Accessibility in MuseScore

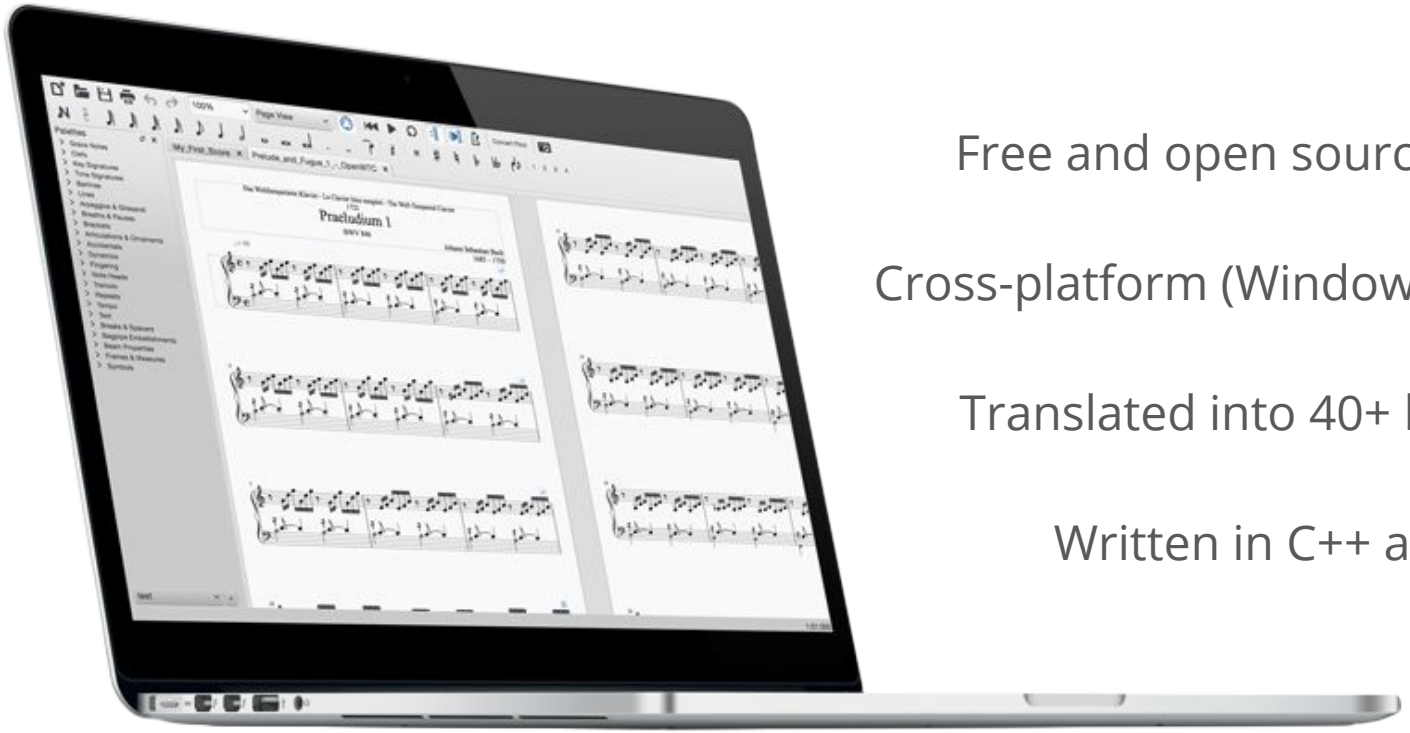
Our experience with Qt and QML

Speakers: Peter Jonas & Marc Sabatella

FOSDEM 2020

musescore

The world's most popular music notation program!



Free and open source (GPL v2)

Cross-platform (Windows, Mac, Linux)

Translated into 40+ languages

Written in C++ and Qt

Musescore.com – home to the largest online community of sheet music creators

Qt

- Pronounced “cute”
  - But often called “Q.T.” (or “cutie”?)
- Cross-platform UI Toolkit
  - Enables apps to run on Windows, Mac & Linux
- Open source
  - GPL 2.0, 3.0 & LGPL 3.0 (commercial licenses available)
- Traditional C++ Framework
  - Signals & slots (event handling)
  - Widgets (ready-made native-looking UI elements)
- Modern QML Language
  - Javascript-based declarative programming & dynamic property system
  - QtQuick (building blocks to create custom UI elements)

Accessibility Concepts

Accessibility

- Inclusiveness
  - Designing for everyone
  - In particular, for people who struggle to use a traditional interface:
    - **Input:** mouse + keyboard
    - **Output:** monitor + speakers
- Everybody is different!
  - Different abilities in terms of:
    - Sight
    - Hearing
    - Movement
    - Concentration
    - Perception
    - Understanding
- 1 in 5 people have a disability
- Technology is an essential part of modern life
  - Work, shopping, bills, taxes, healthcare, dating, entertainment...

Our focus

Accessibility for blind people

- Why focus on this topic?
  - **More difficult** to implement
    - Completely different interaction methods
    - Requires abstract thinking
  - **More rewarding** to implement
    - Changes are useful for tackling other accessibility problems
    - Leads to better overall design, even for sighted users!

Assistive devices

Screen readers

- Program that runs in the background on user's machine
- Synthesised voice describes what happens on the computer
  - *"Load button: load a file from your hard drive. To click this button, press Space."*
  - *"Play checkbox, not checked. Playback of selected element disabled. To enable, press Space."*
- Experience like a telephone menu system
  - *"Press 1 for Sales, 2 for Marketing, 3 for Support"*
- Not able to "see pixels"
  - Developers must use accessibility API to expose UI elements to the screen reader
- Users able to adjust voice
  - Speed, pitch, verbosity, etc.

Popular screen readers

Platform	Screen reader	Cost	Shortcuts
Linux (GNOME)	Orca	Free (built-in, open source)	Turn on/off: Alt+Super+S Orca: Insert / Capslock
macOS	VoiceOver	Free (built-in, proprietary)	Turn on/off: Cmd+F5 VO (VoiceOver): Ctrl+Option
Windows	Narrator	Free (built-in, proprietary)	Turn on/off: Ctrl+Win+Enter Narrator: Capslock
	NVDA	Free (open source)	NVDA: Insert / Capslock Turn off: NVDA+Q Toggle speech: NVDA+S
	JAWS	Paid (proprietary)	JAWS: Insert / Capslock Turn off: JAWS+F4 Toggle speech: JAWS+Space, S

