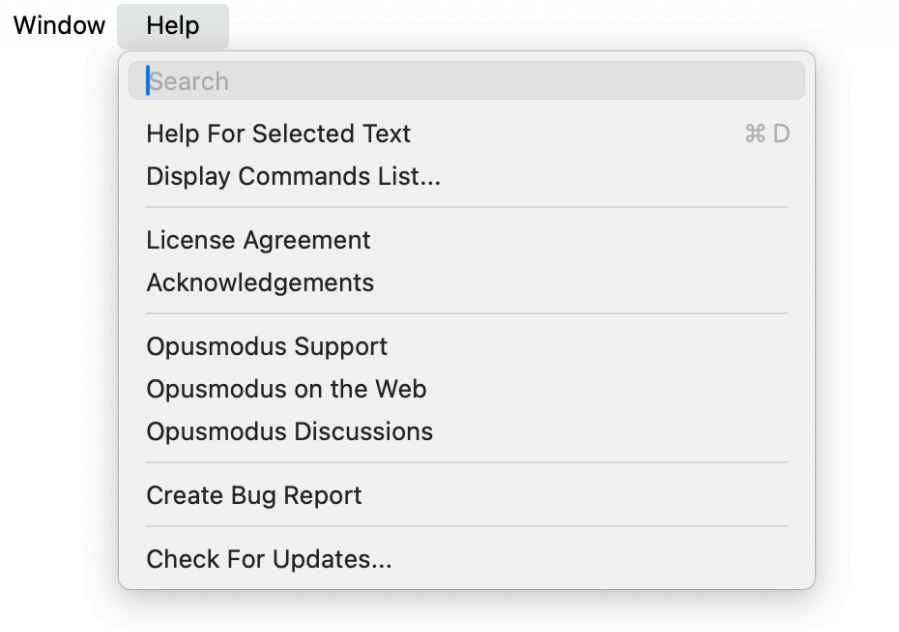
- Set **Score scale / Zoom** (or **Zoom To Fit**) to frame exactly what you need before copying.
- In the Notation Viewer, **Hide bar #** if you require a clean excerpt.
- The copied PDF is **resolution-independent** and prints crisply at any size.

11 Menus and Commands

This section provides a concise tour of the **application menus** relevant to first use, with references to the corresponding panels.

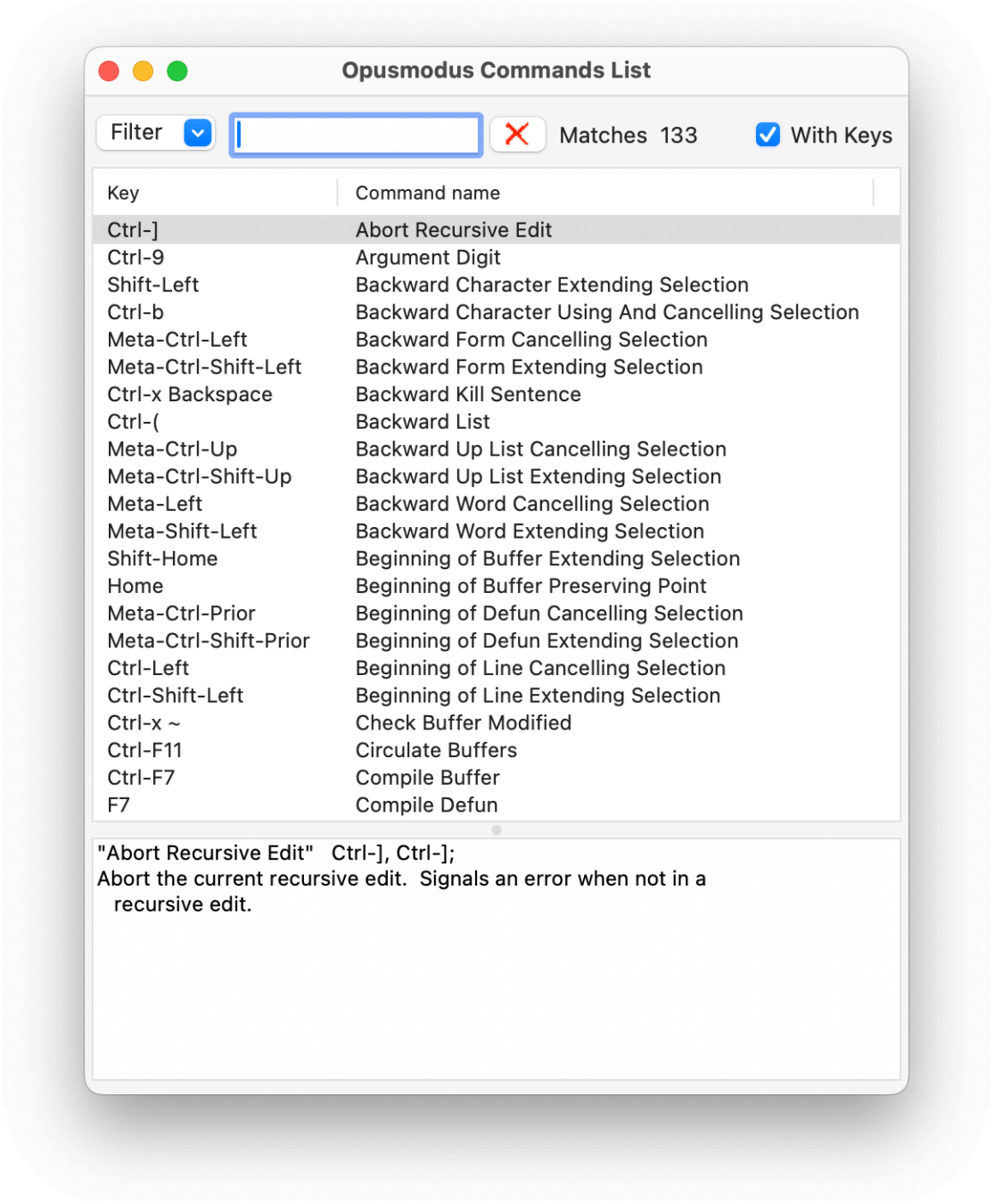
11.1 Help menu

The **Help** menu offers access to assistance, support resources, and update checks.



- **Help for Selected Text** — Opens context help for the current selection.
- **Display Commands List…** — Opens the searchable list of commands and shortcuts.
- License Agreement / Acknowledgements — Displays legal information.
- **Opusmodus Support / Opusmodus on the Web / Opusmodus Discussions** — Opens the official support site, product website, and user forum.
- **Create Bug Report** — Prepares a report with the necessary diagnostic context.
- **Check for Updates…** — Triggers a manual update check.

The **Commands List** provides search, an option to restrict to commands with defined keys, and a detail pane for the selected command.



11.2 Opusmodus menu

The Opusmodus application menu contains global settings and licensing.

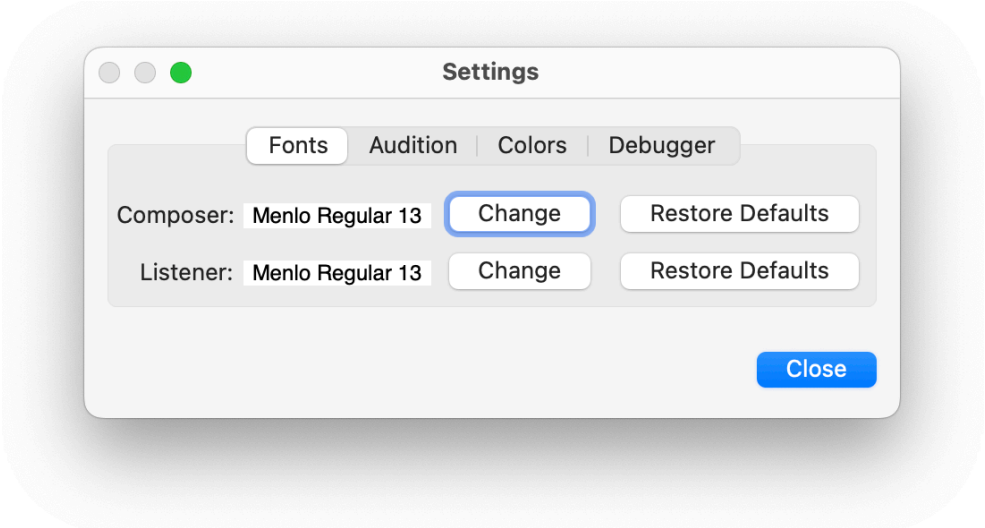
- **Settings…** — Opens application preferences.
- **License…** — Opens licensing status and actions.
- **Services / Hide / Quit** — Standard operating system items.

11.2.1 Settings

The **Settings window** comprises four tabs. Defaults can be restored per tab.

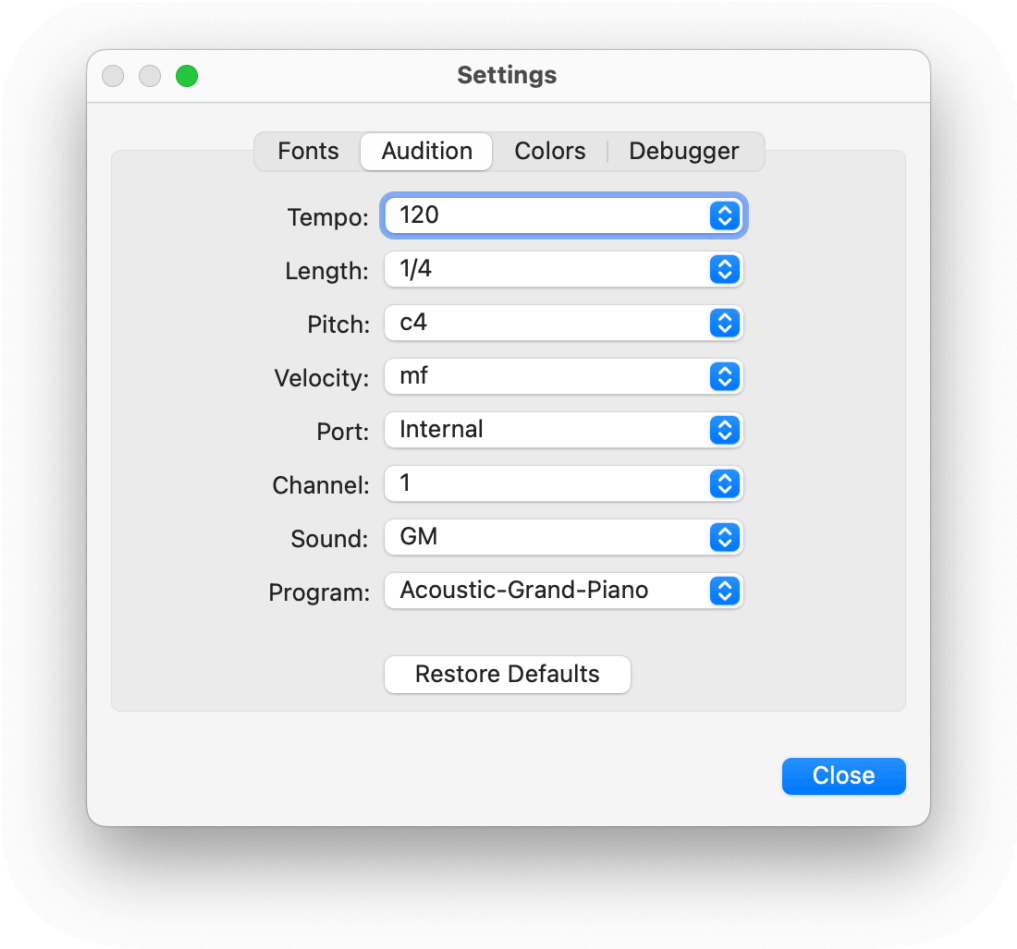
11.2.1.1 Fonts

Composer (Editor) and **Listener** font families and sizes.



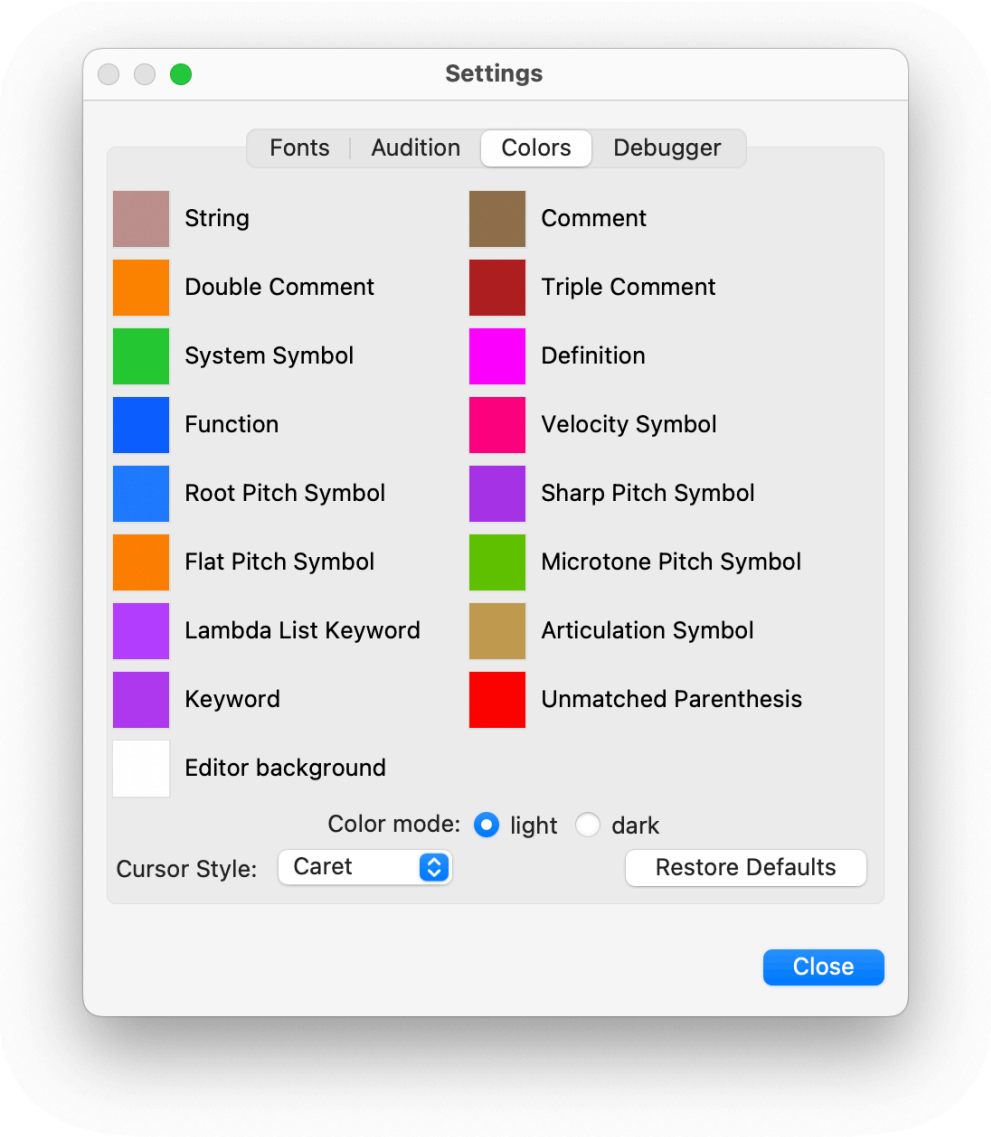
11.2.1.2 Audition

Defaults for **preview playback** (snippets): `tempo`, default `length`, `pitch`, `velocity`, and target (`port`/`channel`, `sound set`, `program`).



