libmusicxml2 architecture overview

xml2guido v2.3, xml2ly v0.9, xml2brl v0.01

March 19, 2021

Jacques Menu

Contents

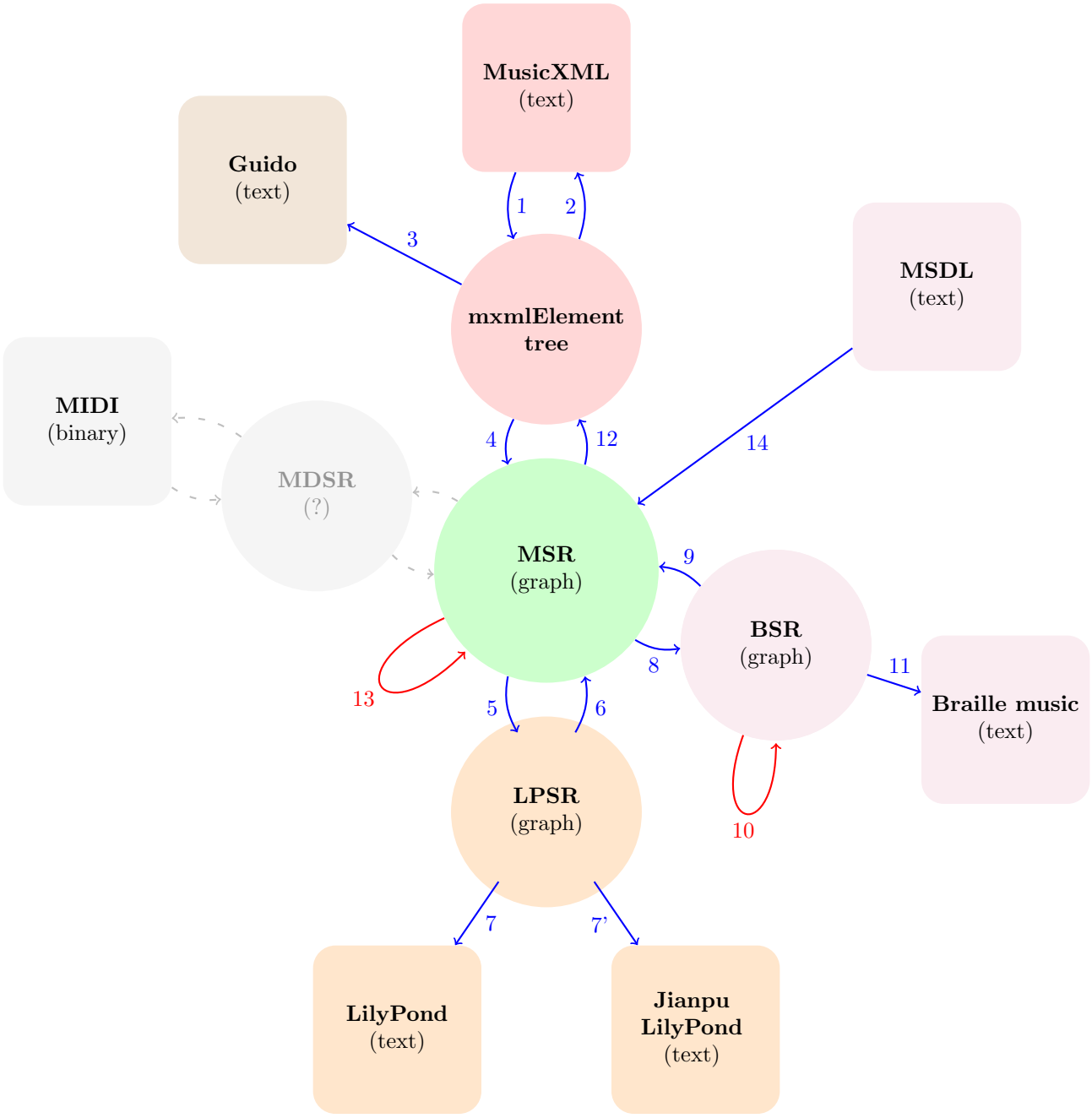
This document shows the architecture of the libmusicxml2 library, to be found at https://github.com/grame-cncm/libmusicxml/tree/lilypond.

libmusicxml2 is written in C++11 and provides a set of music scores representations and translators between various textual music scores formats. Building it only requires a C++11 compiler and cmake.

1 Architecture

The picture at figure 1, page 2, shows how libmusicxml2 is structured. The dimmed, dashed arrows indicate items not yet available. The numbered arrows show the existing conversions between formats and representations.

