Maintainer's guide to `xml2ly`

Jacques Menu

December 23, 2019 version

Abstract

This document presents the design principles and architecture os `xml2ly`, as well as information needed to maintain it. It is part of the `libmusicxml2` documentation, to be found at `https://github.com/grame-cncm/libmusicxml/tree/lilypond/doc`. `xml2brl` is mentioned but not described in detail.

In the `libmusicxml2` library, the source code specific to `xml2ly` can be found at `https://github.com/grame-cncm/libmusicxml/tree/lilypond/src/lilypond` and `https://github.com/grame-cncm/libmusicxml/tree/lilypond/src/interface`.

All the examples mentioned can be downloaded from `https://github.com/grame-cncm/libmusicxml/tree/lilypond/files/samples/musicxml`. They are grouped by subject in subdirectories, such as `basic/HelloWorld.xml`.

1 Acknowledgements

Many thanks to Dominique Fober, the designer and maintainer of the `libmusicxml2` library!

2 Overview of `xml2ly`

2.1 Why `xml2ly`?

MusicXML (*Music eXtended Markup Language*) is a specification language meant to represent music scores by texts, readable both by humans and computers. It has been designed by the W3C Music Notation Community Group (`https://www.w3.org/community/music-notation/`) to help sharing music score files between applications, through export and import mechanisms.

The homepage to MusicXML is `https://www.musicxml.com`.

MusicXML data contains very detailed information about the music score, and it is quite verbose by nature. This makes creating such data by hand quite difficult, and this is done by applications actually.