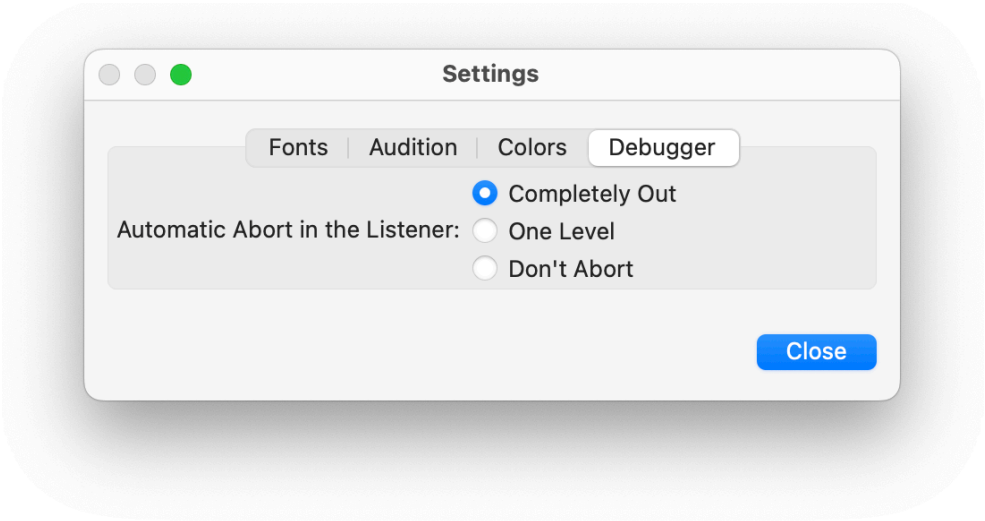
11.2.1.3 Colours

Syntax colours for **Composer** (Editor), including **pitch symbols** and editor background. Choice of light/dark mode and **cursor style**.



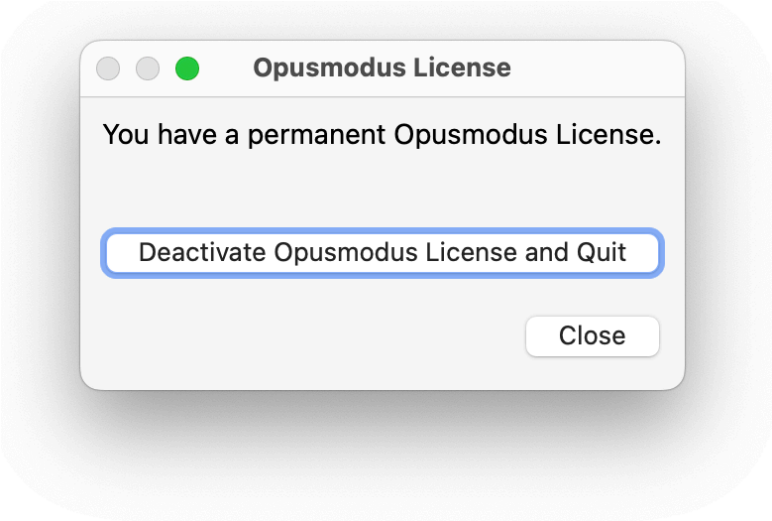
11.2.1.4 Debugger

Behaviour of automatic abort in the **Listener**.



11.2.2 License

The **license panel displays** the current status and allows deactivation when required (e.g., before moving to another machine). **Deactivation** quits the application.



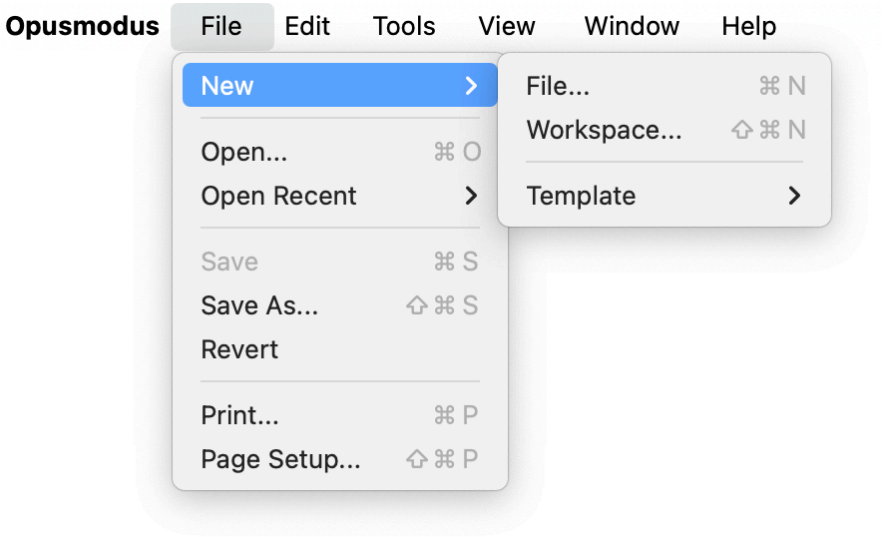
11.3 File menu

This section covers creating a workspace, saving, and evaluating your first file, and clarifies the roles of the **Composer** and the **Listener**.

11.3.1 File menu: New

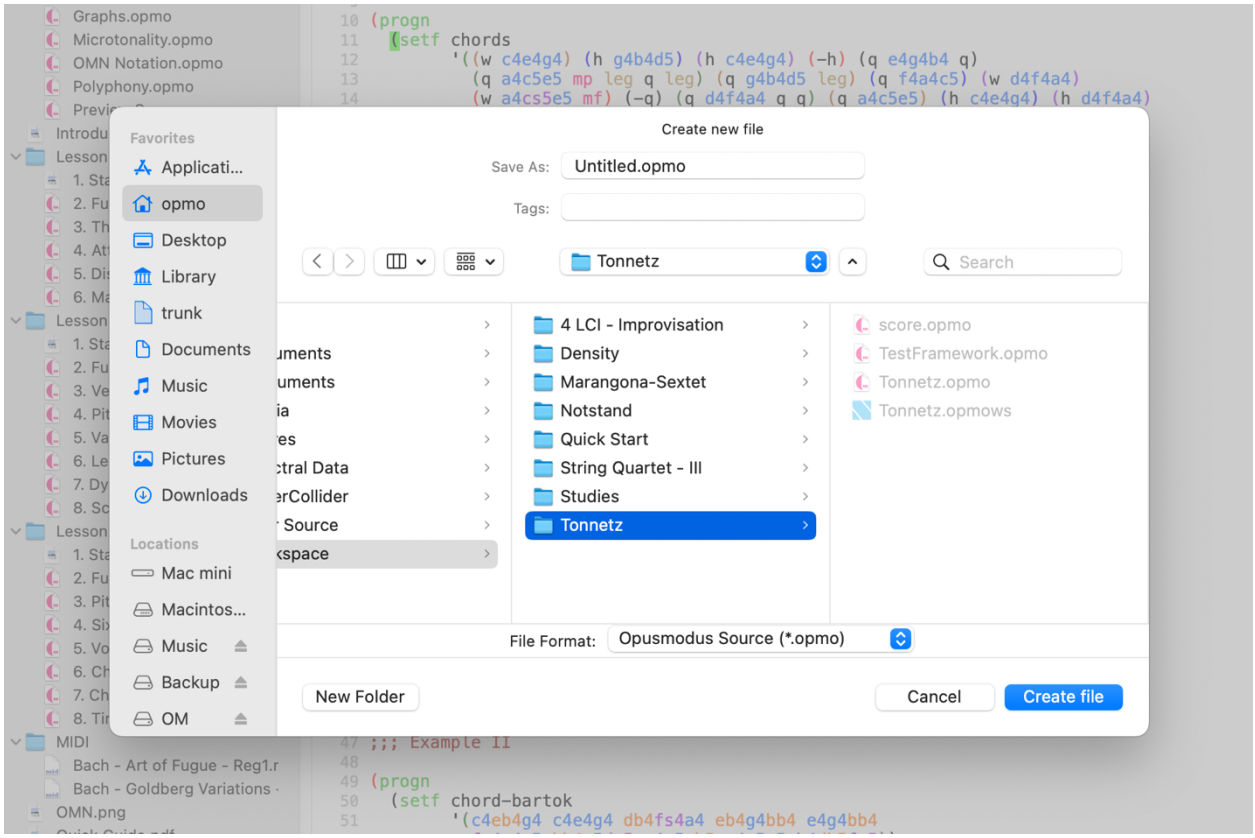
The **File → New** submenu exposes the three principal entry points for work:

- **File...** — create a new source file.
- **Workspace...** — create a new workspace file.
- **Template** — start from a pre-configured scoring template.



11.3.2 New file and saving

Use **File → Save** (**or Save As…**) to store the file inside the active workspace. Use the default extension offered by the Save panel.

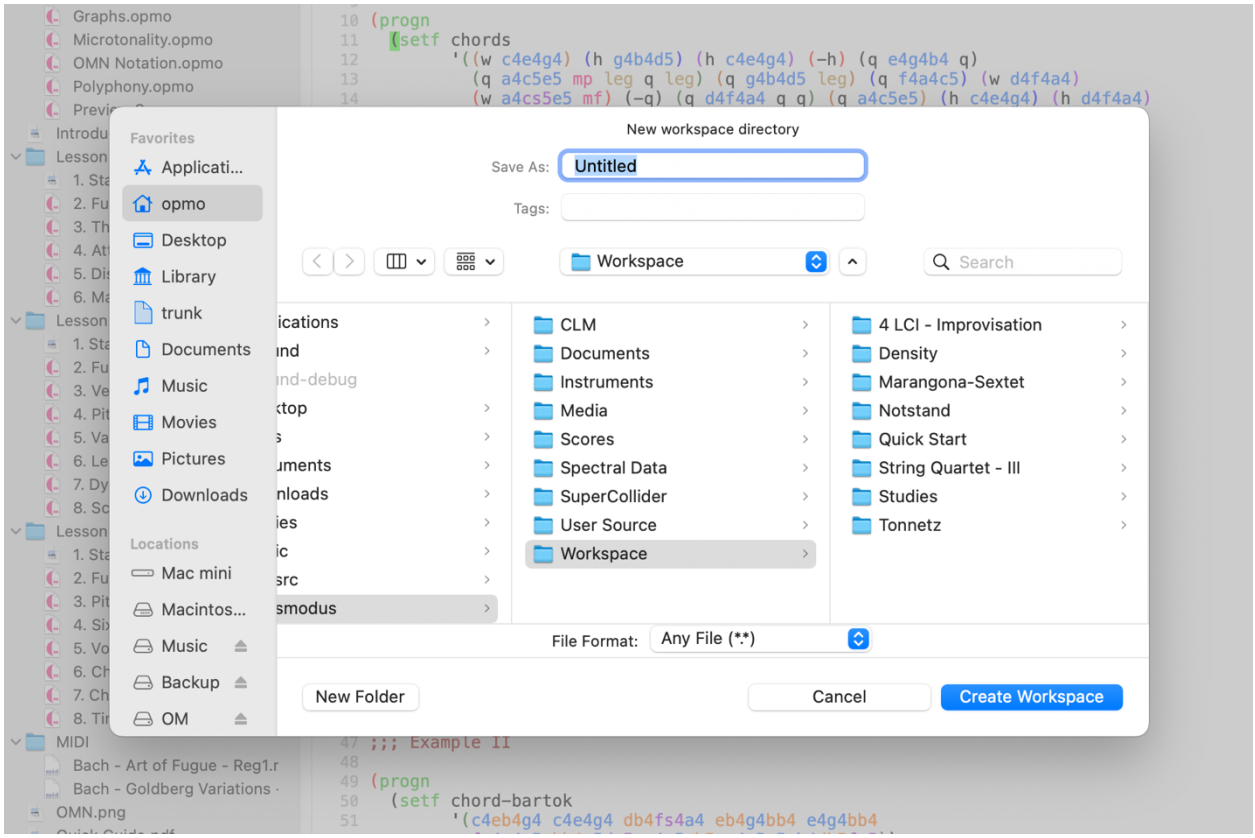


Recommendation: organise files by function (e.g., *materials*, *sketches*, *scores*) using subfolders within the workspace.

