

User's guide to xml2ly

Jacques Menu

December 23, 2019 version

Abstract

This document presents the design principles behind `xml2ly`, as well as the way to use it. It is part of the `libmusicxml2` documentation, to be found at <https://github.com/grame-cncm/libmusicxml/tree/lilypond/doc>.

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 sub-directories, such as `basic/HelloWorld.xml`.

1 Acknowledgements

Many thanks to Dominique Fober, the designer and maintainer of the `libmusicxml2` library. This author would not have attempted to work on a MusicXML to LilyPond translator without it already available.

In particular, the conversion of MusicXML data to a tree is extremely well done directly from the MusicXML DTD, and that was a necessary step to produce LilyPond code. Dominique also provided a nice way to browse this tree with a two-shot visitor design pattern, which this author used extensively in his own code. The interested reader can find information about that in [libmusicxml2.pdf](#).

`xml2ly` and some of the specific examples presented in this document are this author's contribution to `libmusicxml2`.

2 Overview of xml2ly

2.1 Why xml2ly?

LilyPond comes with `musicxml2ly`, a translator of MusicXML files to LilyPond syntax, which has some limitations. Also, being written in Python, it is not in the main stream of the LilyPond development and maintenance group. The latter has much to do with C++ and Scheme code already.

After looking at the `musicxml2ly` source code, and not being a Python developer, this author decided to go for a new translator written in C++.

The design goals for `xml2ly` were:

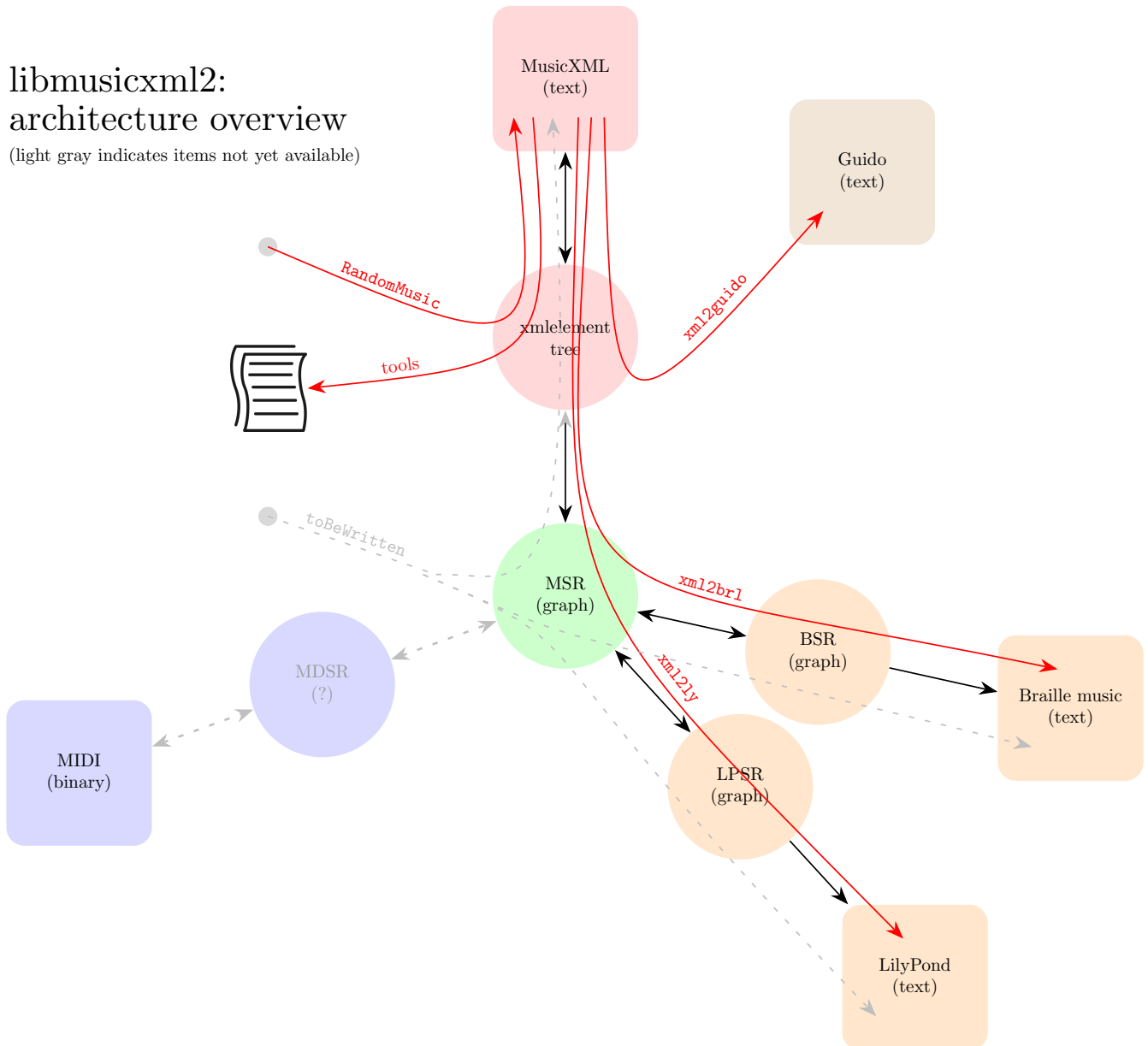
- to perform at least as well as `musicxml2ly`;
- to provide as many options as needed to adapt the LilyPond code generated to the user's needs.