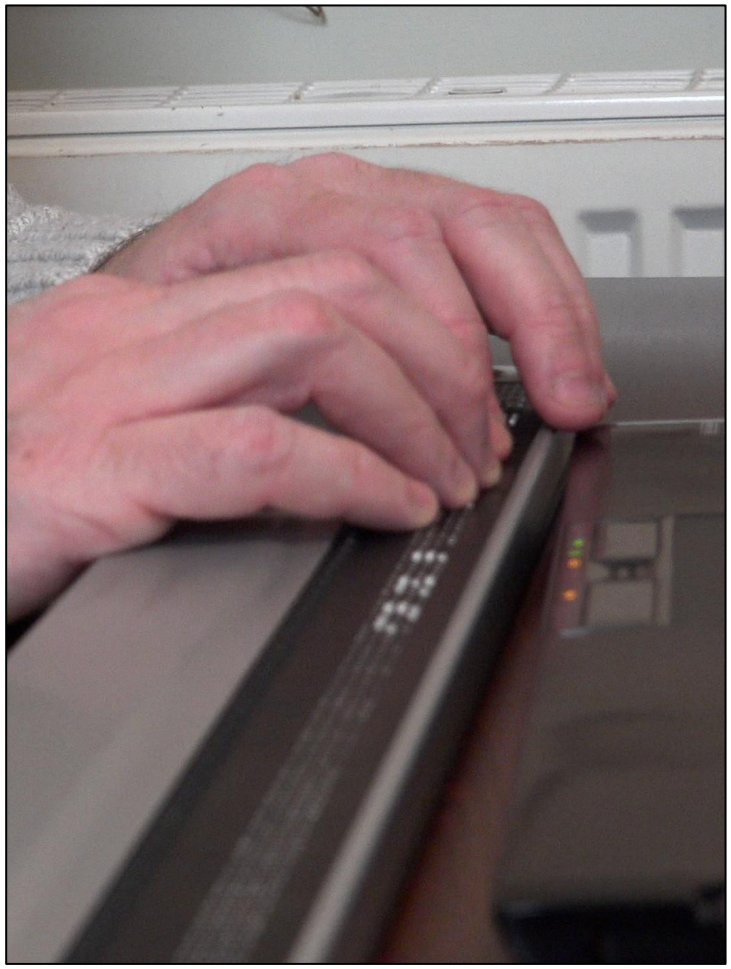
All screen readers have a shortcut named after themself. This key is used to send other keyboard strokes to the screen reader instead of the application (e.g. Orca+H activates Orca's help mode).

On Windows: JAWS has most users, NVDA works best with Qt (in our experience), Narrator is improving rapidly and gives the best overall experience, but it doesn't always work.

Assistive devices

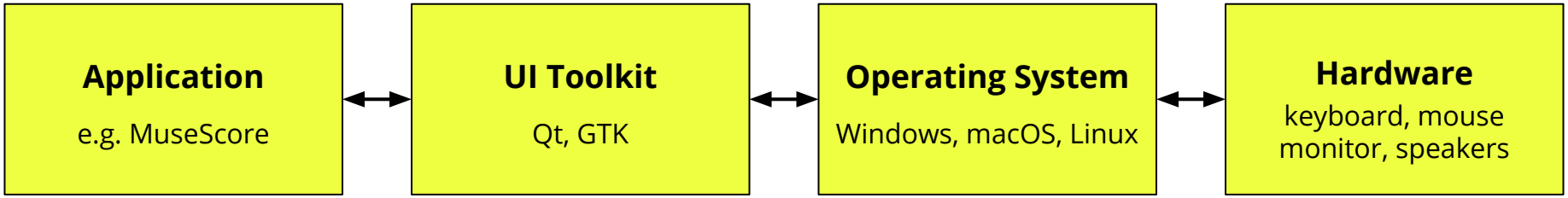
Braille terminals / Refreshable Braille displays

- Single row of Braille cells
- Raised dot pattern in each cell represents one letter/digit
- Alternative to screen readers
  - PROS
    - Silent
    - Able to skip or repeat text
    - Works for deafblind people
  - CONS
    - Not built-in to laptops
    - Must learn Braille
    - Expensive ($1000+ USD)
- Not needed by developers
  - You can use a screen reader for testing
  - Ask blind users to test on their terminals
  - Use a virtual Braille display (e.g. NVDA Braille Viewer)



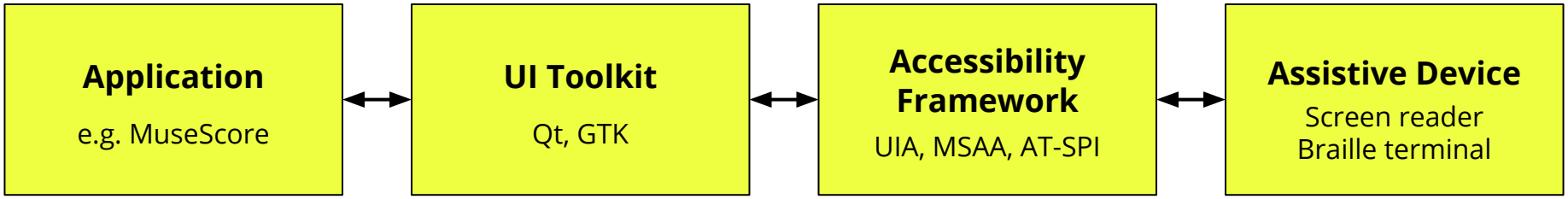
Accessibility API

Traditional model



UI Toolkit serves as an abstraction layer. App developers don't have to deal with OS and hardware specifics.

Accessibility model



UI Toolkit again takes care of platform specifics.

PRO: No need to import additional libraries.