11.3.3 Create a workspace

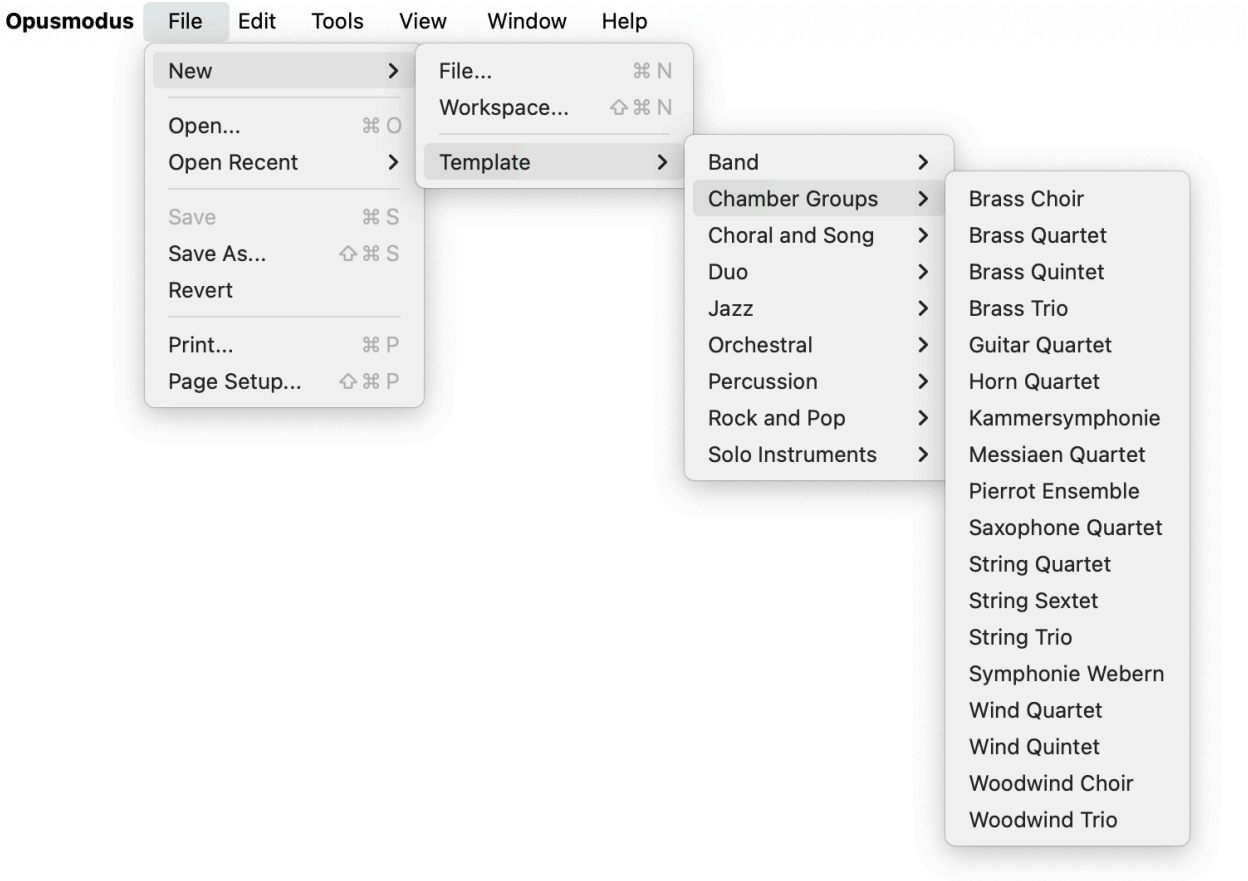
A workspace is a folder where Opusmodus stores and organizes your material. You may maintain multiple workspaces (e.g., per project) and switch between them as needed.

Create: choose **File → New → Workspace...** and select the folder above. Opusmodus creates a **folder** and a **workspace file** of the same name. The workspace file identifies the workspace and can be used to reopen or switch workspaces.



11.3.4 Creating a new file from templates

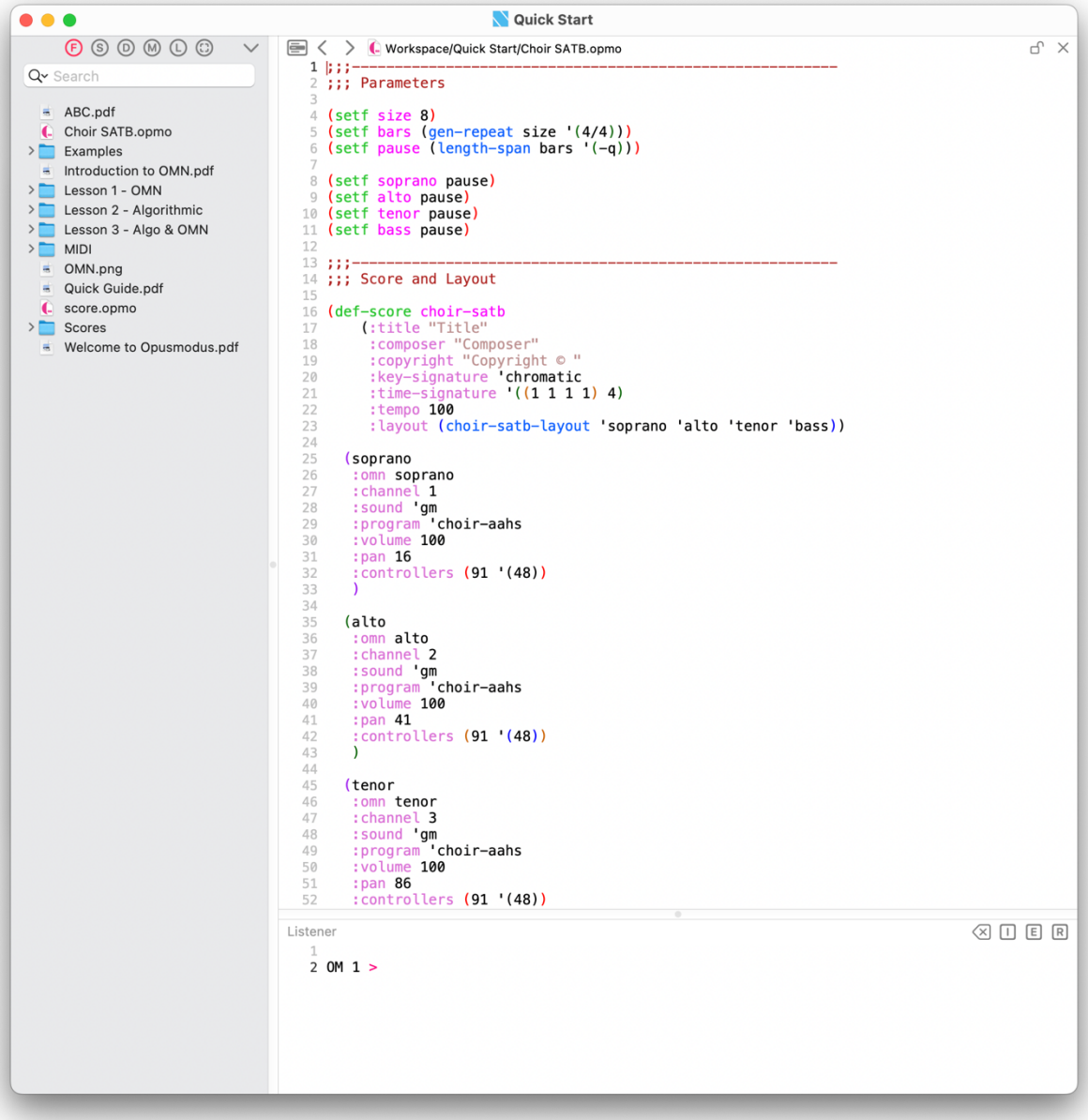
The **File → New → Template** submenu provides ready-made instrumentations (e.g., *Choir SATB*, *Jazz*, *Orchestral*, *Solo Instruments*) for rapid project set-up. Selecting a template creates a **new score file** with pre-configured staves and parts, opened immediately for audition.



Procedure

Choose **File → New → Template**.
Select the desired instrumentation.

The newly created score is **placed in the active workspace** and appears in the **Workspace Navigator** for immediate access.



Notes

Templates are copied into your workspace; the **originals remain unchanged**. The generated score opens in the **Composer** and is ready for playback using the current Audition settings.

11.4 Tools menu

All commands in this menu operate on expressions and scores: evaluation, visualisation, audition, and diagnostics.

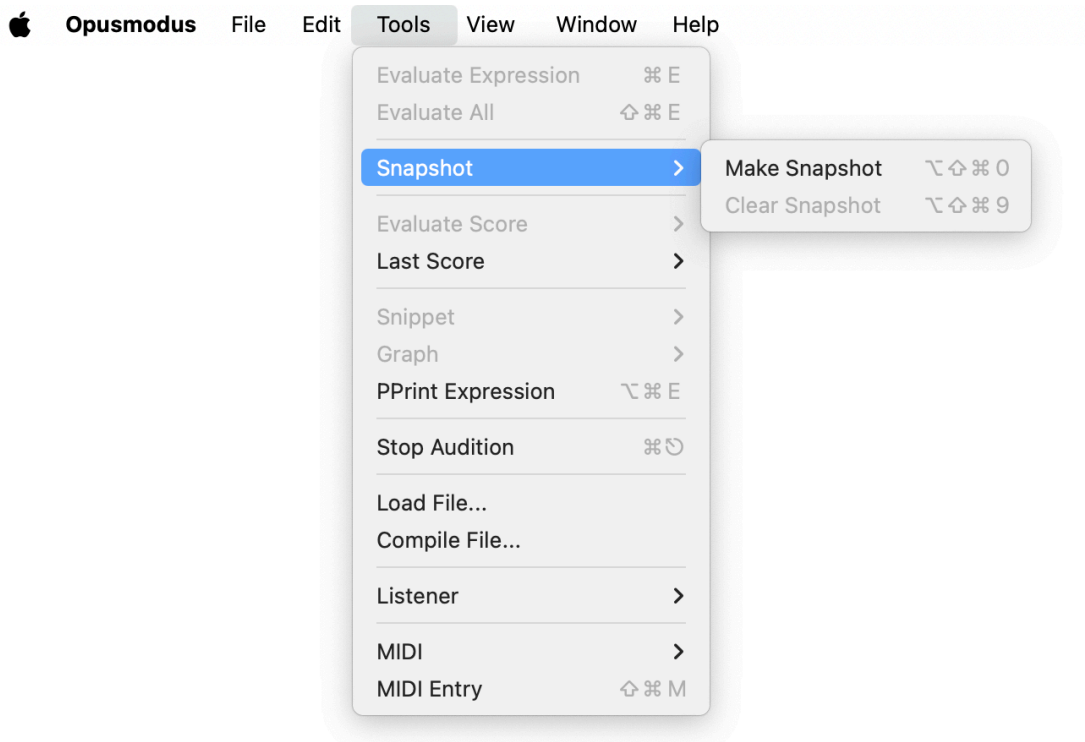
11.4.1 Evaluate Expression / Evaluate All

- **Evaluate Expression** — evaluate the form at point (expression). Any score returned becomes the *current score*.
- **Evaluate All** — evaluate the entire buffer.

11.4.2 Snapshot

The Snapshot utility offers a lightweight, reversible checkpoint for interactive development. It captures the state of a single package - every function, macro, variable, and class definition, so that experimental changes can be discarded without restarting Opusmodus.

- **Make Snapshot** — records the current symbol table of the OM package.
- **Clear Snapshot** — deletes every definition added or altered since the snapshot and restores anything overwritten.



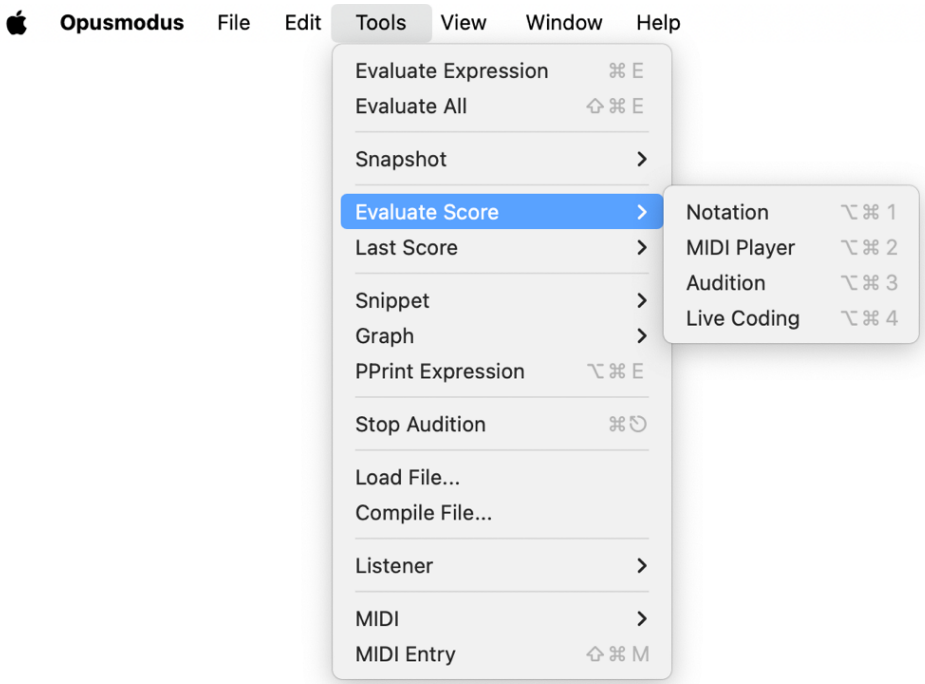
Typical Workflow

1. **Make Snapshot** to capture a clean baseline.
2. Prototype new forms in the **Listener** or a scratch buffer.
3. Test the experimental code.
4. If the code works, move the validated definitions into a file under `~/Opusmodus/User Source/Extensions/`, evaluate the file, and continue.
5. If not, invoke **Clear Snapshot** to return the environment to its baseline.

11.4.3 Evaluate Score

Evaluate the **entire Composer buffer**—functionally equivalent to **Evaluate All**, and then open the requested rendering of the score(s) defined therein. If multiple score forms are present, the **last evaluated score** becomes the *current score* for display/playback.

- **Notation** — engraves the current score in the notation viewer.
- **MIDI Player** — opens the MIDI viewer for the current score.
- **Audition** — plays the current score using the active Audition settings.
- **Live Coding** — sends the current score to the Live Coding Instrument.