Speed was not an explicit goal, but as it turns out, `xml2ly` is not bad in this respect.

Figure 1: libmusicxml2 architecture

libmusicxml2: architecture overview

(light gray indicates items not yet available)



Entity	Description
xmlelement tree	a tree representing the MusicXML markups such as <code><part-list></code> , <code><time></code> and <code><note></code>
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 <code>\score</code> 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 <code>countnotes</code> , <code>xmltranspose</code> and <code>partsummary</code>
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

2.2 What xml2ly does

The architecture of libmusicxml2, which can also be seen at [libmusicxmlArchitecture.pdf](#), is presented in figure 1. It shows the place of xml2ly in the whole.

The '-about' option to xml2ly details that somewhat:

```
1 menu@macbookprojm > xml2ly -about
2
3 What xml2ly does:
4
5     This multi-pass translator basically performs 5 passes:
6         Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
7                   and converts it to a MusicXML tree;
8         Pass 2a: converts that MusicXML tree into to
9                   a Music Score Representation (MSR) skeleton;
10        Pass 2b: converts that tree and the skeleton into a
11                  Music Score Representation (MSR);
12        Pass 3:  converts the MSR into a
13                  LilyPond Score Representation (LPSR);
14        Pass 4:  converts the LPSR to LilyPond source code
15                  and writes it to standard output.
16
17        Other passes are performed according to the options, such as
18        printing views of the internal data or printing a summary of the
19        score.
20
21        The activity log and warning/error messages go to standard error.
```

3 Options and help

xml2ly is equipped with a full-fledged set of options with the corresponding help. Since there are many options and the translation work is done in successive passes, the help is organized in a hierarchy of groups, each containing sub-groups of individual options called '*atoms*'.

3.1 Basic principles

Options are introduced on the command line either by '-' or '--', which can be used at will. There no difference between the two.

Each option has a short name and an optional long name. The latter is not needed if the short name is sufficiently explicit and not too long, such as '-jianpu', '-cubase', '-ambitus' or '-custos'.

Some options have their usual meaning in open-source software, such as '-h' (help), '-a' (about), and '-o' (output file name).

Some options name, short or long, share a common prefix, which allows them to be contracted, as in '-h=msr,lily', which is equivalent to '-msr, -lily', and '-trace=voices,notes', equivalent to '-trace-voices, -trace-notes'.

There are single-character options, which can be clustered: '-vac' is equivalent to: '-v, -a, -c'.

3.2 Introspection

One can obtain help on any specific group, sub-group or atom, such as:

```
1 menu@macbookprojm > xml2ly -option-name-help ambitus
2
3 --- Help for option 'ambitus' in subgroup "Engravers" of group "
4     LilyPond" ---
5
6 LilyPond (-hlily, -help-lilypond):
7     These lilypond control which LilyPond code is generated.
```

```

7 | -----
8 |
9 | Engravers (-hlpe, -help-lilypond-engravers):
10 |
11 |     -ambitus
12 |         Generate an ambitus range at the beginning of the staves/
        voices.

```

Some options have an optional value such as '-option-name-help', whose default value is... 'option-name-help':

```

1 | menu@macbookprojm > xml2ly -option-name-help
2 |
3 | --- Help for option 'onh' in subgroup "Options help" of group "Options
        and help" ---
4 |
5 | Options and help (-hoah, -help-options-and-help):
6 | -----
7 | Options help (-hoh, -help-options-help):
8 |
9 |     -onh, -option-name-help[=OPTION_NAME]
10 |         Print help about OPTION_NAME.
11 |         OPTION_NAME is optional, and the default value is 'onh'.

```

3.3 Trace options

xml2ly is equipped with a range of trace options, that are crucially needed by this author when testing and fine-tuning the code base.