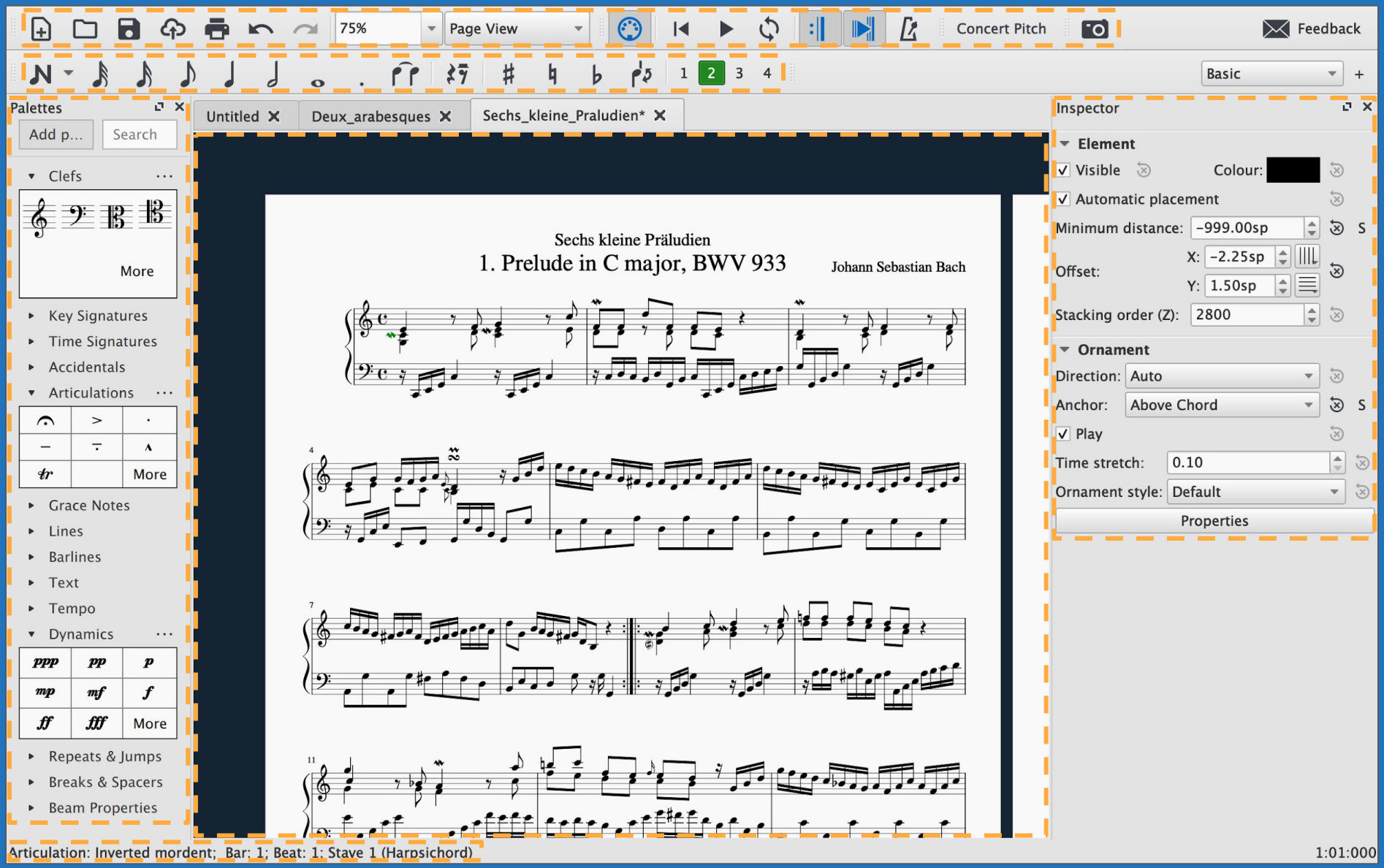
CON: Problems can occur in any layer (difficult to debug).

MuseScore Tour

Main Toolbar
Load & save scores, playback controls

Score View
Displays current score

Note Input Toolbar
Add notes, rests, accidentals, ties



Palettes
Add elements (musical symbols)

Status bar
Information about selected element

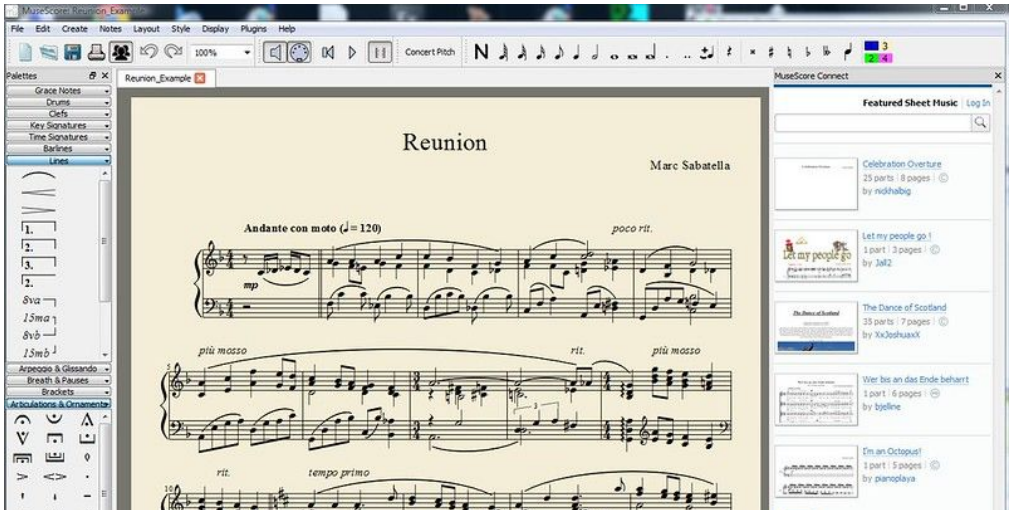
Inspector
Modify selected element properties

Accessibility Timeline

Accessibility in MuseScore 1

- No particular thought given to accessibility
- Standard Qt widgets are nominally accessible by default
- Our usage of standard widgets not optimized for accessibility
- Custom widgets such as score view and palette not accessible at all

MuseScore 1: blind and low vision users unable to use

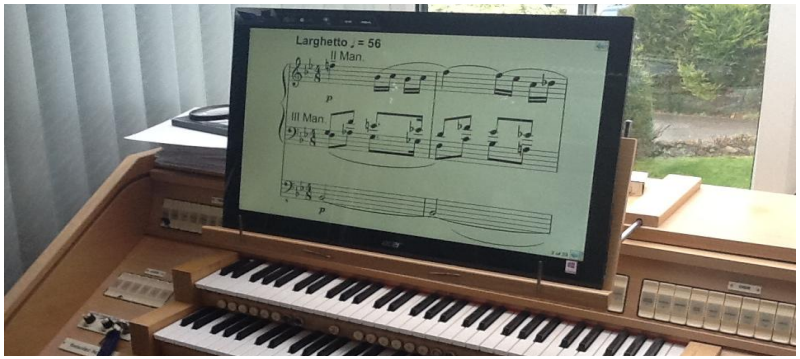


Accessibility in MuseScore 2

Initial accessibility work (in collaboration with RNIB, GSoC, OpenScore)

- Producing Modified Stave Notation for low-vision readers
- Improved accessibility of standard widgets
- Enhanced keyboard navigation of score view
- Implemented screen reader support for score view (NVDA only)
- Still no accessibility for palettes