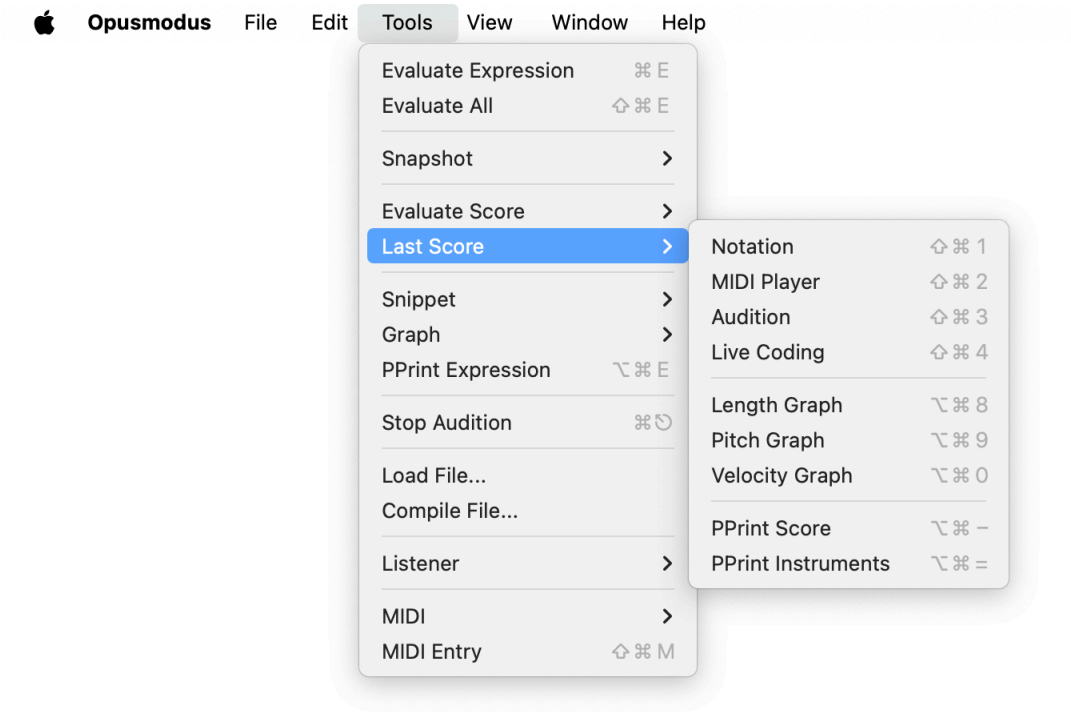


Notes

- Use **Tools → Last Score** to reopen the most recently evaluated score *without re-evaluation*.
- Output and diagnostics appear in the Listener (behaviour governed by **Settings → Debugger**).

11.4.4 Last Score

Reopen views for the most recently displayed score-like result - the `*last-score*` - without re-evaluation. The last score may be a Snippet, a full Score, or any expression previously compiled as a score that produced a Notation, MIDI view or Audition.



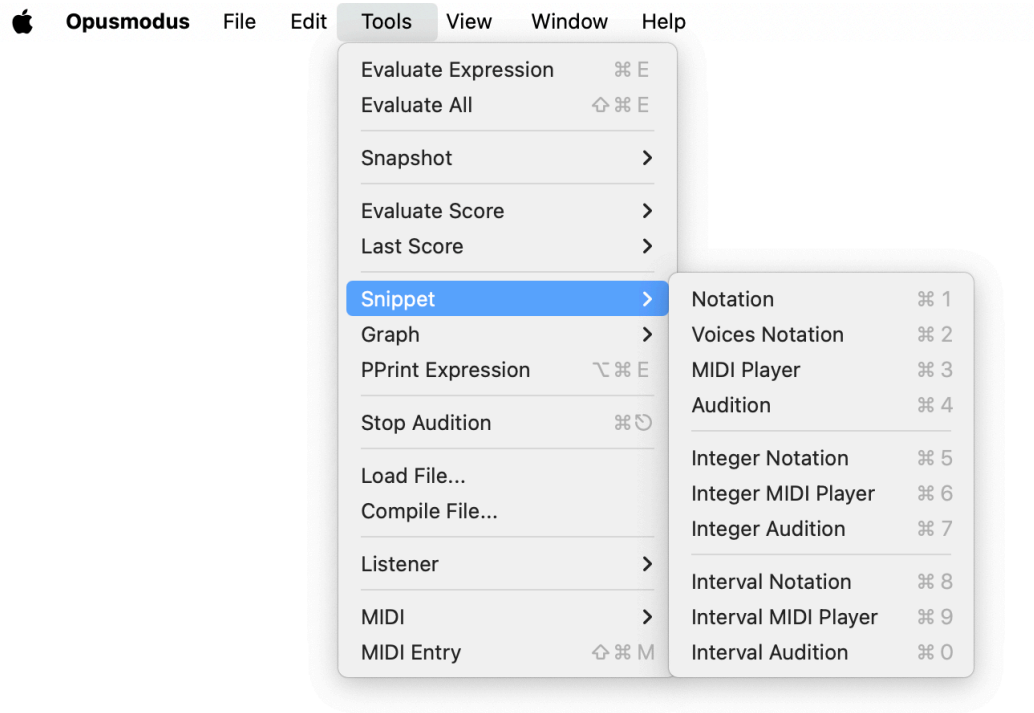
- **Notation** — engraves the last score in the notation viewer.
- **MIDI Player** — opens the MIDI viewer for the last score.
- **Audition** — plays back the last score using current Audition settings.
- **Live Coding** — sends the last score to the Live Coding Instrument.
- **Length Graph** — displays the score's length values in the Graph viewer.
- **Pitch Graph** — displays the score's pitch values in the Graph viewer.
- **Velocity Graph** — displays the score's velocity values in the Graph viewer.
- **PPrint Score** — prints a reconstructible score form to the Listener.
- **PPrint Instruments** — prints the list of instruments referenced by the last score to the Listener.

Notes

The **last score** is session-scoped and updates whenever a new score-like object is evaluated or rendered.

11.4.5 Snippet

Preview a short passage, either the current selection or the expression at point, without constructing a full score. **Snippet** commands render to a viewer and/or audition immediately; the source buffer is not altered.



OMN (Opusmodus Notation) variants

- **Notation** — engraves the selection in a lightweight notation viewer.
- **Voices Notation** — engraves two or more **OMN lists** as independent voices.
- **MIDI Player** — shows the resulting MIDI event stream in the MIDI viewer.
- **Audition** — plays the selection using the current Audition settings.

Integer-domain variants

- **Integer Notation** — displays integer lists in the notation viewer via the integer-to-pitch mapping in force.
- **Integer MIDI Player** — opens the MIDI viewer for the derived event stream.
- **Integer Audition** — plays the material derived from the integer list.

Interval-domain variants

- **Interval Notation** — converts the interval list to a pitch stream and engraves it in the notation viewer.
- **Interval MIDI Player** — opens the MIDI viewer for the converted stream.
- **Interval Audition** — plays the material derived from the interval list.

Notes

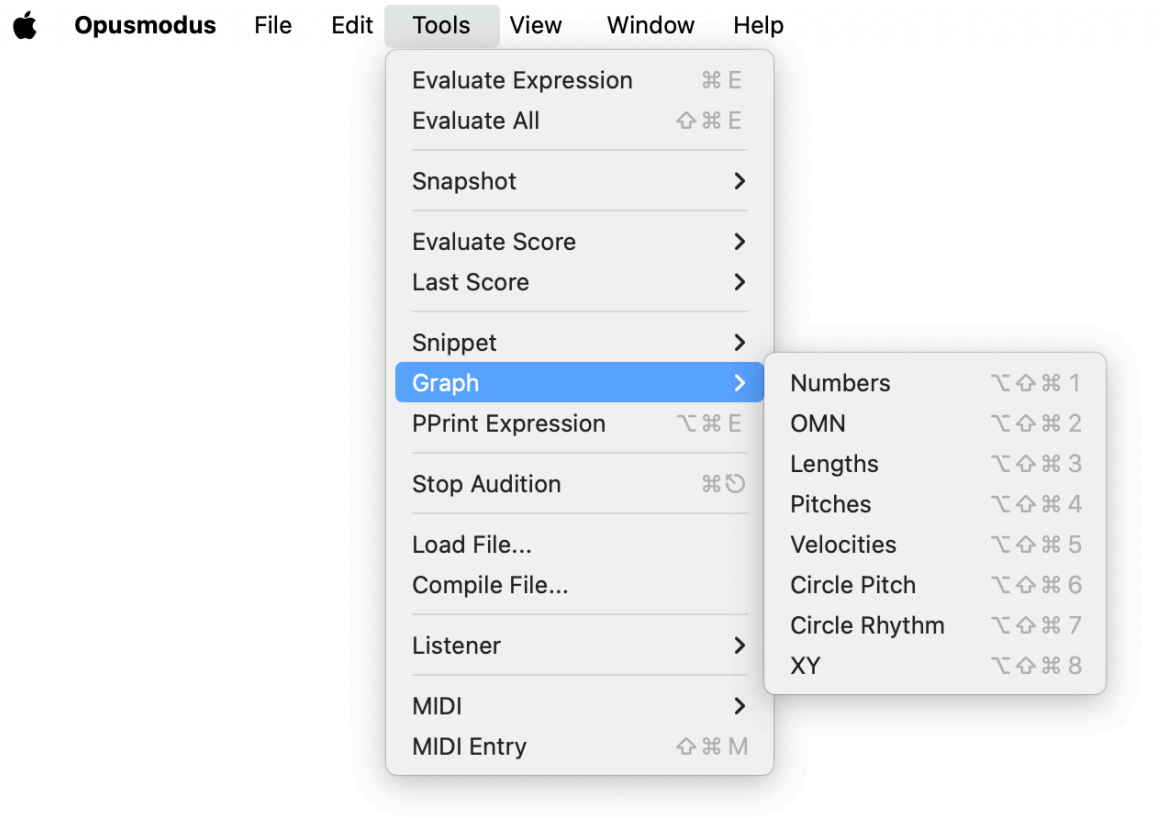
Snippet operates on compact material intended for rapid inspection and audition.

11.4.6 Graph (Plot)

Open analytical plots for **numbers** or **OMN** streams—lengths, pitches, velocities, circle pitch/rhythm, and XY traces—on the current **selection** or the **expression at point**.

Options

- **Numbers** — display a selection in a list-plot view.
- **OMN** — display a selection in an OMN list-plot view.
- **Lengths** — display a selection in a length-list-plot view.
- **Pitches** — display a selection in a pitch-list-plot view.
- **Velocities** — display a selection in a velocity-list-plot view.
- **Circle Pitch** — depict pitch-class sets on the chromatic circle (accepts pitch symbols or integers 0–11).
- **Circle Rhythm** — depict rhythmic patterns on a circle of pulses.
- **XY** — plot pairs of (x,y) values (e.g., time vs amplitude).



11.5 PPrint Expression

Pretty-print the value of the current expression in the **Listener** for inspection.

11.6 Stop Audition

Immediately stops all ongoing playback.

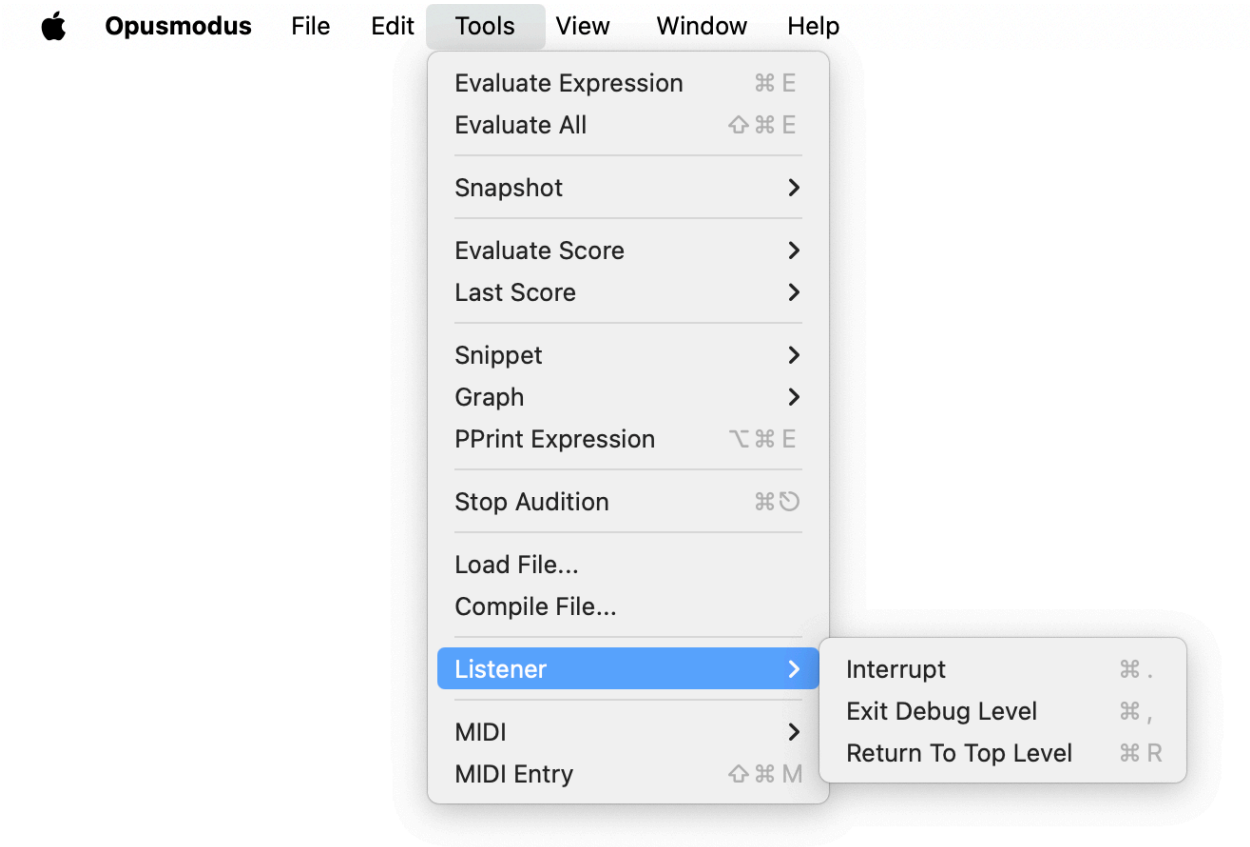
11.7 Load File… / Compile File…

Load File… — load and evaluate an external file into the current image.
Compile File… — compile definitions without opening them.

11.8 Listener controls

Send control signals to the Listener/debugger.