Figure 1: `libmusicxml2` architecture



2 Formats and representations

The formats supported by `libmusicxml2` are:

Format	Description
MusicXML	a text containg markups such as `<part-list>`, `<time>` and `<note>`;
Guido	a text containg markups such as `\barFormat`, `\tempo` and `\crescEnd`;
LilyPond	a text containg commands such as `\header`, `\override` and `\transpose`;
Jianpu LilyPond	a text containg LilyPond commands and the use of `lilypond-Jianpu` (https://github.com/nybbs2003/lilypond-Jianpu/jianpu10a.ly) to obtain a Jianpu (numbered) score instead of the default western notation. `lilypond-Jianpu` should be accessible to LilyPond for it to produce the score;
Braille music	a text containg 6-dot cells, as described in http://www.brailleauthority.org/music/Music_Braille_Code_2015.pdf;
MSDL	a text describing a score in the MSDL language.

The representations used by `libmusicxml2` are:

Representation	Description
MSR	Music Score Representation, in terms of part groups, parts, staves, voices, notes, etc. This is the heart of the multi-language translators provided by `libmusicxml2`;
mxmlelement tree	a tree representing the MusicXML markups such as `<part-list>`, `<time>` and `<note>`;
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.

3 Basic tools

`libmusicxml2` supplies a number of basic tools using its features:

- `xmlread` converts MusicXML data and displays the corresponding xmlElement tree;
- `countnotes` reads MusicXML data and displays the number of notes it contains;
- other programs such as `xmltranspose` and `partsummary` demonstrate the possibilities of the library, in particular those of the two-phase visitors pattern it uses.
- `xml2midi` reads MusicXML data and outputs a midi version of it.

It is to be noted that:

- LilyPond provides `midi2ly` to translate MIDI files to LilyPond code;
- LilyPond can generate MIDI files from its input.

4 Generators

A generator is an executable that creates data representing a score without reading any input file. For example:

- `RandomMusic` generates an mxmlelement tree containing random music, and writes it as MusicXML to standard output;
- `RandomChords` generates an mxmlelement tree containing random two-note chords, and writes it as MusicXML to standard output;
- `MusicAndHarmonies.cpp` generates an mxmlelement tree containing notes and harmonies, and writes it as MusicXML to standard output;

5 Conversion passes

The numbers in the picture refer to so-called passes (compiler writing terminology), i.e. atomic components of the library that convert a representation into another. The passes are numbered in the order they were added to the library:

Passes	Description
1	reads MusicXML data from a file or from standard input is '-' is supplied as the file name, and creates an mxmlelement tree containg the same data;
2	converts an mxmlelement tree into MusicXML data. This is a mere 'print()' operation;
3	converts an mxmlelement tree into Guido text code, and writes it to standard output;
4	converts an mxmlelement tree into and MSR representation. MusicXML represents how a score is to be drawn, while MSR represents the musical contents with great detail. This pass actually consists in two sub-passes: the first one builds an MSR skeleton containing empty voices and stanzas, and the second one the fills this with all the rest;
5	converts an MSR representation into an LPSR representation, which contains an MSR component build from the original MSR (pass 5). The BSR contains LilyPond-specific representations such as `\layout`, `\paper`, and `\score` blocks;
6	converts an LPSR representation into an MSR representation. There is nothing to do, since the former contains the latter as a component;
7	converts an LPSR representation into LilyPond text code, and writes it to standard output;
7'	converts an LPSR representation into LilyPond text code using `lilypond-Jianpu`, and writes it to standard output. This pass is run with `xml2ly -jianpu`;

8	converts an MSR representation into a BSR representation, which contains an MSR component build from the original MSR (pass 5). The BSR contains Braille music-specific representations such as pages, lines and 6-dot cells. The lines and pages are virtual, i.e. not limited in length;
9	converts a BSR representation into an MSR representation. There is nothing to do, since the former contains the latter as a component;
10	converts a BSR representation into another one, to adapt the number of cells per line and lines per page from virtual to physical. Currently, the result is a mere clone;
11	converts a BSR representation into Braille music text, and writes it to standard output;
12	converts an MSR representation into an mxmlelement tree;
13	converts an MSR representation into another one, built from scratch. This allows the new representation to be different than the original one, for example to change the score after is has been scanned and exported as MusicXML data, or to add skip (invisible) notes to avoid the LilyPond issue #34. For simplicity and efficiency reasons, this pass is not present as such, but 'merges' within passes 6 and 9;
14	converts an MSDL score description into an MSR representation.