MuseScore 2: blind and low vision users can read scores

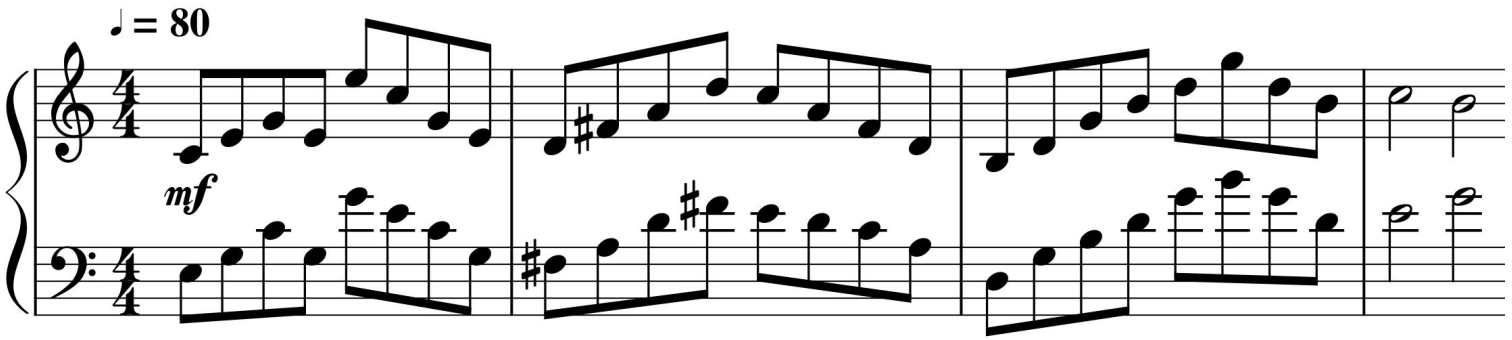


Accessibility in MuseScore 3

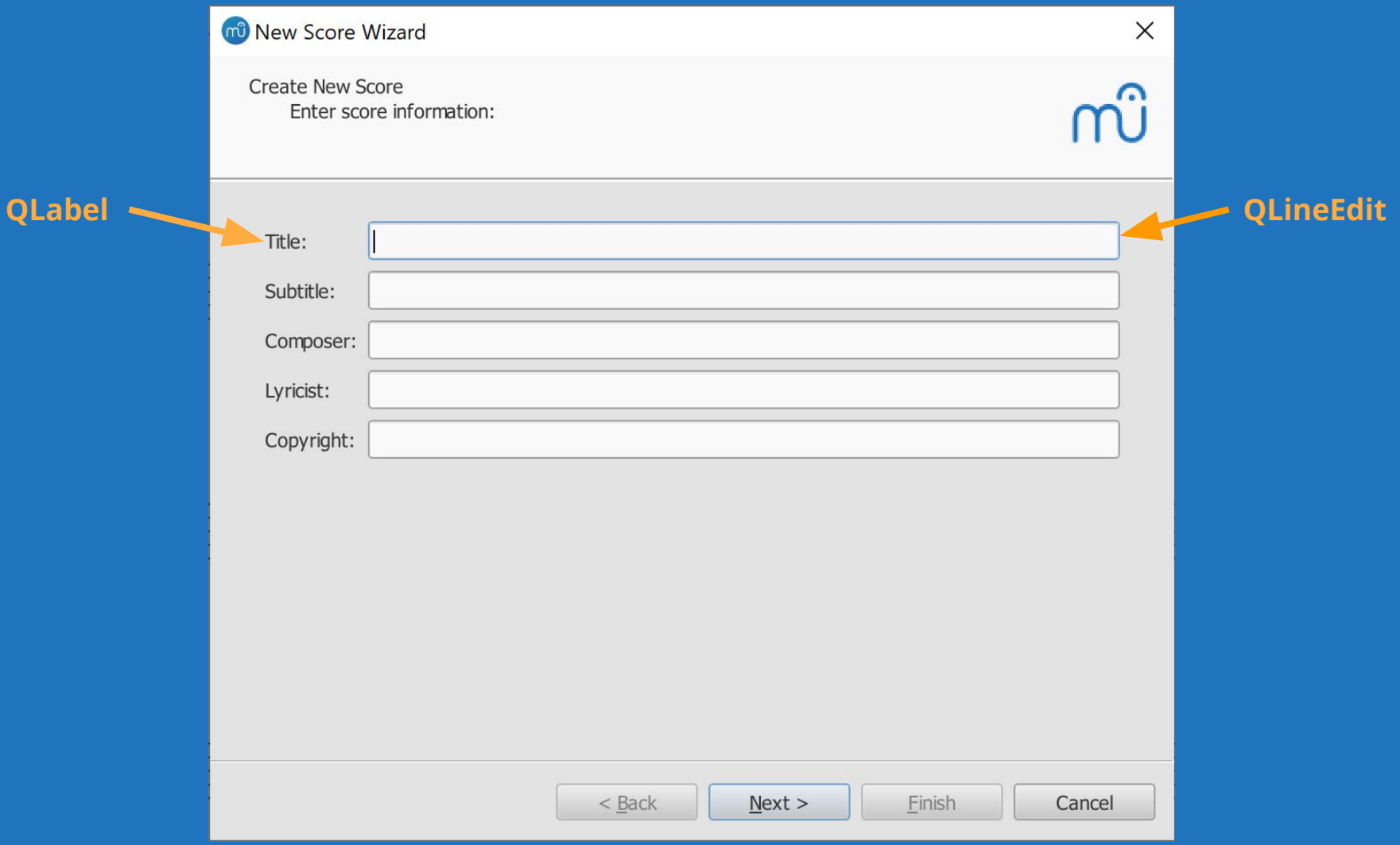
Current accessibility work: towards a “fully accessible” MuseScore

- Added UI size and color customization for low-vision users
- Enhanced score navigation and screen reader feedback
- Automatic placement reduces need for visual adjustments to score
- Reimplemented palettes using QML
- Scripts to support screen readers other than NVDA

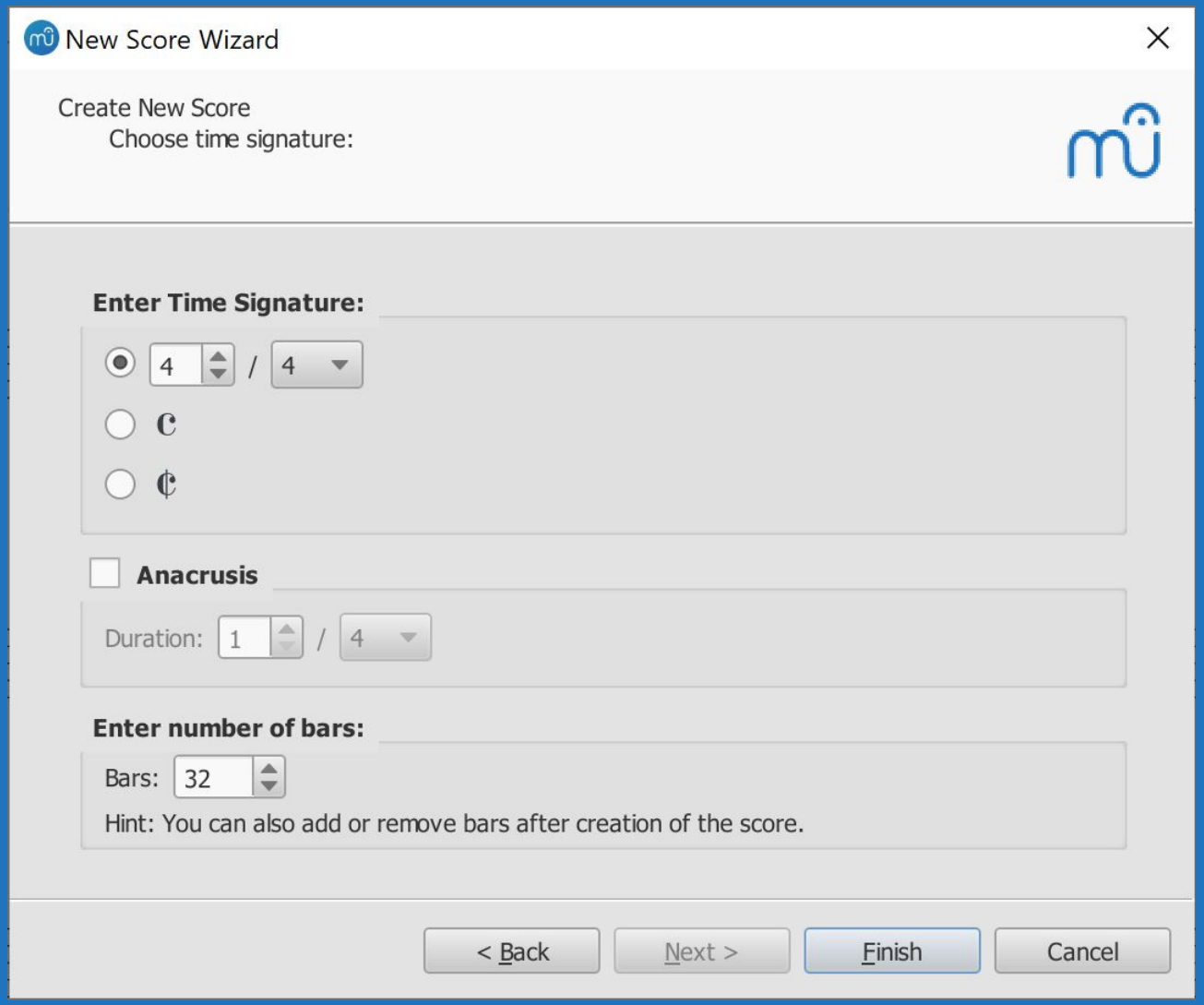
MuseScore 3: blind and low vision users can create and edit scores