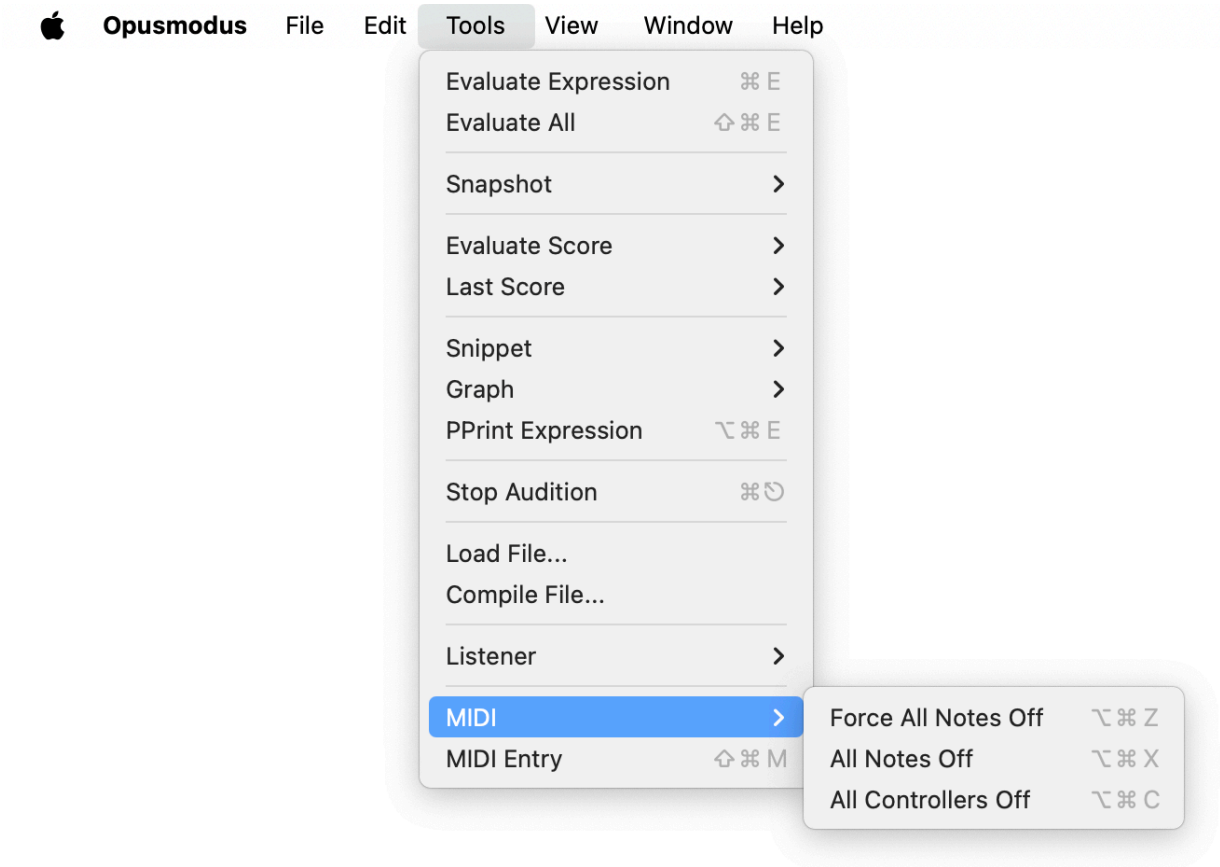
- **Interrupt** — break the current evaluation.
- **Exit Debug Level** — unwind one level of the debugger.
- **Return To Top Level** — return to the REPL's top level.



11.9 MIDI utilities

Emergency and cleanup commands for connected devices.

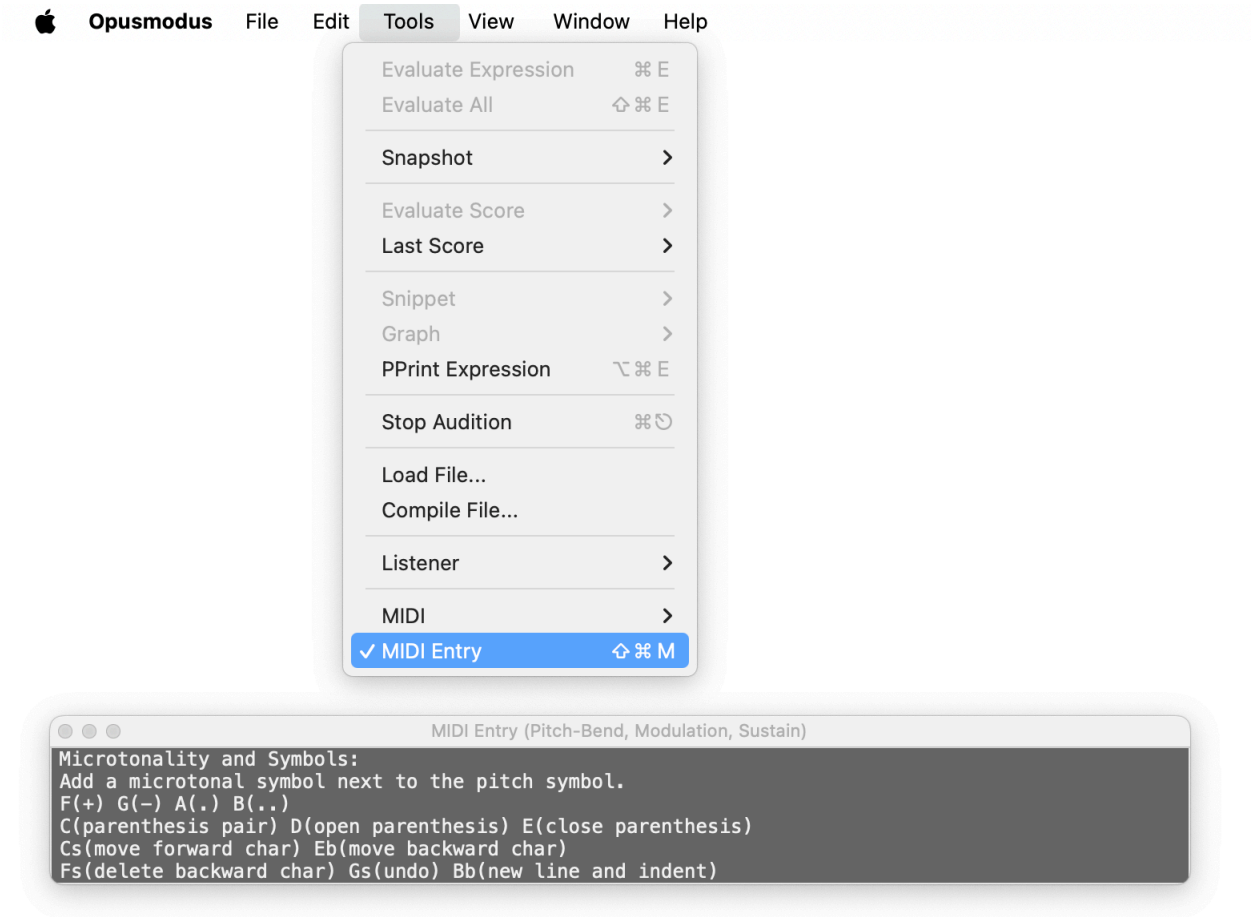
- **Force All Notes Off** — MIDI panic (all channels).
- **All Notes Off** — silence notes.
- **All Controllers Off** — reset controllers.



11.10 MIDI Entry

To start **MIDI Entry** in a score file, choose **Tools → MIDI Entry**. When active, a floating console appears beside the **Listener**.

The console reflects the current state of **pitch-bend**, **modulation wheel**, and **sustain pedal**. Commands are designed for efficient entry directly from the MIDI keyboard. All entries are **octave-independent**, except pitches.



11.10.1 Pitch-bend entries

Pitches — enter pitch symbols and chords into your score.

11.10.1.1 Length & Tuplet Notes (Up-Full)

Key	Value
C	1
D	1/2
E	1/4
F	1/8
G	1/16
A	1/32
B	1/64
Cs	3
Ds	5
Fs	6
Gs	7
Bb	9

11.10.1.2 Length & Tuplet Rests (Down-Full)

Key	Value
C	−1
D	−1/2
E	−1/4
F	−1/8
G	−1/16
A	−1/32
B	−1/64
Cs	−3
Ds	−5
Fs	−6
Gs	−7
Bb	−9

11.10.1.3 Velocities (Up-Half)

Key	Velocity
C	pppp
Cs	ppp
D	pp
Eb	p
E	mp
F	mf
Fs	f
G	ff
Gs	fff
A	ffff
Bb	<
B	>

11.10.1.4 Attributes (Down-Half)

Key	Attribute
C	stacc
Cs	ord
D	marc
Eb	fermata
E	mart
F	ten
Fs	pizz
G	leg
Gs	trem
A	tie
Bb	ped1
B	ped

11.10.2 Microtonality and commands — Modulation wheel (Up)

Add a microtonal symbol next to the pitch symbol and issue editing commands from the keyboard.

Key	Action
F	+
G	–
A	.
B	..
C	parenthesis pair
D	open parenthesis
E	close parenthesis
Cs	move forward char
Eb	move backward char
Fs	delete backward char
Gs	undo
Bb	new line and indent

Notes

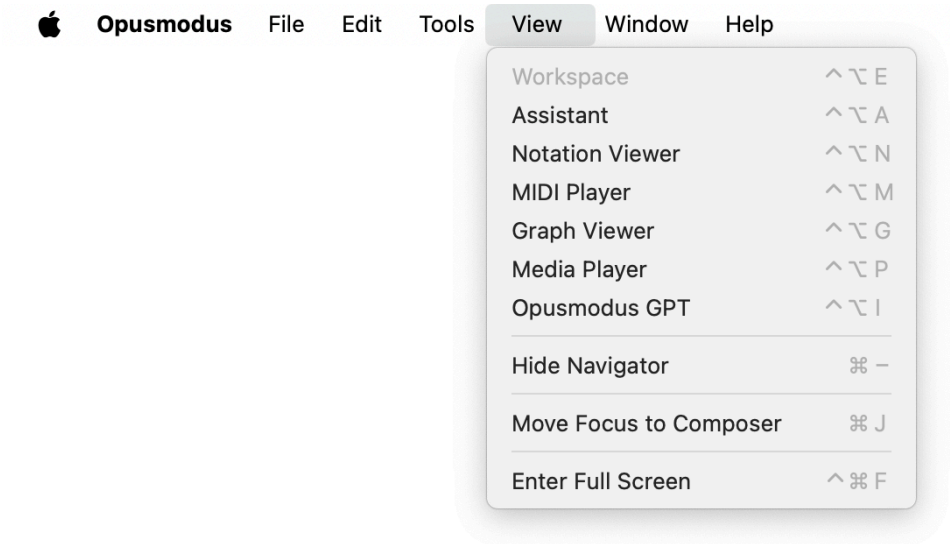
If you enable MIDI input with the modulation wheel **up**, move it **down and up again** for the state to be recognised.

11.10.3 Entries with sustain pedal down

- **Pitch** — add a pitch symbol next to the previous pitch (build a chord).
- **Length and Tuplet Notes** — Pitch-bend: Up-Full; add a length note to the previous value (combine lengths).
- **Length and Tuplet Rests** — Pitch-bend: Down-Full; add a length rest to the previous rest value (combine lengths).
- **Velocities** — Pitch-bend: Up-Half; add ‘**<**’ or ‘**>**’ dynamic symbol to the previous velocity value.
- **Attributes** — Pitch-bend: Down-Half; add ‘**+**’ attribute to the previous attribute value (combine attributes).

11.11 View menu

Open or bring to the front the principal panels of the **current workspace**, and control layout/visibility.



- **Assistant** — focus the current Assistant; if none exists, open a new Assistant window for this workspace.
- **Notation Viewer** — bring the notation window to the front; if no viewer is open, create one (showing the *last score* if available).
- **MIDI Player** — focus or create a MIDI piano-roll window (initialised with the *last score/MIDI* if available).
- **Graph Viewer** — focus or create a Graph window (initialised with the *last plot* if available).
- **Media Player** — focus or create the audio/video player for items from **Media** (e.g., .aif/.wav, .mp4/.mov).
- **Opusmodus GPT** — open or focus the integrated chat assistant window.

11.11.1 Layout commands

- **Hide Navigator** — **toggles** the Navigator of the **frontmost** window: if the **Composer** is on top, it hides/shows the **Composer's Navigator**; if the **Assistant** is on top, it hides/shows the **Assistant's Navigator**.
- **Move Focus to Composer** — set keyboard focus to the **Composer** without changing windows.
- **Enter Full Screen** — toggle macOS full-screen mode for the frontmost window.

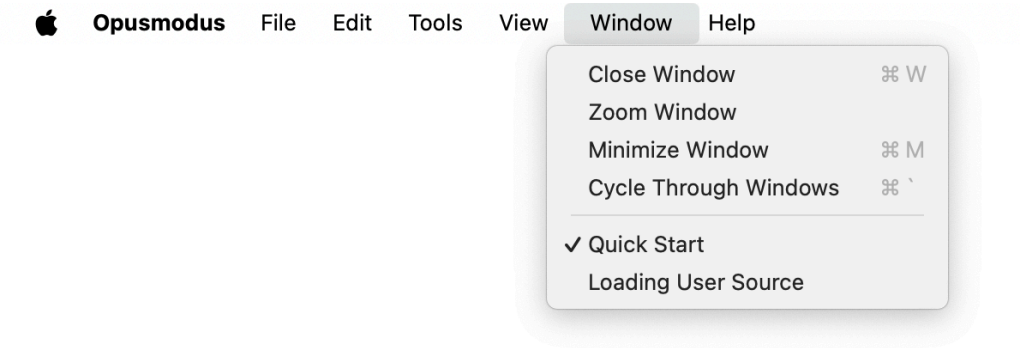
Notes

Each command acts **within the current workspace**. If the relevant panel is absent, a **new instance** is created; otherwise the existing window is **brought to the front**.

Shortcuts are shown in the menu and apply per workspace.

11.12 Window menu

Standard window management for Opusmodus.



- **Close Window** — close the frontmost window.
- **Zoom Window** — toggle between the window's standard and maximised sizes.
- **Minimise Window** — send the frontmost window to the Dock.
- **Cycle Through Windows** — iterate through all open Opusmodus windows.
- **Window list.** Below the divider, all open windows are listed (e.g., *Quick Start*, *Loading User Source*).

Tick ✓ marks the frontmost window.
Selecting an item **brings it to the front** (and restores it if minimised).

Notes

Items such as **Loading User Source** are transient status windows shown during start-up or library loading.

12 OMN The Language

OMN is a **scripting language for musical events**. It encodes the control and organisation of music, rather than audio itself, so that material can be **transformed**, **extended**, and **reorganised** by algorithm.

Rationale. Contemporary composition is multi-faceted and often interdisciplinary (mathematics, literature, the visual arts, etc.). Conventional sequencers and scorewriters provide useful but **limited algorithmic tools** and a **fragmented workflow**. Opusmodus addresses the **composing continuum**: a single environment from first sketches to engraved score and reference audio.

Representation. OMN is a **linear**, **symbolic script**. Parameters (`pitch`, `length`, `velocity` and `articulation`) can be developed **separately** and recombined, enabling both traditional techniques (transposition, inversion, retrograde) and advanced processes (stochastic, fractal, statistical, or graphical methods).

