User's guide to `libmusicxml2`

Jacques Menu

March 25, 2021

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-phase visitor designe 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 Options and help (OAH)

2.1 OAH basics

- OAH (Options And Help) is supposed to be pronounced something close to "whaaaah!" The intonation is left to the speaker, though... And as the saying goes: "OAH? oahy not!"
- options handling is organized as a hierarchical, instrospective set of classes. An options and its corresponding help are grouped in a single object.
- the options can be supplied thru:
  - the command line, in argv. This allows for mixed options and arguments in any order, à la GNU;
  - the API function such as musicxmlfile2lilypond(), in an options vector.
- oahElement is the super-class of all options types, including groups and subgroups. It contains a short name and a long name, as well as a decription. Short and long names can be used and mixed at will in the command line and in option vectors (API), as well as '-' and '–'. The short name is mandatory, but the long name may be empty if the short name is explicit enough.

- prefixes such '-t=' and -help=' allow for a contracted form of options. For example, -t=meas,notes is short for '-t-meas, -tnotes'. A oahPrefix contains the prefix name, the ersatz by which to replace it, and a description.
- a oahHandler contains a list of oahGroup's, each handled in a pair or .h/.cpp files such as msrOah.h and msrOahGroup.cpp, and a list of options prefixes.
- a oahGroup contains a list of oahSubGroup's and an upLink to the containing oahHandler.
- a oahSubGroup contains a list of oahAtom's and an upLink to the containing oahGroup.
- each oahAtom contains an atomic option and the corresponding help, and an upLink to the containing oahSubGroup.

2.2 Features

- partial help can be obtained, i.e. help about any group, subgroup or atom, showing the path in the hierarchy down to the corresponding option.
- there are various subclasses of oahAtom such as oahIntegerAtom, oahBooleanAtom and oahRationalAtom, to control options values of common types.
- oahThreeBooleansAtom, for example, allows for three boolean settings to be controlled at once with a single option.
- oahAtomStoringAValueInAVariable describes options for which a value is supplied in the command line or in option vectors (API).
- a class such as lpsrPitchesLanguageAtom is used to supply a string value to be converted into an internal enumerated type.
- a oahCombinedBooleansAtom contains a list of boolean atoms to manipulate several such atoms as a single one, see the 'cubase' combined booleans atom in mxmlTree2msrOah.cpp.
- oahMultiplexBooleansAtom contains a list of boolean atoms sharing a common prefix to display such atoms in a compact manner, see the 'ignore-redundant-clefs' multiplex booleans atom in mxmlTree2msrOah.cpp.
- storing options and the corresponding help in oahGroup's makes it easy to re-use them. For example, xml2ly and xml2lbr have their three first passes in common, (up to obtaining the MSR description of the score), as well as the corresponding options and help.
- oahAtomsCollection.h/.cpp contains a bunch of general purpose options such as oahContactAtom, oahFloatAtom and oahLengthAtom.
- a regular handler (used by default unless the '-insider' option is used), presents the options and help grouped by subject, such as voices and tuplets. It uses an insider handler, which groups them by internal representation and conversion pass. This is how options groups are re-used for various translators such as xml2ly, xml2brl and xml2xml.

2.3 Handling

- each option short name and non-empty long name must be unique in a given handler, to avoid ambiguities.
- an executable main() calls applyOptionsAndArgumentsFromArgcAndArgv(), in which:
- handleOptionName() handles the option names
- handleOptionValueOrArgument() handle the values that may follow an atom name and the arguments to the executable.
- contracted forms are expanded in handleOptionName() before the resulting, uncontracted options are handled.
- options handling works in two passes:
- the first one creates a list of 'elementUse' instances from argc/argv or an options vector;

- the second one traverses this list to apply the options that are used.
- the options are applied the the applyElement(), applyAtomWithValue() And applyAtomWithDefaultValue() methods.
- handleOptionValueOrArgument() associatiates the value to the (preceding) fPendingAtomWithValue if not null, or appends it fHandlerArgumentsVector to otherwise.
- fPendingArgvAtomWithValue is used in argv contents handling to associate an option name with it value, which is the next element in argv.

3 Overview of `xml2ly`

The initial name of `xml2ly`, when it started as a clone of `xml2guido`, was `xml2lilypond`. Both Dominique Fober and Werner Lemberg, an early tester, found it too long, and they chose `xml2ly` among other names this author proposed to them.

3.1 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 maintainance 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 developper, 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.

3.2 What `xml2ly` does

The architecture of `libmusicxml2` s presented in figure 1, page 4, i figure **??**. It shows the place of `xml2ly` in the whole.