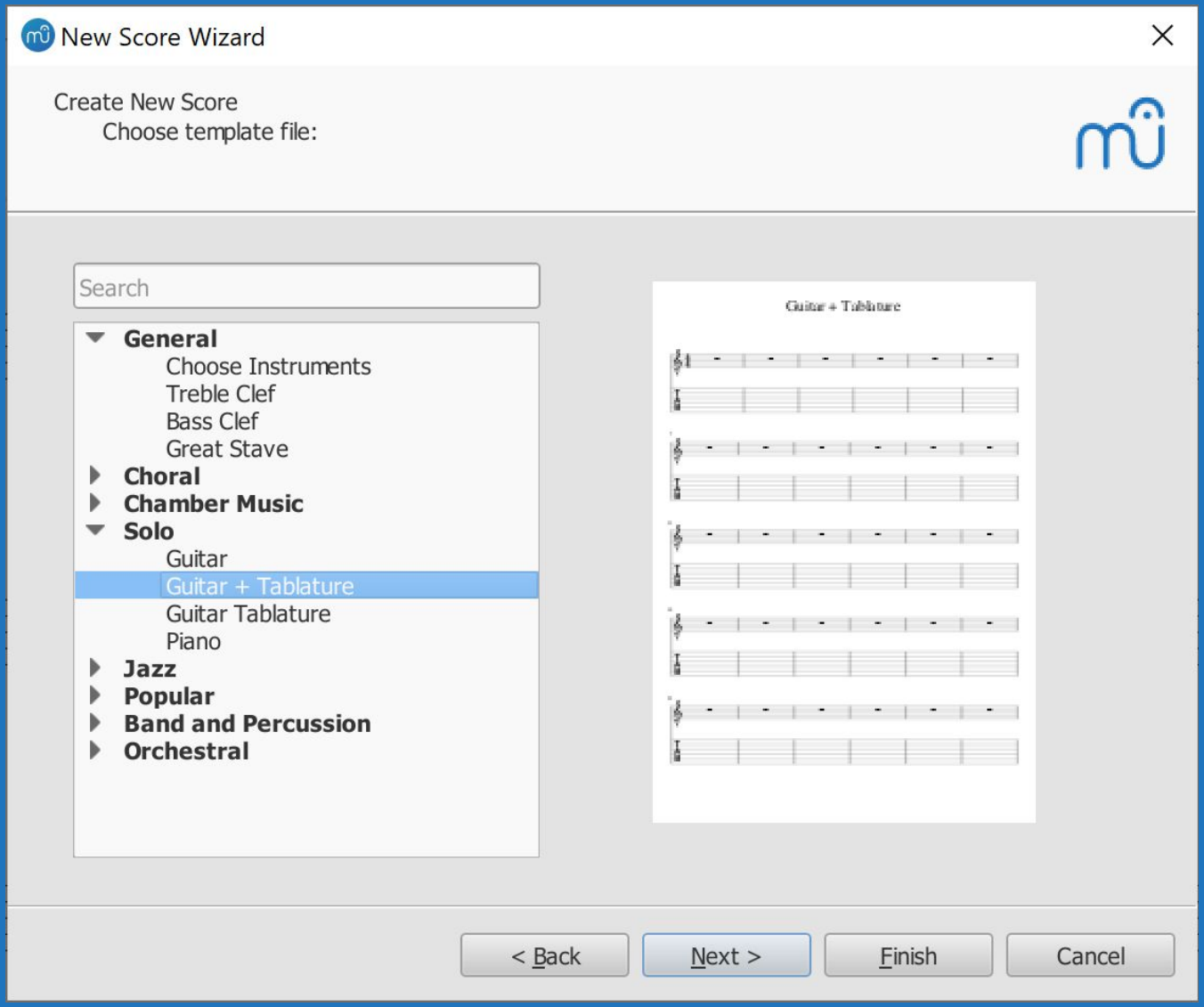
New Score Wizard



- Standard Qt widgets are accessible for free!
- Problem: labels are not included in tab-order
- Solution: QLineEdit given accessible **name** and **description**
  - also made **“buddies”** with QLabels



- Same for other Qt Widgets like QComboBox, QRadioButton, etc.
- Screen readers can deal with these as long as they have a name & desc.

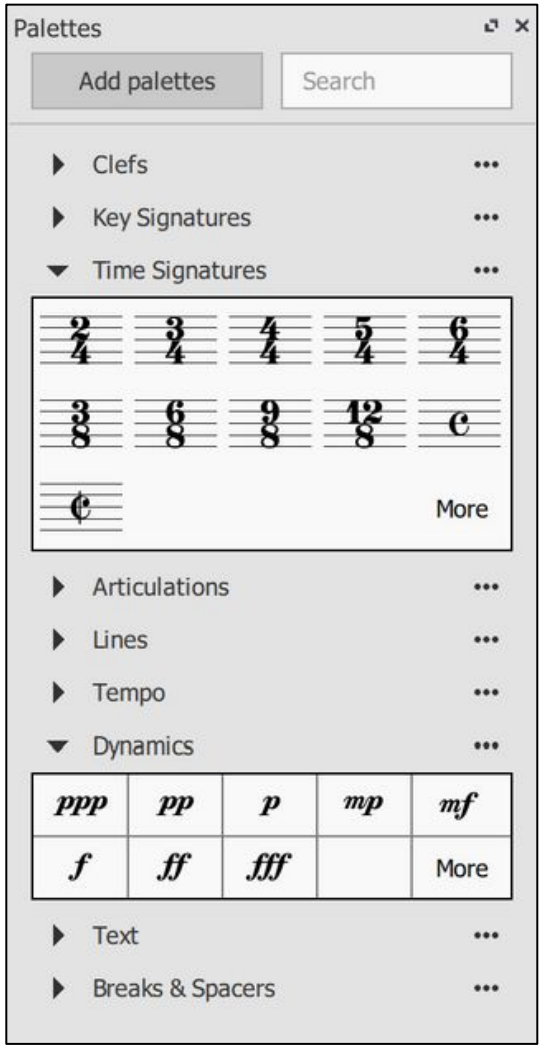


- Item view widgets (list view, grid view, tree view) contain many items
- Each item requires a name and description
- Items are not widgets! (Not tabbable, etc.)