Notation linkage. A defining feature of OMN is its **direct correspondence with staff notation**. Any OMN passage can be rendered instantly to notation and **auditioned**; the language mirrors standard notation not only in rhythm, pitch, and dynamics but also through an extensive **attribute lexicon** covering instrumental and vocal techniques.

Practice. OMN supports rapid, on-the-fly experimentation: try an idea, hear it, view it, and, if successful, **retain it verbatim** in the score. Quick **snippets** provide immediate engraved feedback alongside the script, sustaining a continuous composing workflow.

12.1 OMN: The Four Elements

OMN treats music as a composition of four primary, separable `parameters`. These may be developed independently and recombined.

1) Length (time) — rhythm as duration

OMN uses `length` as the governing term for rhythmic `values` (e.g., `q`, `e`, `s`), since notated rhythm is a pattern of differing durations.

```
(q)
```

2) Pitch

Pitch names with octaves (e.g., `c4`) are added to lengths to produce pitched events; chords are lists of pitches.

```
(q c4)
```

3) Velocity (dynamics)

Dynamic level (e.g., `ppppp` … `fffff`) specifies intensity; it is independent of `pitch` and `length`.

```
(q c4 mp)
```

4) Articulation (attribute)

Articulation and technique tokens (e.g., `ten`, `stacc`, `trill`, `trem`, instrument techniques) further qualify execution. Realisation depends on the active sound set/mapping.

```
(q c4 mp trem)
```

12.2 Assemble and Disassemble

In many workflows it is advantageous to develop **one parameter at a time**. **OMN** supports this by allowing you to **disassemble** a composite **OMN list** into its constituent streams (`length`, `pitch`, `velocity`, `articulation`), process them independently, and then **re-assemble** them into a new **OMN sequence**.

Disassemble:

```
(disassemble-omn '(q c4 mp d4 e4 e f4 f g4))
=> (:length (1/4 1/4 1/4 1/8 1/8)
    :pitch (c4 d4 e4 f4 g4)
    :velocity (mp mp mp f f)
    :articulation (- - - - -))
```

Assemble:

```
(make-omn :length '(q q q e e)
          :pitch '(c4 d4 e4 f4 g4)
          :velocity '(mp mp mp f f))
=> (q c4 mp d4 e4 e f4 f g4)
```

Notes

`DISASSEMBLE-OMN` returns a **property list** keyed by parameter (`:length`, `:pitch`, `:velocity`, `:articulation`).

`MAKE-OMN` composes a sequence from parameter lists; **cardinalities must agree** (one value per event).

Omitted parameters take **neutral defaults** under OMN semantics (e.g., no articulation → **–**).

12.3 OMN: the way forward

This introduction should enable first steps with OMN. The *Stages tutorials* illustrate how closely the language integrates with algorithmic composition.

Working practice. Most core functions accept OMN lists, yet in early stages it is often preferable to keep parameters **separate** (*pitch*, *length*, *velocity*, *articulation*) and combine them later. **The tutorials demonstrate this workflow**.

When to write directly in OMN. Some projects benefit from immediate OMN entry, notably vocal writing. See the *How-To* examples on word-setting with syllabic splitting.

Algorithmic integration. OMN interlocks with integer and interval processes—for example, mapping pitch-class sets to generate tone rows. *The Stages* materials provide concrete models.

Scope. OMN scripts both traditional staff notation and experimental/conceptual practices via parametric modelling. As notation moves from print to screen, OMN offers a precise, programmable representation well suited to digital presentation.

Positioning. Whereas DAWs and scorewriters emphasise production and engraving, Opusmodus provides a third approach: **compositional programming** driven by its own notation script (OMN).

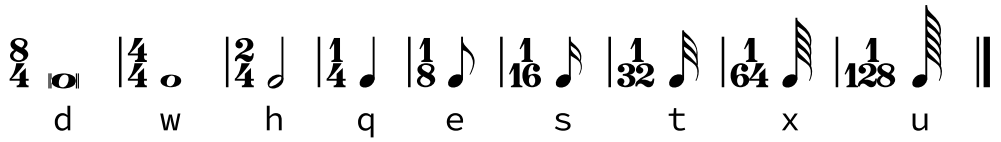
Iteration. OMN supports on-the-fly experiments: try an idea, audition it, and—if successful—retain it verbatim in the score. This continuity underpins an efficient composing process from sketch to final work.

13. The Four Elements in Detail

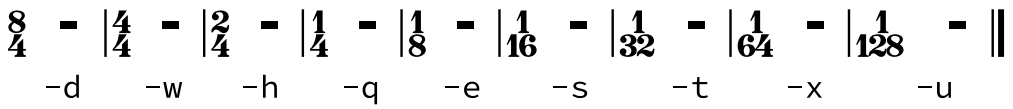
13.1 Length (first element)

In OMN, `length` encodes rhythmic **duration** using textual tokens that parallel staff notation. Length may appear alone (pure rhythm) or combined with pitch, velocity, and articulation.

Basic note-lengths. The common values use initial letters of American names:



Rests. A rest is the corresponding length prefixed with **–**:



Three quarter-notes:

```
'(q q q)
```

Note and rest-lengths combined:

```
'(q –e e q)
```

Bar counts. A leading integer repeats a bar-length:

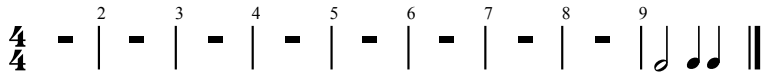
- **n** → *n* bars of whole-note lengths
- **-n** → *n* bars of whole-note rests

8 bars of **whole-note lengths**, then h q q:

```
'(8 h q q)
```

8 bars of **whole-note rests**, then h q q:

```
'(–8 h q q)
```



13.1.1 Dotted lengths

Symbol: `.` `..` `...` (up to three dots), as in traditional notation.

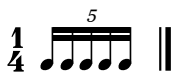
```
(q. q s e. e)
```

13.1.2 Tuplets

OMN’s base division is duple; tuplets borrow from the next binary unit. The integer **prefix** marks the tuplet:

3q (triplet of eighths in a quarter beat), 5q (quintuplet of sixteenths in a quarter beat), etc.

```
'(5q 5q 5q 5q 5q)
```

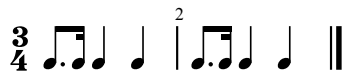


13.1.3 Repeat operators

Note-length: =
Rest-length: –

Repeats compact scripts of recurring patterns; widely used for percussion and ostinati.

```
'((e. s q =) (e. s q =))
```



If a **note-length repeat** follows a **rest-length**, a **note** of the same duration is produced:

```
'(s e – s – e. –s e s –e)
```

The use of the rest-length repeat can bring clarity to the visual layout of a phrase.

13.1.4 Compound lengths

A single event may contain **concatenated length elements** (compound duration):

```
'(hqs –s qe)
```

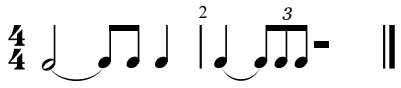
13.1.5 Ties

Two tie mechanisms are available:

- **Length tie symbol**: underscore _ joins notated lengths within a list.
- **Attribute tie**: **tie** joins across list boundaries (or when a symbolic **tie** is notationally inconvenient).

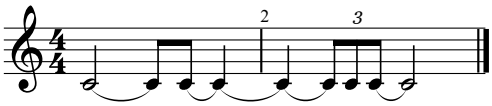
Length tie:

```
'((h_e e q) (q_3q 3q 3q –h))
```



Attribuite tie:

```
'((h_e c4 e_q tie) (q_3q c4 3q 3q_h))
```



13.1.6 Extended lengths

Note: ==
Rest: ––

Adjacent extenders lengthen the prior `value`, analogous to tying repeated units.

```
'(3q == –e = s – = – == ––)
```

13.1.7 Ratios

Durations may be written as **rational numbers**; ties work identically with ratios.

```
'((q = e = h) (q – e. t =))

;; same as
'((1/4 1/4 1/8 1/8 1/2) (1/4 –1/4 3/16 1/32 1/32))
'((q_e e q) (q –3q = = h))

;; same as
'((1/4_1/8 1/8 1/4) (1/4 –1/12 1/12 1/12 1/2))
```

13.1.8 Length–duration override

Symbol: caret ^

x^y keeps the **notated** value **x** but **plays** with the duration of **y** (independent of engraving).

s^t: notated sixteenth, performed as a thirty-second:

```
'(–e s g4 p c5 s^t e5 g4 c5 s^t e5)
```

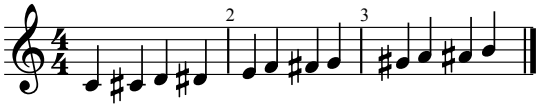
s^q: notated sixteenth, performed as a quarter:

```
'(s^q c4 s e4 g4 c5)
```

13.2 Pitch (second element)

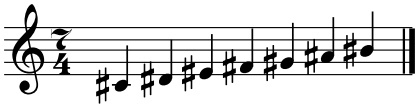
In OMN a pitch is a symbol combining the lower-case letter name with an octave number; c4 denotes middle C (the fourth C on an 88-key piano). OMN spans the MIDI range **0–127**, thus extending beyond the physical keyboard. Chromatic (sharps preferred ascending):

```
'(c4 cs4 d4 ds4 e4 f4 fs4 g4 gs4 a4 as4 b4)
```

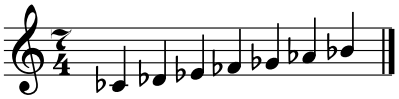


Sharps and flats. **s** marks **sharp**, **b** marks **flat**:

```
'(cs4 ds4 es4 fs4 gs4 as4 bs4)
```



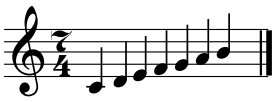
```
'(cb4 db4 eb4 fb4 gb4 ab4 bb4)
```



Enharmonic spellings may be mixed by OMN functions to suit diatonic context.

Diatonic scale (C major):

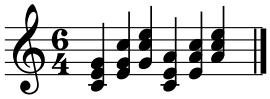
```
'(c4 d4 e4 f4 g4 a4 b4)
```



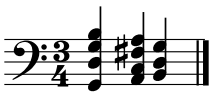
13.2.1 Chords

A chord is a single event containing a list of pitches; ordering within the list does not affect engraving.

```
'(q c4e4g4 e4g4c5 g4c5e5 c4e4a4 e4a4c5 a4c5e5)
```



```
'(q g2d3g3b3 a2c3fs3a3 b2d3g3)
```



13.2.2 Microtonality — symbols

Microtonal offsets attach to the pitch token:

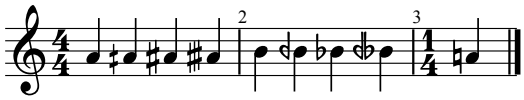
Token	Meaning
`+`	+1/4-tone
`-`	−1/4-tone
`.` with sharp	+1/8-tone
`..` with sharp	+3/8-tone
`.` with flat	−1/8-tone
`..` with flat	−3/8-tone
`+.` with (♯/♭)	±1/8-tone
`+..` with (♯/♭)	±3/8-tone
`-.` with (♯/♭)	∓1/8-tone
`-..` with (♯/♭)	∓3/8-tone

Numeric correspondences (for interval/integer domains):

`0.5` = 1/4-tone, `0.25` = 1/8-tone.

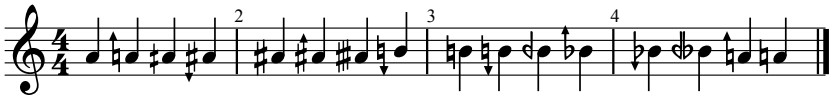
13.2.3 Quarter tones (examples)

```
'(q a4 a4+ as4 as4+ b4 b4- bb4 bb4- a4)
```



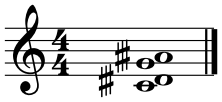
13.2.4 Eighth tones (examples)

```
'(q a4 a4. a4+ a4.. as4 as4. as4+ b4-. b4
    b4-. b4- b4-.. bb4. bb4- a4. a4)
```



13.2.5 Microtonal Chords

```
'(w c4ds4+g4as4+)
```



Chord with quarter-tone sharps on **D** and **A**.

13.2.6 Transposition (microtonal)

Fractional semitone steps allow microtonal transposition.

```
(pitch-transpose 2.5 '((a4 a4+ b4 d5) (e5 e5+ fs5 a5)))
=> ((b4+ c5 cs5+ e5+) (fs5+ g5 gs5+ b5+))
```

Generate a quarter-tone row with **RND-ROW**; the `:quantize` argument controls the step (`1/4` or `1/8`):

```
(setf mat (rnd-row :quantize 1/4 :type :pitch :seed 34))
=> (c4 g4 ds4+ cs4 d4+ b4+ f4+ a4+ gs4+ e4+ b4 as4+
    e4 bb4 cs4+ a4 d4 g4+ fs4+ eb4 c4+ gs4 fs4 f4)
```

13.2.7 Intervals

Convert between pitch and (possibly fractional) interval series:

```
(pitch-to-interval mat)
=> (7 -3.5 -2.5 1.5 9 -6 4 -1 -4 6.5 0.5 -6.5
    6 -8.5 7.5 -7 5.5 -1 -3.5 -2.5 7.5 -2 -1)

(interval-to-pitch '(1.5 2 -1 -1.5 1 .5 .5))
=> (c4 cs4+ ds4+ d4+ cs4 d4 d4+ eb4)
```

13.2.8 Quantising from frequency

Quantise frequency lists to the **nearest microtonal grid** (1/4-tone shown):

```
(setf hertz '(448 880 1320 1760 2200 2640 3212 3520))

(hertz-to-pitch hertz :quantize 1/4)
=> (a4+ a5 e6 a6 cs7 e7 g7+ a7)
```

Notes

In prose we use **quantise**; function names use `:quantize`.

Engraving follows the tokens supplied; playback accuracy depends on the active **sound set and device resolution**.

13.3 Velocity (third element)

Velocity encodes dynamic level. OMN provides twelve discrete tokens:

```
ppppp pppp ppp pp p mp mf f ff fff ffff fffff
```

Exact MIDI values are **sound-set dependent**; the mapping is monotonic from `ppppp` to `fffff`.