`xml2ly` performs the 4 hops from MusicXML to LilyPond to translate the former into the latter, using these passes:

$1 \Rightarrow 4 \Rightarrow 5 \Rightarrow 7$

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

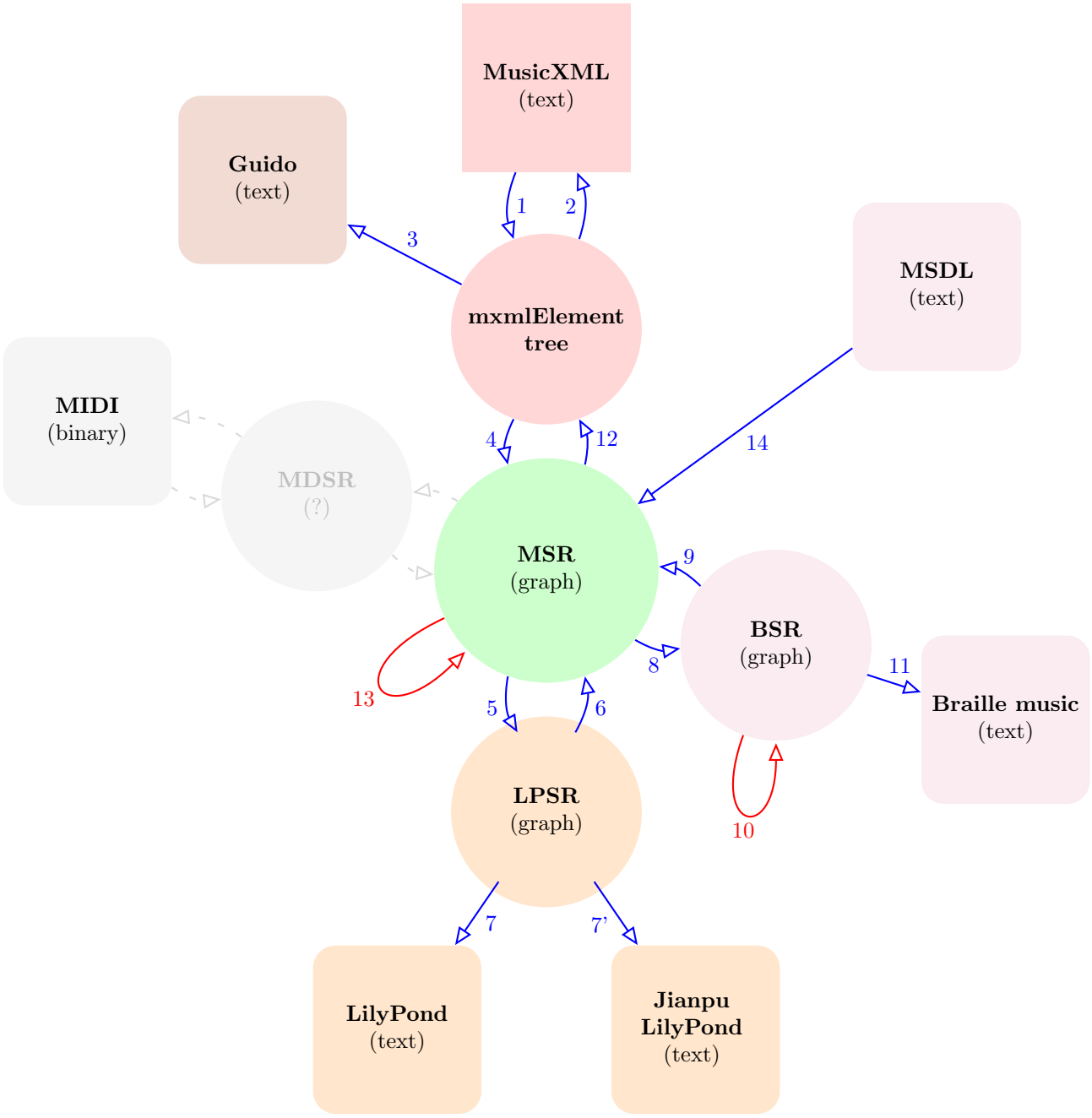
```
menu@macbookprojm > xml2ly -about

What xml2ly does:

    This multi-pass translator basically performs 5 passes:
        Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
                 and converts it to a MusicXML tree;
        Pass 2a: converts that MusicXML tree into to
                 a Music Score Representation (MSR) skeleton;
        Pass 2b: converts that tree and the skeleton into a
                 Music Score Representation (MSR);
        Pass 3:  converts the MSR into a
                 LilyPond Score Representation (LPSR);
        Pass 4:  converts the LPSR to LilyPond source code
                 and writes it to standard output.

    Other passes are performed according to the options, such as
    printing views of the internal data or printing a summary of the
   score.
```

Figure 1: `libmusicxml2` architecture




```
The activity log and warning/error messages go to standard error.
```

4 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*'.