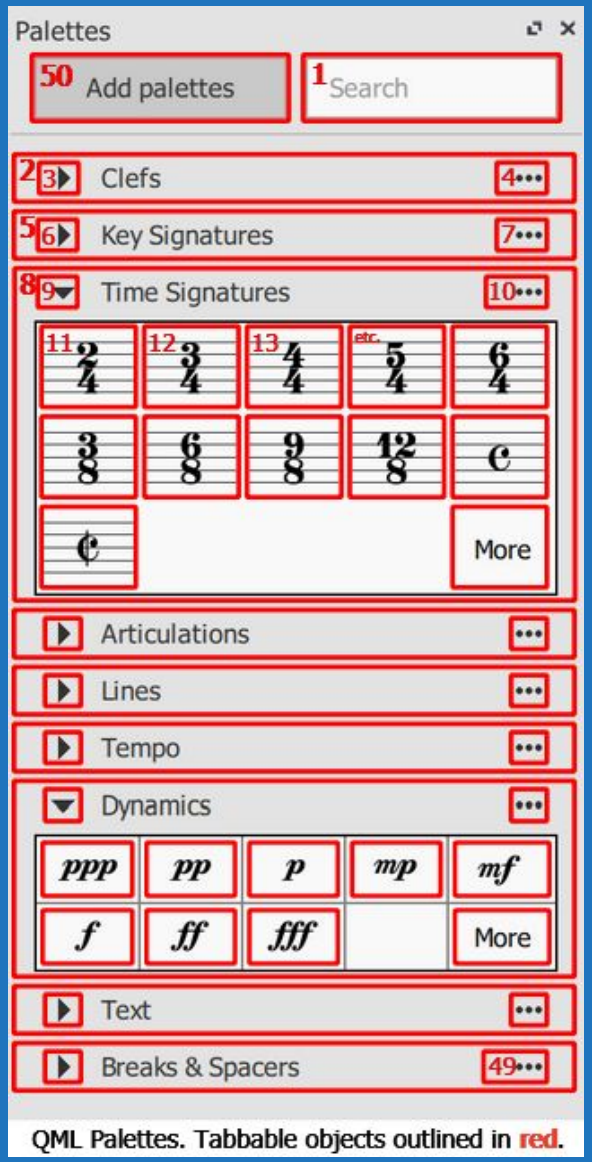
Palettes

Palettes

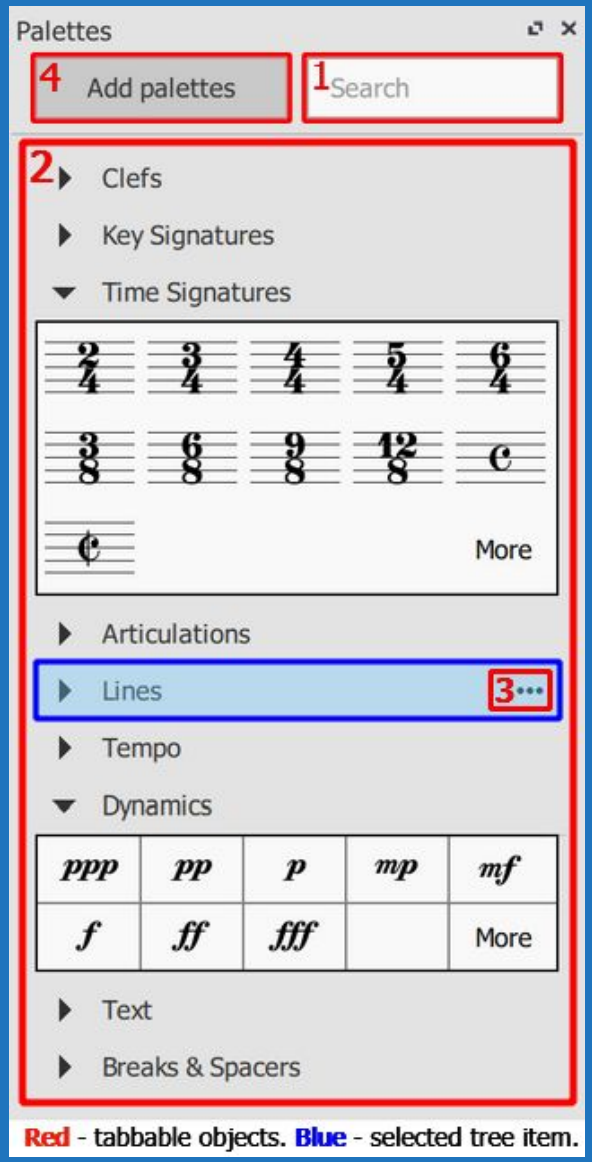
- We want a “tree of grids”
  - No standard widget behaves as desired
- Options:
  - **Combine multiple widgets** (e.g. list views inside tree view)
  - **Create new item view** (reimplement QAbstractItemView)
  - **Switch to QML**
- QML is more flexible than Qt Widgets
  - Everything is an item!
  - Able to:
    - Set accessible name and description
    - Set tab behaviour
    - Set arrow key behaviour



Navigation with Tab only



Arrow keys and tab

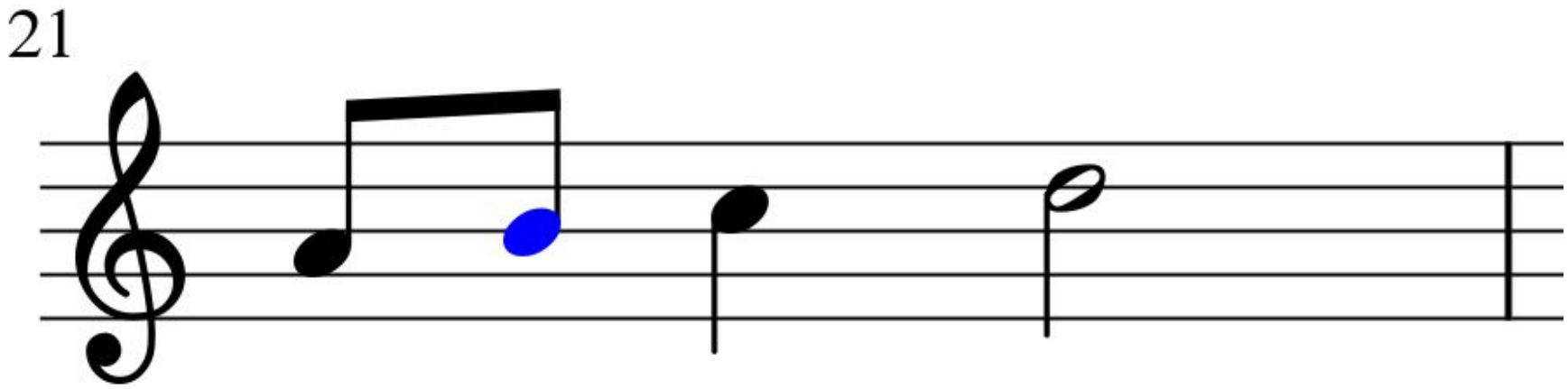


- Use tab for high-level control & arrow keys for low-level control
- Adds structure (not just a flat UI) -> saves time for keyboard users