13.3.1 Dynamic modifiers

Symbols that mark **dynamic shaping** rather than absolute level:

`cresc`	progressive increase
`dim`	progressive decrease
`<`	hairpin crescendo
`>`	hairpin diminuendo
`0<`	from silence to a target level
`>0`	from a given level to silence

13.3.2 Crescendo family

Dynamic level **with attached crescendo** (**<**):

```
ppppp< pppp< ppp< pp< p< mp< mf< f< ff< fff< ffff<
```

13.3.3 Diminuendo family

Dynamic level with attached diminuendo (**>**):

```
pppp> ppp> pp> p> mp> mf> f> ff> fff> ffff> fffff>
```

13.3.4 Sforzando symbols

A non-exhaustive set (use those supported by your sound set / notation export):

```
pfp fpf pf fp ffp fffp sfp sfpp sfppp sffp sffp
sf sfff sfff sffff sfz sffz sfffz fz ffz fffz rf rfz
```

13.3.5 Single-note dynamic patterns

These notate **within-note** dynamic shapes (common in wind, brass, and strings, e.g., with mutes).

Let dyn ∈ {ppppp … `fffff`}.

Pattern schema

`0<dyn`	silence → *dyn*
`dyn>0`	*dyn* → silence
`0<dyn>`	silence → *dyn* → (release)
`<dyn>0`	attack → *dyn* → silence
`0<dyn>0`	silence → *dyn* → silence
`<dyn>`	attack → *dyn* → release
`>dyn<`	accent then relax to *dyn*
`dyn<>, dyn><`	two-phase swells around *dyn*

`dyn<dyn` , `dyn>dyn` , `dyn<>dyn` , `dyn><dyn`
start at *dyn*, shape, end at *dyn*

Minimal markers: `>0<`, `0<>0`, `<>0`, `0<>`, `<>`, `><`

13.4 Articulation (Attribute — fourth element)

In **OMN**, *articulation* denotes the vocabulary of performance **attributes** attached to events. Attributes modify **onset**, **duration**, **timbre**, and **dynamic shaping**. They may apply to a **single event** or persist as **sticky** indications until reset (e.g., by `ord`, `nat`, `non-trem`).

Organisation. For clarity the lexicon is grouped as follows:

- **General articulations** — accents, legato/tenuto/staccato, **caesura**, **fermata**, **ties**, **pedal** signs. These instruct how events are articulated or sustained and may govern a single note or a span.
- **Ornaments** — acciaccatura/appoggiatura, **arpeggio**, **glissando/portamento**, **mordent**, **trill**, **tremolo** (including two-note tremolo), **turn**. Ornaments often introduce **additional pitches** and **redistribute durations**.
- **Marks** — structural and rehearsal indications (e.g., **repeat** signs, **rehearsal marks**) that annotate the score rather than change sound generation directly.

Instrument-specific sets

- **Woodwind/Brass.** Contemporary techniques (e.g., **flutter-tongue**, **multiphonics**, air-noise variants, mouthpiece effects, **stopped/open** mute states) are available as attributes. Through **DEF-SOUND-SET** they can be mapped to libraries so that printed indications also **select the appropriate samples**.
- **Strings.** A comprehensive set (e.g., **arco**, **pizz**, **col legno**, **harmonics**, **sul ponticello/tasto**, bowing and vibrato variants) supports layered and mixed timbres when a compatible library is configured. Attributes may be combined (e.g., `pizz+trem`) and later normalised with `ord`.

Attributes occupy the **articulation slot** (forth element) in **OMN** and can be combined with **+**.

Sound realisation. Printed attributes are rendered in notation; audible realisation depends on the active **sound set** and device routing. Where no mapping exists, attributes still function as **engraving semantics**.

13.4.1 Appendix — OMN Articulations

Accents
det marc mart stacc stacs ten

Grace Note
–acc –acc–e –acc–h –acc–q –acc–s –acc–t –acc–x –acc.
–app –app–e –app–h –app–q –app–s –app–t –app–x –app.
acc acc–e acc–h acc–q acc–s acc–t acc–x acc.
app app–e app–h app–q app–s app–t app–x app.

Bow
dbow ubow

Strings Articulations
alto–ponte alto–tasto arco arco–lento arco–ord arco–ponte
arco–tasto arm batt con–vib crini da–ponte div espr
extr–ponte extr–tasto flaut gettato jete knock legno
legno–batt legno–tratto lh–pizz lh–slap molto–ponte
molto–tasto molto–vib non–arm non–vib pizz pizz–chit
pizz–nail pizz–ord pizz–trem poco–ponte poco–vib ponte
ponte–tasto ponte–tasto–ponte punta ric secco senza–vib
slap snap soli solo spicc sulla–corda tallone tap tasto
tasto–ponte tasto–ponte–tasto tutti tutto–arco unis vib
vib–norm vib–ord

Open String
sul sul1 sul2 sul3 sul4 sul5 sula sulc suld sule sulg

Muting
con–sord mute open senza–sord unmute via–sord

Harmonic
harm harm2

Harp
bisb clang close–to–table dampened fingernail
hand–on–the–corpus hand–on–the–strings hit
knuckle–on–the–corpus pedal–noise semitone–downwards
semitone–upwards thin–pick tuning–wrench wholetone–downwards
wholetone–upwards xylophone–tone

Arpeggio
arp arp–adlib arp–down arp–up

Caesura
caesura

Fermata
bl-fermata bl-fermata-l bl-fermata-s bl-fermata-vl
bl-fermata-vs fermata fermata-l fermata-s fermata-vl
fermata-vs

Finger
dig1 dig2 dig3 dig4 dig5

Tongue
frull tong-blocked tong-hard tong-soft tong1 tong2 tong3

Ending
end1 end2 end3 end4 end5 end6 end7 end8 end9 end10

General Pause
gp lp

Glissando
gliss gliss2 gliss3 gliss4 kgliss kgliss-ch

Legato
leg

Tie
tie

Brass and Wind
air-noise-f air-noise-h air-noise-k air-noise-p air-noise-s
air-noise-sh air-noise-t finger-damp flutter-tongue
half-depressed-valves harsh-blow high-noise-blow
hit-on-mouthpiece hum breathy insert-straight-mute-into-bell
kiss low-noise-blow mouthpiece-backwards mouthpiece-only
multiph over-blow play-and-sing silent-brass
snap-with-a-finger-on-the-bell stop-mute-closed
stop-mute-open stop-mute-wahwah-effect under-blow
without-air without-mouthpiece without-tubings

Barlines
bl-da bl-do bl-h bl-hh bl-hl bl-lh bl-ll bl-none bl-norm
bl-s

Hand
lh md ms rh

Mordent Lower Double
ldmordent1 ldmordent1-t ldmordent1-x ldmordent2 ldmordent2-t ldmordent2-x

Mordent Lower
lmordent1 lmordent1-t lmordent1-x lmordent2 lmordent2-t lmordent2-x

Mordent Upper Double
dmordent1 dmordent1-t dmordent1-x dmordent2 dmordent2-t dmordent2-x

Mordent Upper
mordent1 mordent1-t mordent1-x mordent2 mordent2-t mordent2-x

Cue Notes
cue

Number
num0 num1 num2 num3 num4 num5 num6 num7 num8 num9 num10 num11

Pedals
half-ped half-ped1 ped ped1 sost-ped sost-ped1 una-corda una-corda1

Rehearsal Marks
reh reha reha-<a-z> rehn rehn-<1-1000>

Comma
comma

Repeat
repeat

Octave Shifts
15ma 15mb 8va 8vb

Reset
adlib nat non-trem norm ord sim

Notehead
nh-bsld nh-cx nh-d nh-none nh-norm nh-sld nh-x

Two-Note-Tremolo
ttrem ttrem–3e ttrem–3s ttrem–5e ttrem–5q ttrem–7e ttrem–7q ttrem–e ttrem–s ttrem–t ttrem–x

Tremolo
trem trem–3h trem–3q trem–3e trem–3s trem–5h trem–5q trem–5e trem–5s trem–7h trem–7q trem–7e trem–7s trem–e trem–s trem–t trem–x

Trill
ltr1 ltr1–3e ltr1–3s ltr1–5e ltr1–5q ltr1–7e ltr1–7qnltr1–s ltr1–t ltr1–x ltr2 ltr2–3e ltr2–3s ltr2–5e ltr2–5q ltr2–7e ltr2–7q ltr2–s ltr2–t ltr2–x tr1 tr1–3e tr1–3s tr1–5e tr1–5q tr1–7e tr1–7q tr1–s tr1–t tr1–x tr2 tr2–3e tr2–3s tr2–5e tr2–5q tr2–7e tr2–7q tr2–s tr2–t tr2–x

Turn Lower Classic
lcturn11 lcturn11–5e lcturn12 lcturn12–5e lcturn21 lcturn21–5e lcturn22 lcturn22–5e

Turn Lower
lturn11 lturn11–s lturn11–x lturn12 lturn12–s lturn12–x lturn21 lturn21–s lturn21–x lturn22 lturn22–s lturn22–x

Turn Upper Classic
cturn11 cturn11–5e cturn12 cturn12–5e cturn21 cturn21–5e cturn22 cturn22–5e

Turn Upper
turn11 turn11–s turn11–x turn12 turn12–s turn12–x turn21 turn21–s turn21–x turn22 turn22–s turn22–x

13.4.2 User-defined text attributes

Extend the OMN attribute vocabulary with your own **textual techniques**, either for one-off use or as persistent (sticky) indications.

Definition. Use **ADD-TEXT-ATTRIBUTES** with one or more *(symbol 'printed text')* pairs. Place the form in `~/Opusmodus/User Source/Extensions/User Function.lisp` to **load at start-up**.

```
(add-text-attributes
  '(ord-tasto "ord⟶tasto")
  '(tasto-ponte "sul tasto⟶pont.")
  )
```

Use. The new attributes behave like built-ins and can be combined with others using the **+** operator.

```
'(h c4 ppp ord-tasto+leg cs5 p tasto-ponte)
```

Stickiness. Apply per-event (**non-sticky**) or, following normal OMN practice, use the **sticky** form to persist until changed (e.g., `ord-tasto+` … followed later by `ord/nat` or another technique).

```
(add-text-attributes
'(m1 "m1" :non-sticky)
'(m2 "m2" :non-sticky)
'(m3 "m3" :non-sticky)
'(am-steg "am steg")
'(ruhig "ruhig")
)
```

14 Definitions

14.1 Argument and Value

In Common Lisp, the terms *`arguments`* and *`values`* are central to understanding how functions work. An *`argument`* is the input given to a `function`, while a *value* is the output that a function produces. When you call a `function`, you pass in *`arguments`*, and when the `function` execution completes, it returns a *`value`* or *`values`*.

14.1.1 Arguments

`Arguments`, or *`parameters`*, are the inputs a function accepts when invoked. When defining a function, you specify a list of *parameters* it expects to receive. These *`parameters`* serve as placeholders for the actual *`values`*, called *`arguments`*, that will be supplied when the `function` is invoked.

```
(defun add (x y)
  (+ x y))

(add 3 4)
```

In this case, *x* and *y* are *`parameters`* defined in the **ADD** function, and 3 and 4 are the *`arguments`* provided when the `function` is called.

14.1.2 Values

A *`value`* refers to the result a `function` yields when it's invoked. In Common Lisp, every function returns at least one *value*, but it can also optionally return multiple *`values`*.

For instance, in this example, the **ADD** function yields the *`value`* 7:

```
(add 3 4)
=> 7
```

This indicates that the **ADD** `function`, given the *`arguments`* 3 and 4, performs an addition operation and returns the `sum`, which is 7. This returned *`value`* can be used for subsequent computations or as the final output. The ability to return *`values`* is a fundamental aspect of `functions` in Opusmodus because it enables the results of computations to be utilised in various ways within a program.

14.2 Attribute

In the context of the Opusmodus Notation (**OMN**), an *attribute* refers to the *articulation* or *expressive element* of a musical *event*. It is the **fourth element** in an **OMN** *event* term, structured as follows: (l p v **a**).

Attributes encompass a wide variety of symbols and words that denote different forms of musical expression. These terms are broadly divided into two categories:

14.2.1 Articulations, Ornaments, and Marks

These apply to all instrumental and vocal performances and include aspects such as *accents*, *cesura*, *fermata*, *legato*, *ties*, and *pedal* instructions. In musical performance, these symbols provide guidance on controlling the dynamic, intensity, or duration of a musical *event*.

Ornaments, while closely related to articulation, typically involve the addition of further pitches and corresponding changes in duration. In **OMN**, ornaments encompass *acciaccatura*, *appoggiatura*, *arpeggio*, *glissando*, *mordent*, *trill*, *tremolo*, *turn*, and *two-note tremolo*. Once an essential skill for musicians, ornamentation in the 17th and 18th centuries frequently exceeded what was indicated in a score. In contemporary music, while the symbols from Baroque performance practice are still utilised, *ornamentation* is usually a precise and obligatory requirement to intensify individual pitched *events*.

14.2.2 Specific Performance Indicators

These are unique to certain instruments, including strings, woodwind, and brass. They provide instructions on specific ways to play these instruments, such as *bowing techniques*, *pizzicato*, and specific timbre types for strings, or particular breath and embouchure techniques for woodwind and brass.

In OMN, woodwind and brass performance instructions encompass terms common in contemporary scores, initially compiled in Bruno Bartolozzi's *'New Sounds for Woodwind'*. They can be linked through **DEF–SOUND–SET** to sample sets of woodwind and brass attack transients.

Performance instructions for strings in **OMN** are extensively cited. In conjunction with a sample library, they can trigger complex mixtures and layers of timbre type. The instructions include *Arco*, *Pizzicato*, *Col Legno*, *Harmonics*, *Sul Ponticello*, *Sul Tasto*, *Bowing Techniques*, and *Vibrato*. In the Opusmodus

Notation (**OMN**), various *events* are used to illustrate the implementation of *attributes*.

Here are a few examples:

```
'(e c4 – q cs4 fermata – fermata-l c4)
```

In this example, a quarter note with a pitch of cs4 is played with a fermata *articulation*, which typically means the note should be sustained longer than its normal duration.

```
'(q bb2 gliss e eb3 eb3 gliss h bb2)
```

Here, a *glissando*, denoted as gliss, is applied to a quarter note with a pitch of bb2, transitioning smoothly to an eighth note with a pitch of eb3. The next *glissando* starts on the next eighth note with the same pitch before ending on a half note with a pitch of bb2.

A noteworthy feature in **OMN** is the use of the '**+**' symbol. This symbol is used to merge two or more *attributes* into a single *articulation*:

```
'(q bb3 pp pizz+ten – gs3e4 pp<> ubow+arco)
```

Here, the *pizzicato* and *tenuto articulations*, represented as pizz and ten, are combined for the quarter note with a pitch of bb3. Similarly, the *up bow* and arco *articulations*, represented as ubow and arco, are combined for the duration of pitches gs3e4, played with varying dynamics. This feature enables the creation of complex and nuanced expressions within a musical piece.