The bulk of these options is placed in a group that is hidden by default:

```

1 | Trace (-ht, -help-trace) (hidden by default)
2 | -----

```

The interested reader can see them with the '-help-trace' group option:

```

1 | menu@macbookprojm > xml2ly -help=trace
2 |
3 | --- Help for group "Trace" ---
4 |
5 | Trace (-ht, -help-trace) (hidden by default)
6 |     There are trace options transversal to the successive passes,
7 |     showing what's going on in the various translation activities.
8 |     They're provided as a help to the maintainers, as well as for the
        curious.
9 |     The options in this group can be quite verbose, use them with small
        input data!
10 |     All of them imply '-tpasses, -trace-passes'.
11 | -----
12 | Options handling trace                                (-htoh, -help-trace-options-handling)
        :
13 |     -toah, -trace-oah
14 |         Write a trace of options and help handling to standard error.
15 |         This option should best appear first.
16 |     -toahd, -trace-oah-details
17 |         Write a trace of options and help handling with more details
        to standard error.
18 |         This option should best appear first.
19 | Score to voices                                (-htstv, -help-trace-score-to-voices)
        :
20 |     -t<SHORT_NAME>, -trace<LONG_NAME>
21 |         Trace SHORT_NAME/LONG_NAME in score to voices.
22 |         The 9 known SHORT_NAMES are:
23 |             score, pgroups, pgroupsd, parts, staves, st, schanges,
        voices and voicesd.

```

```

24         The 9 known LONG_NAMES are:
25         -score, -part-groups, -part-groups-details, -parts, -staves
26         .
26         ... ..

```

As can be seen, there are event options to trace the handling of options and help by xml2ly. The source code contains many instances of trace code, such as:

```

1 #ifdef TRACE_OAH
2     if (gTraceOah->fTraceVoices) {
3         gLogOstream <<
4             "Creating voice \"" << asString () << "\"" <<
5             endl;
6     }
7 #endif

```

Building xml2ly with tracing disabled only gains less than 5% in speed, this is why tracing is available by default.

3.4 Non-musical options

3.4.1 Timing measurements

There is a '-cpu' option to see show much time is spent in the various translation activities:

```

1 menu@macbookprojm > xml2ly -option-name-help cpu
2
3 --- Help for option 'cpu' in subgroup "CPU usage" of group "General"
4 ---
5 General (-hg, -help-general):
6 -----
7     CPU usage (-hgcpu, -help-general-cpu-usage):
8
9         -cpu, -display-cpu-usage
10         Write information about CPU usage to standard error.

```

In practise, most of the time is spent in passes 1 and 2b. The 'time' command is used to obtain the total run time, since xml2ly cannot account for input/output activities:

```

1 menu@macbookprojm > time xml2ly -aofn -cpu xmlsamples3.1/
2 ActorPreludeSample.xml
3 *** MusicXML warning *** xmlsamples3.1/ActorPreludeSample.xml:44: <
4 system-distance /> is not supported yet by xml2ly
5 ... ..
6 *** MusicXML warning *** xmlsamples3.1/ActorPreludeSample.xml:27761: <
7 direction/> contains 2 <words/> markups
8 Warning message(s) were issued for input lines 44, 45, 46, 551, 584,
9 732, 1121, 1215, 4724, 27761
10
11 Timing information:
12
13 Activity          Description          Kind    CPU (sec)
14 -----
15 Pass 1            build xmlelement tree from file  mandatory  0.268994
16 Pass 2a           build the MSR skeleton             mandatory  0.076413
17 Pass 2b           build the MSR                      mandatory  0.276732
18 Pass 3            translate MSR to LPSR              mandatory  0.056381
19 Pass 4            translate LPSR to LilyPond         mandatory  0.082213
20
21 Total            Mandatory  Optional
22 -----
23 0.760733         0.760733    0

```

```

21
22
23 real  0m0.814s
24 user  0m0.751s
25 sys 0m0.058s

```

This compares favorably with musicxml2ly measurements:

```

1 menu@macbookprojm > time musicxml2ly xmlsamples3.1/ActorPreludeSample.
  xml
2 musicxml2ly: Reading MusicXML from xmlsamples3.1/ActorPreludeSample.xml
  ...
3 musicxml2ly: Converting to LilyPond expressions...
4 ... ..
5 musicxml2ly: Converting to LilyPond expressions...
6 musicxml2ly: Output to 'ActorPreludeSample.ly'
7 musicxml2ly: Converting to current version (2.19.83) notations ...
8
9 real  0m4.113s
10 user  0m3.659s
11 sys 0m0.407s