Score View

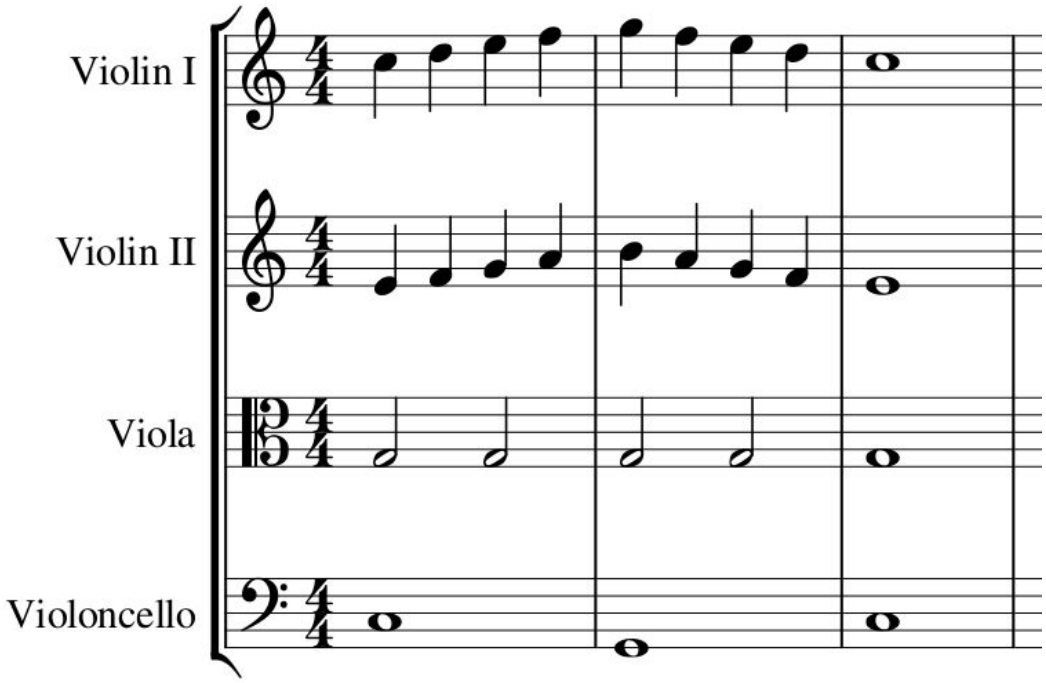
Score View - Accessibility Requirements

- Keyboard navigation
- Screen reader feedback
- With the blue "B" selected, pressing right arrow should:
  - Move selection to "C"
  - Read "Note C5 quarter beat 2"



Score View - Navigation Similar To Text

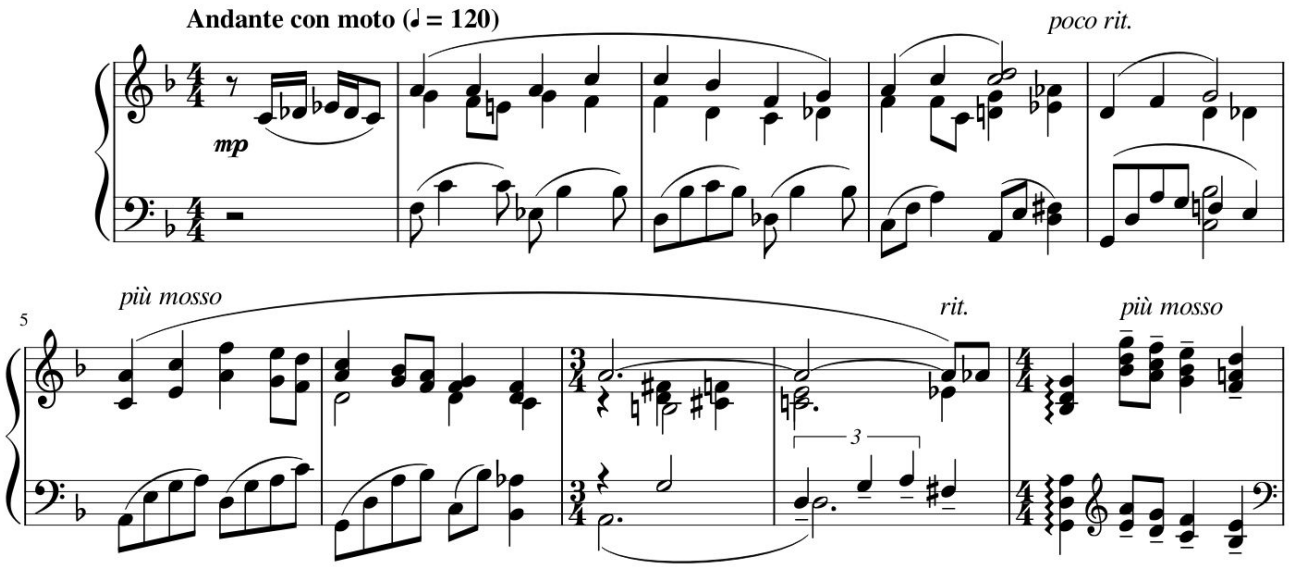
- Characters ⇒ notes (Left/Right)
- Words ⇒ measures (Ctrl+Left/Right)
- Lines ⇒ staves (Alt+Shift+Up/Down, because Up/Down change pitch)