4.1 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'.

4.2 Introspection

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

```
menu@macbookprojm > xml2ly -option-name-help ambitus

--- Help for option 'ambitus' in subgroup "Engravers" of group "
    LilyPond" ---

LilyPond (-hlily, -help-lilypond):
  These lilypond control which LilyPond code is generated.

--------------------------
  Engravers (-hlpe, -help-lilypond-engravers):

    -ambitus
          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':

```
menu@macbookprojm > xml2ly -option-name-help

--- Help for option 'onh' in subgroup "Options help" of group "Options
    and help" ---

Options and help (-hoah, -help-options-and-help):
--------------------------
  Options help (-hoh, -help-options-help):

    -onh, -option-name-help[=OPTION_NAME]
          Print help about OPTION_NAME.
          OPTION_NAME is optional, and the default value is 'onh'.
```

4.3 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:

```
  Trace (-ht, -help-trace) (hidden by default)
  --------------------------
```

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

```
menu@macbookprojm > xml2ly -help=trace

--- Help for group "Trace" ---

Trace (-ht, -help-trace) (hidden by default)
  There are trace options transversal to the successive passes,
  showing what's going on in the various translation activities.
```

```
  They're provided as a help to the maintainers, as well as for the
   curious.
  The options in this group can be quite verbose, use them with small
   input data!
  All of them imply '-tpasses, -trace-passes'.
-------------------------
  Options handling trace          (-htoh, -help-trace-options-handling)
   :
    -toah, -trace-oah
          Write a trace of options and help handling to standard error.
          This option should best appear first.
    -toahd, -trace-oah-details
          Write a trace of options and help handling with more details
   to standard error.
          This option should best appear first.
  Score to voices                 (-htstv, -help-trace-score-to-voices)
   :
    -t<SHORT_NAME>, -trace<LONG_NAME>
          Trace SHORT_NAME/LONG_NAME in score to voices.
          The 9 known SHORT_NAMEs are:
            score, pgroups, pgroupsd, parts, staves, st, schanges,
   voices and voicesd.
          The 9 known LONG_NAMEs are:
            -score, -part-groups, -part-groups-details, -parts, -staves
   .
... ... ... ... ... ...
```

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:

```
#ifdef TRACE_OAH
  if (gTraceOah->fTraceVoices) {
    gLogOstream <<
      "Creating voice \"" << asString () << "\"" <<
      endl;
  }
#endif
```

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

4.4 Non-musical options

4.4.1 Timing measurements

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

```
menu@macbookprojm > xml2ly -option-name-help cpu

--- Help for option 'cpu' in subgroup "CPU usage" of group "General"
   ---

General (-hg, -help-general):
-------------------------
  CPU usage (-hgcpu, -help-general-cpu-usage):

    -cpu, -display-cpu-usage
          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:

```
menu@macbookprojm > time xml2ly -aofn -cpu xmlsamples3.1/
   ActorPreludeSample.xml
```

```
*** MusicXML warning *** xmlsamples3.1/ActorPreludeSample.xml:44: <system-distance /> is not supported yet by xml2ly
... ... ... ... ...
*** MusicXML warning *** xmlsamples3.1/ActorPreludeSample.xml:27761: <direction/> contains 2 <words/> markups
Warning message(s) were issued for input lines 44, 45, 46, 551, 584, 732, 1121, 1215, 4724, 27761

Timing information:

Activity                        Description        Kind   CPU (sec)
--------  --------------------------------  ---------  ---------

Pass 1    build xmlelement tree from file  mandatory   0.268994
Pass 2a   build the MSR skeleton           mandatory   0.076413
Pass 2b   build the MSR                    mandatory   0.276732
Pass 3    translate MSR to LPSR            mandatory   0.056381
Pass 4    translate LPSR to LilyPond       mandatory   0.082213

Total       Mandatory   Optional
-------     ---------   ---------
0.760733     0.760733    0


real  0m0.814s
user  0m0.751s
sys 0m0.058s
```

This compares favorably with `musicxml2ly` measurements:

```
menu@macbookprojm > time musicxml2ly xmlsamples3.1/ActorPreludeSample.xml
musicxml2ly: Reading MusicXML from xmlsamples3.1/ActorPreludeSample.xml ...
musicxml2ly: Converting to LilyPond expressions...
... ... ... ... ...
musicxml2ly: Converting to LilyPond expressions...
musicxml2ly: Output to 'ActorPreludeSample.ly'
musicxml2ly: Converting to current version (2.19.83) notations ...

real  0m4.113s
user  0m3.659s
sys 0m0.407s
```