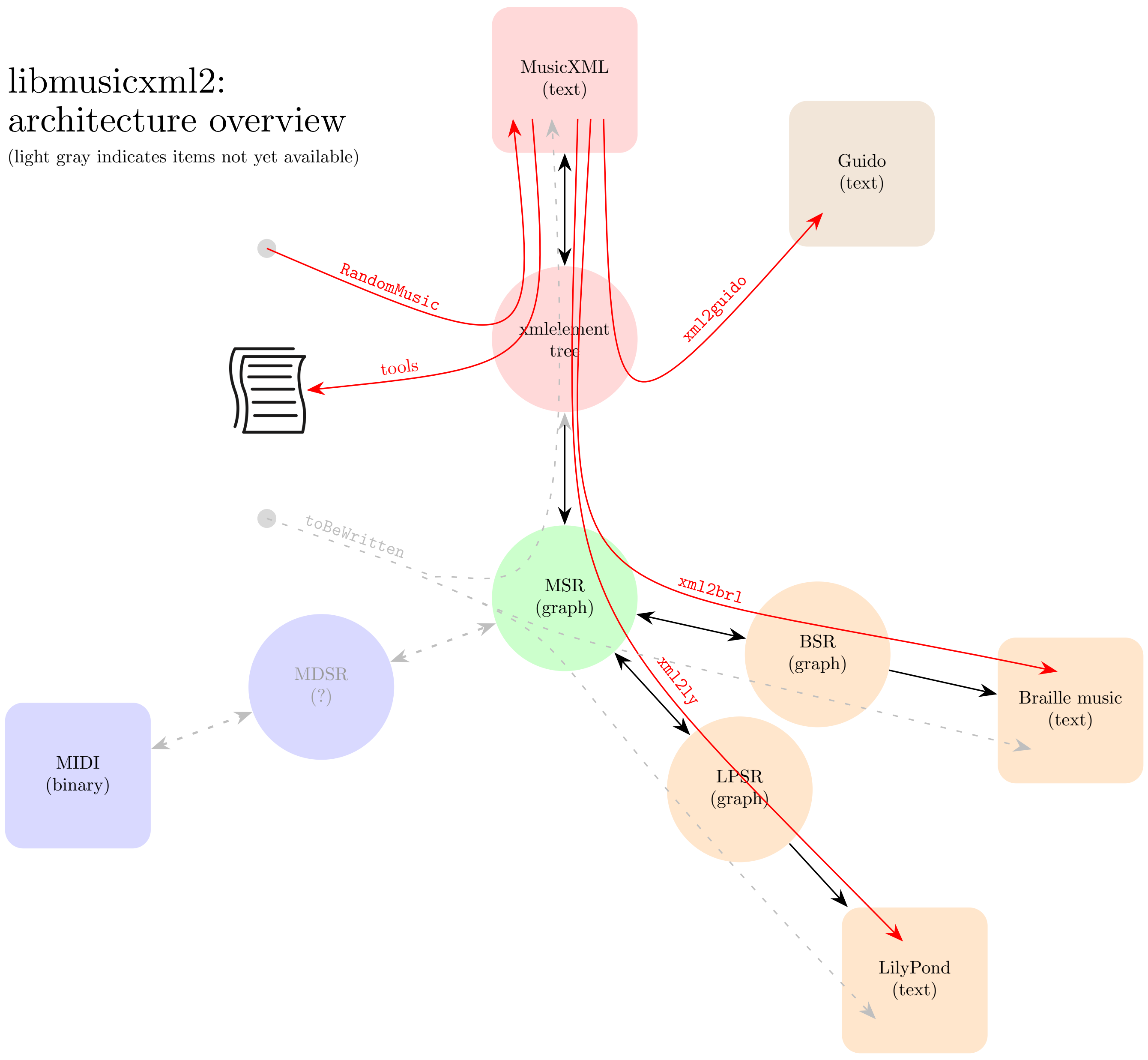
2.2 What `xml2ly` does

3 Prerequisites

In order to maintain `xml2ly`, one needs to do the following:

- obtain a working knowledge of C++ programming. The code base of `xml2ly` uses classes, simple and multiple inheritance, and templates;
- study MusicXML, starting maybe from `IntroductionToMusicXML.tex`. A deep knowledge of that matter comes with experience;
- study the architecture of `libmusicxml2`, which can be seen at `libmusicxmlArchitecture.pdf`, and is presented in figure 1. It shows the place of `xml2ly` in the whole.
-

Figure 1: `libmusicxml2` architecture



Entity	Description
xmlelement tree	a tree representing the MusicXML markups such as `<part-list>`, `<time>` and `<note>`
MSR	Music Score Representation, in terms of part groups, parts, staves, voices, notes, ...
LPSR	LilyPond Score Representation, i.e. MSR plus LilyPond-specific items such as `\score` blocks
BSR	Braille Score Representation, with pages, lines and 6-dots cells
MDSR	MIDI Score Representation, to be designed
`RandomMusic`	generates an xmlelement tree containing random music and writes it as MusicXML
tools	a set of other demo programs such as `countnotes`, `xmltranspose` and `partsummary`
`toBeWritten`	should generate an MSR containing some music and write it as MusicXML, LilyPond and Braille music
`xml2ly`	performs the 4 hops from MusicXML to LilyPond to translate the former into the latter
`xml2brl`	performs the 4 hops from MusicXML to Braille music to translate the former into the latter (draft)

- Note: `xml2ly` has a '`-jianpu`' option
- Note: `midi2ly` translates MIDI files to LilyPond code
- Note: `lilypond` can generate MIDI files from its input

`xml2guido` v2.3, `xml2ly` v0.9, `xml2brl` v0.01, August 2019

4 Programming style and conventions

4.1 Source code presentation

The following text-editing conventions are used:

- tabs are not used before the first non-space character in a line, two spaces are used instead;
- the code is not tightly packed: declarations in classes have the members' names aligned vertically, with many spaces before them if needed, and empty lines are used to separate successive activities in methods.

4.2 File names

The name '`lilypond`' was chosen by Dominique long before work started on
xmlTobrl.

There's a single '`lilypond`' folder to contain MSR, LPSR, BSR, `xml2ly` and `xml2brl`, even though BSR and braille music are a distinct branch. This has been preferred by Dominique as the manager of `libmusicxml2`.

Most file names start with an identification of the context they belong to, such as '`oah`', '`mxmlTree`', '`msr`', '`lpsr`', '`lilypond`', '`bsr`', '`braille`', '`xml2ly`' and '`xml2brl`'.

The '`*Oah.*`' files handle the options and help for the corresponding context, such as '`xml2lyOah.h/.cpp`'.

The '`traceOah.h/.cpp`', '`musicXMLOah.h/.cpp`', '`extra`' and '`general`' context are about the corresponding help groups.

There are a couple of 'globlal' files not related to any particular context: '`utilities.h/.cpp`', '`messagesHandling.h/.cpp`' and '`version.h/.cpp`'.

4.3 Defensive programming

The code base of `xml2ly` is *defensive programming* oriented, which means that:

- identifiers are explicit and long if needed – only very local ones are short, such as iteration loops indexes;
- the code is organized in sections, with an initial comment documenting what the code does;
- '`msrAssert()`' is used to do sanity checks, such as detect a null pointer prior to using it;

The MusicXML data is not systematically checked for correctness. Checks are done, however, to ensure it won't crash due to missing values.

5 The two-shot visitors pattern

5.1 Basic mechanism

5.2 An example

6 Passes organization

7 Translating MusicXML data to an mxmlTree

8 Translating an mxmlTree to an MSR

9 Translating an an MSR to a LPSR

10 Translating an an LPSR to a LilyPond code

Listings

List of Figures

Contents