W3C Music Notation Community Group Meeting

30 April 2020

W3C Music Notation Community Group

- Founded in July 2015
- Develops and maintains format and language specifications for notated music used by web, mobile, and desktop applications
- Maintains and updates the MusicXML and Standard Music Font Layout (SMuFL) specifications
- Developing new MNX spec to handle new use cases and technologies
- Community group membership is free of charge and does not require W3C membership
- https://www.w3.org/community/music-notation/

Agenda

- MusicXML 3.2 kickoff
- SMuFL update
- MNX status and progress

Why MusicXML 3.2?

- We can improve how MusicXML works for its core use cases of music notation exchange and archiving
- In particular, MusicXML needs better support for parts
- Address many issues and suggestions from the past few years
- Working on MNX does not make MusicXML obsolete
- However, issues that cannot be resolved effectively given MusicXML's design are best postponed to MNX-Common

Current MusicXML 3.2 Plans

- About 40 open issues in the V3.2 milestone, grouped in 6 themes
  - Better support for parts
  - Improve support for machine listening applications
  - Better XML tool support
  - Fix gaps in preserving score appearance
  - Fix gaps in preserving score playback
  - Clarify documentation of several features
- When to release
  - First quarter of 2021

Improved Part Support

Parts

- Many applications allow editing of score and parts in a single file
- MusicXML files encode either a score or a part, with no standard way to represent relationships
- Issue 278 has the main proposal: optionally include a full score and full copy of all parts within the zip archive of an .mxl file
- Also provide better support for creating parts when just importing a score as a .musicxml text file
  - Specify transpositions for parts within concert scores (Issue 279)
  - Specify top staff and bottom staff directions (Issue 37)
- 8 issues so far

Machine Listening

Listening

- Machine listening includes notation applications like interactive performance and music practice that rely on listening to a performer
- These applications often need supplemental data beyond what is required for music display or playback
- Issue 294 has the main proposal
  - A new <listen> element, parallel to the current <sound> and <play> elements already used for playback
  - A new <player> element to identify performers in divisi parts
  - Current examples including waiting points for performance (fermatas, section breaks) and identifying performers for practice assessment

Improved Tool Support

Tools

- Help application developers by enhancing support for contemporary XML tools
- 5 issues so far:
  - Add an XML catalog for local copy of schema or DTD (Issue 259)
  - Select a standard namespace to use when combining MusicXML with other XML vocabularies (Issue 266)
  - Document solutions to common code generation issues (Issue 280)
  - Add XSLT stylesheets for automatically generating ID attributes (Issue 260)
  - Update MusicXML schema and DTD locations to use https (Issue 285)

Gaps in Appearance

Appearance

- Fill in the gaps where score appearance cannot be shared completely between applications using standard elements
- 8 issues so far:
  - Piano pedal lines that do not explicitly end with a pedal up (Issue 302)
  - Differentiate styles of guitar bends, polychords, enclosures, and multiple-rest measure numbers
  - Percussion staves with widely spaced lines (Issue 305)
  - Vertical alignment and positioning for lyrics and measure numbers
- Is it now time to tackle other alignment issues?
  - Specify SMuFL alignment for rest display-step (Issue 5)
  - Specify exact default-x origin for barlines (Issue 6)

Gaps in Playback

Playback

- Fill in the gaps where score playback cannot be shared completely between applications using standard elements
- 4 issues so far:
  - Add support for swing that matches how applications work (Issue 283)
  - Add virtual instrument changes for doublings (Issue 293)
  - Clarify how the number attribute works for the ending element (Issue 291)
  - Add new tin whistle instrument to sounds.xml file (Issue 289)

Documentation

Documentation

- Keep on fixing places where the MusicXML documentation is incomplete, inaccurate, or confusing
- 12 issues so far, not counting those that are also in other categories
- Roman numeral analysis (Issue 295) is included here but looks like more than documentation. Any volunteers to complete a design?
- We will also take another look to see if we can move the MusicXML 3.0 documentation from the musicxml.com site to the W3C site, updated for MusicXML 3.2

Postponed to MNX-Common

- Some common suggestions currently postponed to MNX-Common
  - Improve semantics for text
  - Improve semantics for brackets and lines
  - Anything that would break compatibility with older MusicXML versions