6 Translators

A translator is a sequence of two or more passes, each converting one representation into another, in a pipeline way. The first one provided by the library was `xml2guido`.

The other translators provided by `libmusicxml2` were added later and are in the form of functions. Executable command-line applications using them are also supplied. They are shown in the table below:

Output format	Input format	
	MusicXML	MSDL
MusicXML	`xml2xml`	`msdl -musicxml`
LilyPond	`xml2ly`	`msdl -lilypond`
Jianpu LilyPond	`xml2ly-jianpu`	`msdl -lilypond -jianpu`
MusicXML	`xml2xml`	`msdl -musicxml`
Braille music	`xml2brl`	`msdl -braille`

The executables available in `libmusicxml2` are:

Translator	Description
`xml2guido`	converts MusicXML data to Guido code, using passes: 1 ⇒ 3

`xml2ly` performs the 4 hops from MusicXML to LilyPond to translate the former into the latter, using these passes:
1 ⇒ 4 ⇒ 5 ⇒ 7
The `-jianpu` option is supplied to create Jianpu (numbered) scores, in which the notes are represented by numbers instead of graphics, using passes:
1 ⇒ 4 ⇒ 5 ⇒ 7'

`xml2brl` performs the 5 hops from MusicXML to Braille music to translate the former into the latter (draft);
1 ⇒ 4 ⇒ 8 ⇒ 10 ⇒ 11

`xml2xml` converts MusicXML data to MSR and back. This is useful to modify the data to suit the user's needs, such as fixing score scanning software limitations or to enhance the data:
1 ⇒ 4 ⇒ 13 ⇒ 12 ⇒ 2

`xml2gmn` converts MusicXML data to Guido code, using passes:
1 ⇒ 4 ⇒ 13 ⇒ 12 ⇒ 3

In order to demonstrate the use of the MSR API, `Mikrokosmos3Wandering` creates an MSR graph representing Bartok's Mikrokosmos III Wandering score, and then produces Guido, LilyPond, braille or MusicXML to standard output, depending on the '`-generated-code-kind`' option.

MSDL (Music Score Description Language) is a language under evolution being created by this author. It is meant for use by musicians, i.e. non-programmers, to obtain scores from a rather high-level description.
`libmusicxml2`supplies `msdl`, a compiler translating MSDL into Guido, LilyPond, braille or MusicXML to standard output, depending on the '`-generated-code-kind`' option.

7 Options and help

Having many executables with many options makes options and help handling a challenge.
This is why `libmusicxml2` uses it own OAH (Options And Help) object oriented infrastructure.

This library provides OAH (Options And Help), a full-fledged object-oriented options and help management infrastructure.

OAH organizes the options and the corresponding help in a hierarchy of groups, sub-groups and so-called atoms. OAH is introspective, thus help can be obtained for every group, sub-group or atom at will.

Each pass supplies a OAH group, containing its own options and help. The executable translators then aggregate the OAH groups of the passes they are composed of to offer their options and help to the user.

8 Multiple languages support

9 The MSR classes inheritance and use

The picture at figure 2, page 9, show the hierarchy of the main MSR classes. The arrows are colored to indicate there meaning:

- red: a link from a class to its base class;
- blue: a link from a class to another that uses smart pointers to instances or instances the former.

When not shown for clarity, the common base class of all these classes is `msrElement`.

The `otherMeasureElements` are:

- part names:
  - `msrPartNameDisplay`
  - `msrPartAbbreviationDisplay`
- bars:
  - `msrBarCheck`
  - `msrBarNumberCheck`
  - `msrBarline`
  - `msrHiddenMeasureAndBarline`
- breaks:
  - `msrLineBreak`
  - `msrPageBreak`
- notes:
  - `msrDoubleTremolo`
  - `msrVoiceStaffChange`
  - `msrOctaveShift`
- clefs, keys, times, tempo:
  - `msrClef`
  - `msrKey`
  - `msrTime`
  - `msrTempo`
- instruments:
  - `msrStaffDetails`

  - msrScordatura
  - msrAccordionRegistration
  - msrHarpPedalsTuning
  - msrPedal
  - msrDamp
  - msrDampAll
- lyrics:
  - msrSyllable
- rehearsals, segno and coda:
  - msrRehearsal
  - msrSegno
  - msrDalSegno
  - msrCoda
- others:
  - msrPrintLayout
  - msrEyeGlasses
  - msrStaffLevelElement
  - msrTranspose
  - msrTupletElement

Figure 2: MSRclasses hierarchy