4.4.2 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:

```
menu@macbookprojm > xml2ly -help=extra

--- Help for group "Extra" ---

Extra (-he, -help-extra):
  These extra provide features not related to translation from MusicXML
    to other formats.
  In the text below:
    - ROOT_DIATONIC_PITCH should belong to the names available in
      the selected MSR pitches language, "nederlands" by default;
    - other languages can be chosen with the '-mpl, -msrPitchesLanguage
   ' option;
    - HARMONY_NAME should be one of:
```

```
        MusicXML chords:
          "maj", "min", "aug", "dim", "dom",
          "maj7", "min7", "dim7", "aug7", "halfdim", "minmaj7",
          "maj6", "min6", "dom9", "maj9", "min9", "dom11", "maj11", "
   min11",
          "dom13", "maj13", "min13", "sus2", "sus4",
          "neapolitan", "italian", "french", "german"
        Jazz-specific chords:
          "pedal", "power", "tristan", "minmaj9", "domsus4", "domaug5",
          "dommin9", "domaug9dim5", "domaug9aug5", "domaug11", "
   maj7aug11"
  The single or double quotes are used to allow spaces in the names
  and around the '=' sign, otherwise they can be dispensed with.
-------------------------
  Chords structures    (-hecs, -help-extra-chord-structures):
    -scs, -show-chords-structures
          Write all known chords structures to standard output.
  Chords contents      (-hecc, -help-extra-chords-contents):
    -sacc, -show-all-chords-contents PITCH
          Write all chords contents for the given diatonic (semitones)
   PITCH,
          supplied in the current language to standard output.
  Chord details        (-hecd, -help-extra-chords-details):
    -scd, -show-chord-details CHORD_SPEC
          Write the details of the chord for the given diatonic (
   semitones) pitch
          in the current language and the given harmony to standard
   output.
          CHORD_SPEC can be:
          'ROOT_DIATONIC_PITCH HARMONY_NAME'
          or
          "ROOT_DIATONIC_PITCH = HARMONY_NAME"
          Using double quotes allows for shell variables substitutions,
    as in:
          HARMONY="maj7"
          xml2ly -show-chord-details "bes ${HARMONY}"
  Chord analysis       (-heca, -help-extra-chords-analysis):
    -sca, -show-chord-analysis CHORD_SPEC
          Write an analysis of the chord for the given diatonic (
   semitones) pitch
          in the current language and the given harmony to standard
   output.
          CHORD_SPEC can be:
          'ROOT_DIATONIC_PITCH HARMONY_NAME INVERSION'
          or
          "ROOT_DIATONIC_PITCH = HARMONY_NAME INVERSION"
          Using double quotes allows for shell variables substitutions,
    as in:
          HARMONY="maj7"
          INVERSION=2
          xml2ly -show-chord-analysis "bes ${HARMONY} ${INVERSION}"
```

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

```
menu@macbookprojm > xml2ly -show-chord-analysis 'bes dommin9 2'
The analysis of chord 'bes dommin9' inversion 2 is:

  Chord 'bes dommin9' inversion 2 contents, 5 intervals:
    d     : majorThird
    bes   : perfectUnison
    ces   : minorNinth
    aes   : minorSeventh
    f     : perfectFifth
```

```
10
11  Chord 'bes dommin9' inversion 2 inner intervals:
12      f      -> aes    : minorThird           (perfectFifth          ->
     minorSeventh)
13      f      -> ces    : diminishedFifth      (perfectFifth          ->
     minorNinth)
14      f      -> bes    : perfectFourth        (perfectFifth          ->
     perfectUnison)
15      f      -> d      : majorSixth           (perfectFifth          ->
     majorThird)
16
17      aes    -> ces    : minorThird           (minorSeventh          ->
     minorNinth)
18      aes    -> bes    : majorSecond          (minorSeventh          ->
     perfectUnison)
19      aes    -> d      : augmentedFourth      (minorSeventh          ->
     majorThird)
20
21      ces    -> bes    : majorSeventh         (minorNinth            ->
     perfectUnison)
22      ces    -> d      : augmentedSecond      (minorNinth            ->
     majorThird)
23
24      bes    -> d      : majorThird           (perfectUnison         ->
     majorThird)
25  This chord contains 2 tritons
```

5 Building the xmlelement tree

6 The MSR graph

7 The LPSR graph

8 LilyPond code generation

Listings

List of Figures

Contents