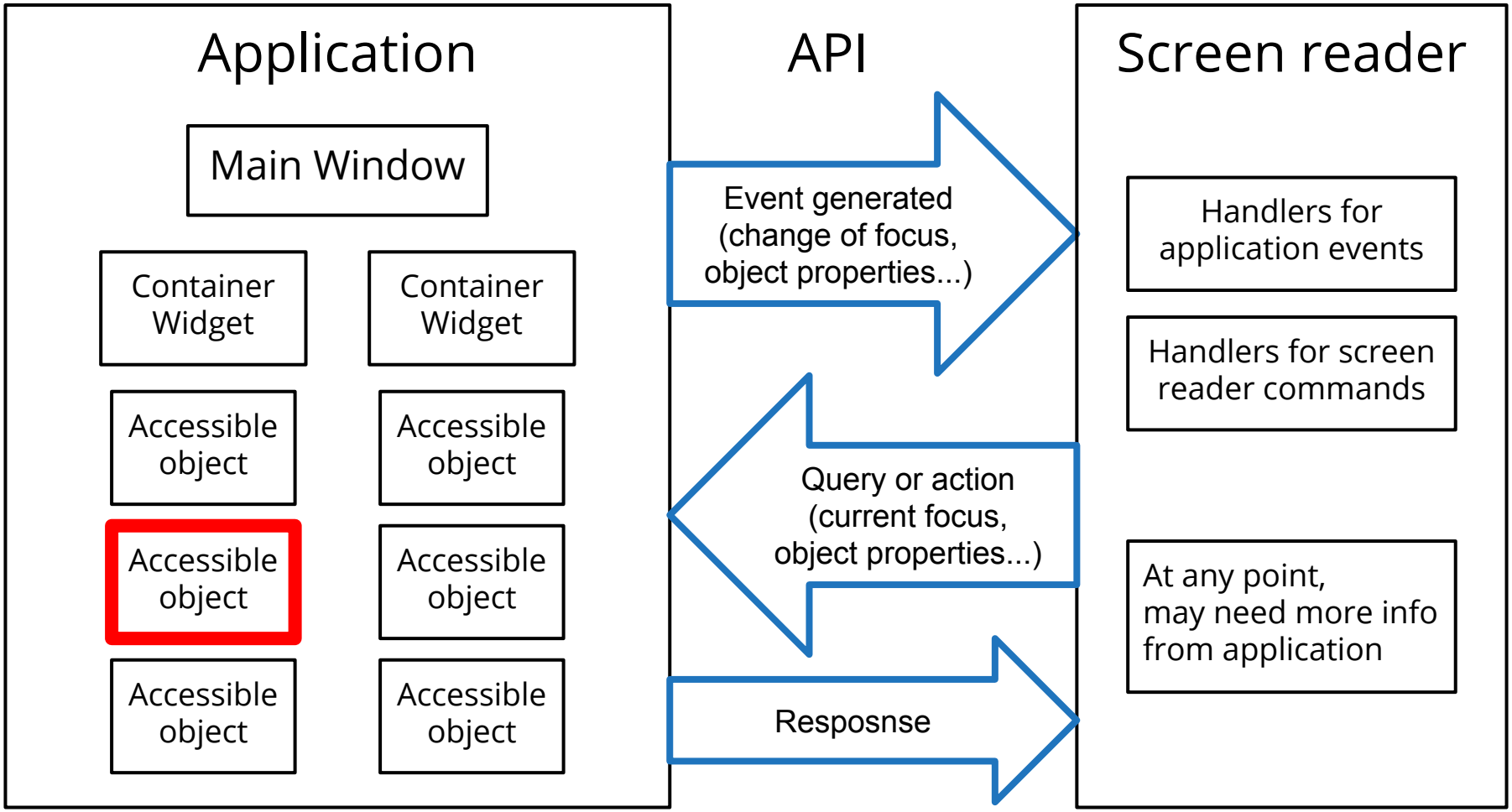
Score View - Navigation Different From Text

- Staves versus systems - which is the next "line"?
- Chords and multiple voices - which is the next "character"?
- Where do other markings fit in?

MuseScore navigation commands linearize music (Alt+Left/Right)



Score View - Application / Screen Reader Interface



Score View - Screen Reader Support Using Qt

AccessibleScoreView

Implements QAccessibleInterface

- role(): StaticText
- text():
  - Name: name of score
  - Value: score info
  - Description: same

ScoreAccessibility

Constructs score info after every command

- Description of selection
  - Name of element ("note C5 quarter")
  - Time position (“beat 2, bar 9”)
  - Staff
- Optimizations
  - Most important information first
  - Repeated information omitted

QAccessibleValueChangeEvent

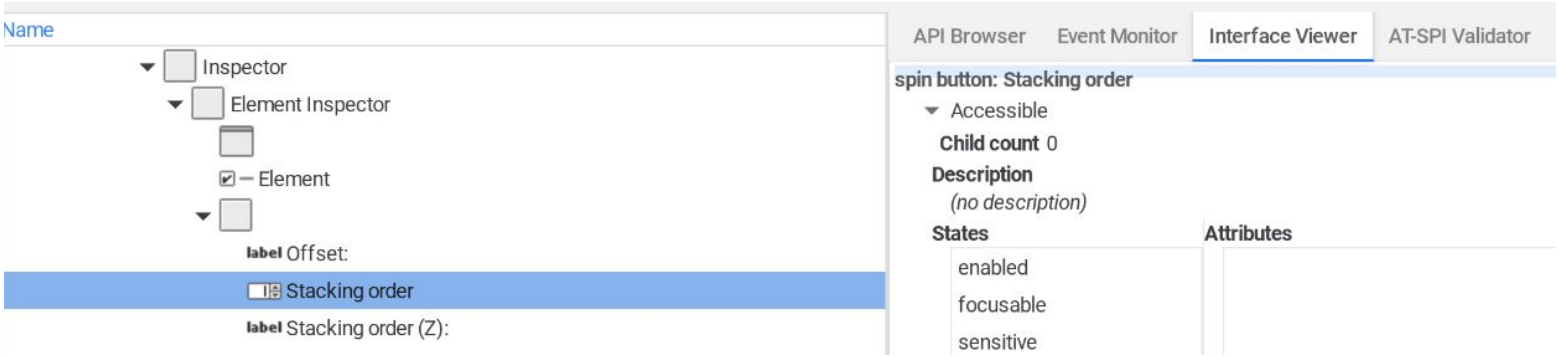
Score View - Platform Dependencies

- Screen readers do not necessarily read changed value by default
- Different underlying API frameworks have different semantics
- NVDA works out of the box with our implementation
- Scripts for Orca and JAWS can detect the event and read the information
- Narrator (Windows), VoiceOver (macOS)?

NVDA, Orca, and JAWS can be fully supported

Implementing Accessibility - Insights

- Implementing accessibility need not be difficult
- It can hard be find relevant information, though
- Toolkits like Qt or GTK can help abstract some details
- Much remains incomplete or dependent on platform
- Analysis tools like Accerciser and Accessibility Insights are useful
- Screen reader scripting may be required
- Be prepared for trial and error
- Open source for Qt, NVDA, Orca, and pyatspi is a wonderful thing!



Implementing Accessibility - Tools and Standards

- Programming Toolkits
  - Qt
  - GTK
- API Frameworks
  - IAccessible2
  - Linux: ATK, AT-SPI
  - Windows: MSAA, UIA
  - macOS: NSAccessibility
- Runtime Analysis Tools
  - Linux: Accerciser
  - Windows: Accessibility Insights (open source)
- Accessibility Standards
  - ISO 9241-171
  - Web: WCAG 2, WAI-ARIA

Case study - Music Education

Blind music students are being "mainstreamed" into schools that are not prepared for the challenges. MuseScore can help these students read and write music as required to succeed.

- Teachers can create educational materials accessible to all
- Students can read and complete assignments, and teachers can read and grade the results
- Students can learn music for lessons and ensembles
- Students can learn more about music notation in general