```

3.4.2 Chords structure

In order to invert chords, as specified by the '<inversion>' element in MusicXML data, musicxml2ly knows the structure of many of them. This can be queried with the options in the 'Extra' group:

```

1 menu@macbookprojm > xml2ly -help=extra
2
3 --- Help for group "Extra" ---
4
5 Extra (-he, -help-extra):
6   These extra provide features not related to translation from MusicXML
   to other formats.
7   In the text below:
8     - ROOT_DIATONIC_PITCH should belong to the names available in
9       the selected MSR pitches language, "nederlands" by default;
10    - other languages can be chosen with the '-mpl, -msrPitchesLanguage
   ' option;
11    - HARMONY_NAME should be one of:
12      MusicXML chords:
13        "maj", "min", "aug", "dim", "dom",
14        "maj7", "min7", "dim7", "aug7", "halfdim", "minmaj7",
15        "maj6", "min6", "dom9", "maj9", "min9", "dom11", "maj11", "
   min11",
16        "dom13", "maj13", "min13", "sus2", "sus4",
17        "neapolitan", "italian", "french", "german"
18      Jazz-specific chords:
19        "pedal", "power", "tristan", "minmaj9", "domsus4", "domaug5",
20        "dommin9", "domaug9dim5", "domaug9aug5", "domaug11", "
   maj7aug11"
21   The single or double quotes are used to allow spaces in the names
22   and around the '=' sign, otherwise they can be dispensed with.
23   -----
24   Chords structures      (-hecs, -help-extra-chord-structures):
25     -scs, -show-chords-structures
26       Write all known chords structures to standard output.
27   Chords contents      (-hecc, -help-extra-chords-contents):
28     -sacc, -show-all-chords-contents PITCH
29       Write all chords contents for the given diatonic (semitones)
   PITCH,
30       supplied in the current language to standard output.

```

```

31 Chord details      (-hecd, -help-extra-chords-details):
32   -scd, -show-chord-details CHORD_SPEC
33       Write the details of the chord for the given diatonic (
34   semitones) pitch
35       in the current language and the given harmony to standard
36   output.
37   CHORD_SPEC can be:
38   'ROOT_DIATONIC_PITCH HARMONY_NAME'
39   or
40   "ROOT_DIATONIC_PITCH = HARMONY_NAME"
41   Using double quotes allows for shell variables substitutions,
42   as in:
43   HARMONY="maj7"
44   xml2ly -show-chord-details "bes ${HARMONY}"
45 Chord analysis      (-heca, -help-extra-chords-analysis):
46   -sca, -show-chord-analysis CHORD_SPEC
47       Write an analysis of the chord for the given diatonic (
48   semitones) pitch
49       in the current language and the given harmony to standard
50   output.
51   CHORD_SPEC can be:
52   'ROOT_DIATONIC_PITCH HARMONY_NAME INVERSION'
53   or
54   "ROOT_DIATONIC_PITCH = HARMONY_NAME INVERSION"
55   Using double quotes allows for shell variables substitutions,
56   as in:
57   HARMONY="maj7"
58   INVERSION=2
59   xml2ly -show-chord-analysis "bes ${HARMONY} ${INVERSION}"

```

For example, one can obtain the structure of the B^b dominant minor ninth chord's second inversion this way:

```

1 menu@macbookprojm > xml2ly -show-chord-analysis 'bes dommin9 2'
2 The analysis of chord 'bes dommin9' inversion 2 is:
3
4 Chord 'bes dommin9' inversion 2 contents, 5 intervals:
5   d      : majorThird
6   bes    : perfectUnison
7   ces    : minorNinth
8   aes    : minorSeventh
9   f      : perfectFifth
10
11 Chord 'bes dommin9' inversion 2 inner intervals:
12   f      -> aes    : minorThird      (perfectFifth      ->
13   minorSeventh)
14   f      -> ces    : diminishedFifth (perfectFifth      ->
15   minorNinth)
16   f      -> bes    : perfectFourth   (perfectFifth      ->
17   perfectUnison)
18   f      -> d      : majorSixth      (perfectFifth      ->
19   majorThird)
20
21   aes    -> ces    : minorThird      (minorSeventh      ->
22   minorNinth)
23   aes    -> bes    : majorSecond     (minorSeventh      ->
24   perfectUnison)
25   aes    -> d      : augmentedFourth (minorSeventh      ->
26   majorThird)
27
28   ces    -> bes    : majorSeventh    (minorNinth        ->
29   perfectUnison)
30   ces    -> d      : augmentedSecond (minorNinth        ->
31   majorThird)

```

```
23      bes    -> d      : majorThird      (perfectUnison      ->
24      majorThird)
25      This chord contains 2 tritons
```

4 Building the xmlelement tree

5 The MSR graph

6 The LPSR graph

7 LilyPond code generation

Listings

List of Figures

1	libmusicxml2 architecture	2
---	-------------------------------------------	---

Contents

1	Acknowledgements	1
2	Overview of xml2ly	1
2.1	Why xml2ly?	1
2.2	What xml2ly does	3
3	Options and help	3
3.1	Basic principles	3
3.2	Introspection	3
3.3	Trace options	4
3.4	Non-musical options	5
3.4.1	Timing measurements	5
3.4.2	Chords structure	6
4	Building the xmlelement tree	8
5	The MSR graph	8
6	The LPSR graph	8
7	LilyPond code generation	8