14.3 Event

In the Opusmodus Notation (**OMN**), an *event* refers to a musical entity made up of four key elements: `(<length><pitch><velocity><attribute>)`, presented in that specific order. Just as in traditional music notation, **OMN** represents key musical `elements`, including `rhythm` and `pitch`. For instance:

```
q          - symbolises a quarter note (length),
c4         - represents the pitch (C in the 4th octave),
mp         - indicates a medium soft dynamic level (velocity),
stacc      - signifies staccato (attribute).
```

Let's examine some examples that illustrate *events* composed of varying numbers of these *`elements`*:

```
2-element event: (q c4)           - length and pitch,
3-element event: (q c4 f)         - length, pitch, and velocity,
4-element event: (q c4 f stacc)   - length, pitch, velocity, and attribute.
```

In **OMN**, a *`rest`* (or pause) is also considered an *event*. A *single `element event`* is always a *`rest`* with the *`length`* of the *`rest`* defined. For instance, `(-q)` represents a **quarter note rest**. There are exceptions, such as a *2-element event* `(-q fermata)`, which does not correspond to the typical `(<length><pitch>)` structure.

Here are a few examples of functions in Opusmodus that work with *events*: The **`SINGLE-EVENTS`** function converts an **OMN sequence** into a *`sequence`* of *`single events`*:

```
(single-events '(s bb3 leg a3 mp leg g3 a3 f c4 f4 a4))
=> ((s bb3 mf leg) (s a3 mp leg) (s g3 mp)
    (s a3 f) (s c4 f) (s f4 f) (s a4 f))
```

The **`FILTER-EVENTS`** function selects all occurrences of a specified `pitch` from a given **OMN sequence**:

```
(filter-events 'd4 '(e c4 mp -e fermata e. d4 -h e. c4 e e4))
=> (-q e. d4 mp -h.s)
```

14.4 Expression

In Common Lisp, an *expression* is a combination of symbols and operators that, when evaluated, yields a *value*. Expressions can range from simple ones, like a *single literal* or *variable*, to more *complex* ones, like nested function calls or mathematical operations. In Lisp, *expressions* are often referred to as forms or s-expressions.

The following are the primary types of *expressions* in Common Lisp:

14.4.1 Atoms

An atom is a simple, indivisible entity. Atoms can be *numbers*, *strings*, *symbols* (which include *variable* and function names), keyword *symbols*, and the special *values* **NIL** and **T**.

Examples of atoms:

```
5               ; a numeric atom
"Hello"         ; a string atom
t               ; a symbol representing true
```

Test Case

```
(setf pitch 'c4)
(atom pitch)
=> t
```

14.4.2 Lists

Lists are ordered collections of other *expressions*, enclosed within parentheses. In Lisp, lists serve as the fundamental *data* structure and represent function calls and other compound *expressions*. The *first element* of a list is usually a function name, which is followed by its *arguments*.

Examples of lists:

```
(+ 1 2)         ; a list representing a function call
(list 1 2 3)    ; a list representing a function call
                ; to create a list (1 2 3)
```

14.4.3 Quoted Expressions

A quoted *expression* is an `atom` or a `list` preceded by **a single quotation mark** (`'`), which prevents the Lisp interpreter from evaluating the *expression*. Instead, the *expression* itself is returned.

Examples of quoted expressions:

```
'(+ 1 2)        ; a quoted list
'c4             ; a quoted pitch symbol
```

14.4.4 Special Forms

`Special forms` resemble `function` calls but exhibit unique behaviour that cannot be replicated by ordinary `functions`. For example, **`IF`** is a `special form` that evaluates one of its two main *`arguments`*, depending on the truth *`value`* of its condition *`argument`*.

Example of a `special form`:

```
(if (> 1 2) 'wrong 'right)
```

In this example, since the condition `(> 1 2)` is *`false`*, `'wrong` will not be evaluated. Instead, `'right` will be returned because it's the alternative outcome when the condition is not met.

14.5 Floating-Point Number

In Common Lisp, a *floating-point number* represents a *real number* that can be either *rational* or *irrational*, *positive*, *negative*, or *zero*. *Floating-point numbers* are used when a higher degree of precision is required beyond what *integers* can provide.

Floating-point numbers are generally represented by a digit string, followed by a decimal point, followed by another digit string, for example, 0.5. An exponent part may follow the decimal point part, introduced by an '**E**', such as 1.31E-4.

Arithmetic operations similar to those for *integers* can also be performed on *floating-point numbers*.

For instance, (+ 1.5 2.3) returns 3.8.

Common Lisp provides several built-in mathematical functions that operate on *floating-point numbers*, including *trigonometric* functions, *logarithmic* functions, among others. *Floating-point numbers* can be either of *single* or *double precision*, although the actual precision is dependent on the implementation.

Common Lisp also provides functions to manipulate *floating-point numbers*. For instance, the **FLOAT** function coerces its *argument* to a *floating-point number* of the same format as the second *argument* or a *single float* if only one *argument* is supplied.

For example, (float 1) returns 1.0.

The **FLOATP** function checks if its *argument* is a *float*, returning *true* if it is and *false* otherwise.

For example, (floatp 0.5) returns **T**.

In Opusmodus, *floating-point numbers* can be converted into musical *elements* such as *pitch*, *length*, and *velocity* values. Consider the following examples:

First, we generate a set of *floating-point numbers*:

```
(setf noise (gen-noise 12 :seed 45))
=> (0.3392996 0.19626139 0.81495554 0.83888537 0.7668492
    0.7997687 0.55251557 0.16572258 0.95386976 0.95762116
    0.41804564 0.1733811)
```

These *numbers* can then be mapped to various musical *elements*:

14.5.1 Length

```
(vector-to-length 1/16 -1 3 noise)
=> (-1/16 1/8 1/8 1/8 1/8 1/16 -1/16 3/16 3/16 -1/16)

(vector-map '(s -s e) noise)
=> (s s e e e e -s s e e -s s)

(quantize noise '(1 2 3 4))
=> (3q 3q 3q_3q 3h_s e. e. s_3q 3q 3q_e q e s)
```

14.5.2 Pitch

```
(vector-to-pitch '(g3 b4) noise)
=> (eb4 c4 c4 b4 b3 eb4 g3 gs3 e4 d4 bb4 d4)

(vector-map '(c4 eb4 fs4 a4) noise)
=> (eb4 c4 fs4 a4 fs4 fs4 eb4 c4 a4 a4 eb4 c4)
```

14.5.3 Velocity

```
(vector-to-velocity 0.27 0.87 noise)
=> (p pp fff fff ff ff mf pp ffff ffff mp pp)

(vector-map '(pp mf f) noise)
=> (pp pp f f f f mf pp f f mf pp)
```

In each case, the set of *floating-point numbers* serves as a foundation for generating musical *elements*, thereby providing a way to create intricate and varied outputs from simple initial *data*.

14.6 Optional and Keyword

`Function` definitions in Opusmodus significantly leverage `Optional` and `Keyword` *`parameters`*. These *parameters* provide extensive flexibility, enabling a broad range of transformations and modifications to be applied to input *`data`*, thereby maximising the potential for creative *`variations`* in musical composition.

`Optional` and `Keyword` have specific meanings:

14.6.1 Optional

In Opusmodus, optional refers to *parameters* in a function definition that are not required when calling the function. If they are not provided, they take on default *values*. These optional *parameters* are declared using the &optional keyword in the function definition.

For example, in the function **`GEN-INTEGER`**, *a* is a required *parameter* while *b* and *step* are optional:

`GEN-INTEGER` (*`a`* `&optional` *`b step`*)

With only the required *`parameter:`*

```
(gen-integer 12)
=> (0 1 2 3 4 5 6 7 8 9 10 11 12)
```

With the optional `parameter b`:

```
(gen-integer -3 12)
=> (-3 -2 -1 0 1 2 3 4 5 6 7 8 9 10 11 12)
```

With both optional `parameters` *`b`* and *`step`*:

```
(gen-integer -3 12 2)
=> (-3 -1 1 3 5 7 9 11)
```

14.6.2 Keyword

In Opusmodus, a keyword is a special kind of symbol and has its own distinct *object* type. Keywords are used in several ways, one of which is to specify keyword *parameters* in a function. Keyword *parameters* are introduced by the &key keyword in the function definition. Unlike *positional parameters*, keyword *parameters* are identified by *name* in the function call. They can also be provided default *values*.

In the function **RND-AIR**, :group, :type, and :seed are keyword *parameters*:

RND-AIR (&key *group type seed*)

With :seed as a keyword *parameter*:

```
(rnd-air :seed 45)
=> (0 1 8 5 7 10 9 3 11 4 2 6)
```

With both :type and :seed as keyword *parameters*:

```
(rnd-air :type :pitch :seed 45)
=> (c4 cs4 gs4 f4 g4 bb4 a4 eb4 b4 e4 d4 fs4)
```

14.7 Parameter

In Opusmodus, a *parameter* is a type of *variable* used in the definition of a function. *Parameters* serve as placeholders for the actual *values*, also known as *arguments*, which are supplied when the function is called. During the function call, these *parameters* are bound to the specific *arguments* provided, and these bindings persist for the duration of the function call.

Consider this simple example of a function definition:

```
(defun add (a b)
  (+ a b))
```

In this case, *a* and *b* are *parameters* of the **ADD** function. These *parameters* will take on the *values* of the *arguments* provided when the function is called.

For example:

```
(add 2 3)
=> 5
```

In this function call, 2 is the *argument* that gets bound to the *parameter a*, and 3 is the *argument* that gets bound to the *parameter b*. The function then performs the operation of *addition* on these *arguments* and returns the result, 5.

14.8 Predicate

In the context of the Common Lisp programming language, a *predicate* refers to a function that tests for a specific condition and returns a *Boolean value*: **T** (*true*) or **NIL** (*false*). *Predicates* serve a pivotal role in conditional statements, filtering operations, and any scenario where a *true* or *false condition* is required. *Predicates* are typically identifiable by the naming convention, which appends a '**p**' (short for *predicate*) at the end of the function name.

Here are some examples of *predicates* in Common Lisp:

Checking if a number is even:

```
(defun even-numberp (n)
  (zerop (mod n 2)))
```

Test Cases

```
(even-numberp 1)
=> nil

(even-numberp 2)
=> t
```

Checking if a list is empty:

```
(defun empty-listp (list)
  (null list))
```

Test Cases

```
(empty-listp '(1 2 3))
=> nil

(empty-listp '())
=> t
```

Checking if a number is positive:

```
(defun positive-numberp (n)
  (> n 0))
```

Test Cases

```
(positive-numberp 1)
=> t

(positive-numberp -1)
=> nil
```

In each of these examples, the `function` checks for a certain *condition* (e.g., is the number `even`, is the `list` empty, is the number `positive`) and returns a *`Boolean value`* (**`T`** for *true*, **`NIL`** for *false*) depending on whether the *condition* is met.

```
(loop for i in '(0 1 2 3 4 5 6 7 8 9 10 11)
      when (even-numberp i)
        collect i)
=> (0 2 4 6 8 10)
```

In this example, a **LOOP** is used to *iterate* over the *`list`* of *`numbers`* `(0 1 2 3 4 5 6 7 8 9 10 11).` For each *`iteration`*, it checks if the *`number`* is `even` by invoking the *`predicate`* `function` **EVEN-NUMBREP** on the current *`number`* (represented by **`i`**). If the *`number`* is `even` (i.e., the *`predicate`* `function` returns **`T`**), the *`number`* is collected into a *`list`*. The **LOOP** continues until it has processed all *`elements`* in the *`list`*. The result is a *`list`* of *even numbers*, (0 2 4 6 8 10), as extracted from the original *`list`* using the **EVEN-NUMBREP** *`predicate`*.

14.9 Rational Number

A *rational number* is any number that can be expressed as a fraction of *two integers*, with a *denominator* that is not *zero*. All *integers* are considered *rational numbers* because each can be represented as a *fraction* with the **DENOMINATOR** as 1.

Common Lisp simplifies each *rational number* to its lowest terms, meaning it automatically reduces *fractions* to their simplest form by dividing both the *numerator* and the *denominator* by their *greatest common divisor* (**GCD**). For example, if you input 8/4, the system will simplify it to 2.

Just like *integers* and *floating-point numbers*, you can perform arithmetic operations on *rational numbers*:

```
(+ 1/2 1/3)
=> 5/6
```

You can use the **NUMERATOR** and **DENOMINATOR** functions to retrieve the *numerator* and *denominator* of a *rational number*, respectively:

```
(numerator 3/4)
=> 3

(denominator 3/4)
=> 4
```

Common Lisp provides the **RATIONALP** function, which checks if its *argument* is a *rational number* (either an *integer* or a *ratio*), returning *true* or *false* as appropriate:

```
(rationalp 1/2)
=> t

(rationalp 1.5)
=> nil
```

In Opusmodus, *rational numbers* are often used to represent the *length* (duration) of a note:

```
'(-1/16 1/8 1/8 1/8 1/8 1/16 -1/16 3/16 3/16 -1/16)
```

14.10 Variable

In Opusmodus, a *variable* serves as a symbolic name tied to a specific *value*. It is a fundamental component of programming, as it designates a named location in memory to store a *value*. This stored value can be modified throughout the execution of a program, hence the term *variable*.

The function **SETF** in Opusmodus is used to assign a *value* to a *variable*. In the given example, **SETF** assigns the outcome of the function (gen-brownian-motion 128) to the *variable data*.

```
(setf data (gen-brownian-motion 128))
```

In the first line, **SETF** is used to assign a list of pitch symbols to the *variable* pitches. Each *pitch* symbol is a text representation of a musical note, where the letter represents the note and the number represents the octave.

```
(setf pitches '(c4 d4 fs4 e4 a4 g4 cs4 gs4 eb4 b4 f4 bb4))
```

In the second line, **SETF** is used to assign a list of duration symbols to the *variable* lengths. Each symbol (like '**s**' for sixteenth note and '**e**' for eighth note) represents a particular duration or *length* of a musical note.

```
(setf lengths '(s e s e s s e s s s s e))
```

In the third line, **SETF** is used to assign the result of the function call (make-omn :length lengths :pitch pitches) to the *variable* omn. The **MAKE-OMN** function combines the *pitches* and *lengths* into a single *list* that represents a *sequence* of musical *events*.

```
(setf omn (make-omn :length lengths :pitch pitches))
```

Overall, this example showcases how *variables* can be used to store and manipulate musical *data* in Opusmodus.

15 Index