MusicXML 3.2 Next Steps

- Does this sound like a reasonable scope for MusicXML 3.2?
- Any additional areas to work on?
- Any areas proposed for work that we perhaps should leave out?
- Is 3.2 a good version number, or might 4.0 be better?
- The current milestone list is a work in progress
  - If you have favorite issues not yet written up, or written up but not included in the 3.2 milestone, please add them or discuss in GitHub
- If this sounds like a good starting point, we can begin right away

Community Group Membership

- To contribute a pull request to any of the W3C Music Notation Community Group projects, you need to be a member of the group
- Membership is free of charge
- Sign up at the Community Group home page
  - Click on the JOIN OR LEAVE THIS GROUP button to get started
  - If you work for an organization in the area of music notation, please join as a representative of that organization
  - The W3C signup process is not always the easiest to use – reach out if you have questions or issues

SMuFL 1.4

- Aim to complete SMuFL 1.4 before the end of 2020
- Around 30 issues in total currently outstanding: 19 currently in scope
  - Define complementary text font family in SMuFL font metadata
  - Expand tuplets in “Beamed Groups of Notes” range
  - Add noteheads for chromatic solfège
  - Support for numbered notation (e.g. Chinese *jianpu*)
- New proposals are also always welcome – just open an issue

MNX by example

MNX-Common by example

Welcome to our overview of the new music notation format currently codenamed MNX-Common (see intro here). Our goal is to show you how MNX-Common documents look and work, by example rather than formal specification. This website is primarily intended for people making music-notation software.

On this page, we present many musical situations — starting from a simple "Hello world" example and getting progressively more complex — to show you exactly how each musical situation is encoded. For each example, you'll see an image of the desired notation, followed by the encoding in MusicXML and MNX-Common.

https://w3c.github.io/mnx/by-example/

MusicXML-to-MNX converter

README.md

mnxconverter

A Python package for converting between MusicXML and the new MNX-Common format.

Disclaimer

This is alpha software! The MNX-Common format is being actively designed — so anything in this code might change, including the very name MNX-Common itself.

This converter is also very limited in scope at the moment. So far, it only reliably converts the types of notations described in MNX-Common by example.

Intro to mnxconverter

- Free, open-source Python library
- Command-line tool to convert MusicXML to MNX
- So far: Perfectly converts every example in “MNX by example”
- There’s LOTS it doesn’t do yet, but the foundation is in

mnxconverter goals

- Make MNX development less conceptual and more practical
- Get a sense of how MNX "feels" in practice
- Get developers involved in the design process for MNX
- Eventually serve as a fully featured, production-ready converter between MNX and MusicXML
- Eventually serve as a fully featured, production-ready "cleaner" for MusicXML files

mnxconverter non-goals

- Becoming a general-purpose abstraction for music notation data in Python
  - Data structures are deliberately undocumented
- Any kind of consumer-facing tools (rendering, musicological research)

mnxconverter tech details

- Imports MusicXML into a Python data structure (Bar, Note, etc.)
- Converts that data structure into MNX
- Architecture allows for import from other formats
- Will include heuristics for MusicXML import
  - Example: “Does <octave-shift> affect the first note following the close tag?”
- Comprehensive test suite

mnxconverter personal reactions

- The first true test
- Feels good!
- Inspired me to tweak data structures in my own notation app ☺

mnxconverter and you

- See how your Favorite MusicXML Files look as MNX
- See an open-source approach to MusicXML parsing
- Get to know the MNX concepts (sequence, event, directions…)
- Selfish reason: It might inspire your own notation data structures

“Should I start using MNX?”

- Don’t implement it yet unless you want to be on the bleeding edge
- BUT, do begin to familiarize yourself with its concepts

The naming issue

- MNX-Common vs. MNX-Generic
- Proposal: MNX and MGX

MNX next steps

- For notational concepts that are already in the spec:
  - Add to “MNX by example”
  - Implement in mnxconverter
- For notational concepts that aren’t in the spec:
  - Include “MNX by example” example in GitHub issue for discussion
  - Update the spec
  - Implement in mnxconverter
- Eventually reach a “Start implementation” milestone this year
  - This means: “The format is still being developed, but it’s stable enough for notation apps to begin working with it.”