MusicFormats command line user guide

https://github.com/jacques-menu/musicformats

v0.9.61 (February 25, 2022)

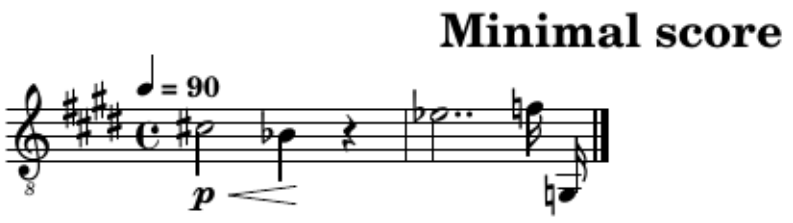
Jacques Menu

MusicFormats is open source software, available with source code and documentation at https://github.com/jacques-menu/musicformats. It is written in C++11 and provides a set of music scores representations and converters between various textual music scores formats. Building it only requires a C++11 compiler and `cmake`.

This document shows how to use the MusicFormats library, both from the command line and from within applications. It is part of the MusicFormats documentation, and can be found at MusicFormatsCLIUserGuide.pdf.

MusicFormats can be used from the command line on Linux, Windows and Mac OS. The API also allows it to be used from applications, including in Web sites.

```
jacquesmenu@macmini > xml2ly -about
What xml2ly does:

    This multi-pass converter 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
                 a first Music Score Representation (MSR) skeleton;
        Pass 2b: populates the first MSR skeleton from the MusicXML tree
                 to get a full MSR;
        Pass 3:  converts the first MSR into a second MSR to apply options
        Pass 4:  converts the second MSR into a
                 LilyPond Score Representation (LPSR);
        Pass 5:  converts the LPSR to LilyPond code
                 and writes it to standard output.

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

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



Contents

List of Figures

Part I

Preamble

Chapter 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 converter without his work being 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 design pattern, which this author uses extensively in his own code. The interested reader can find information about that in `libmusicxml2.pdf`, and more technical details in `MusicFormatsMaintainanceGuide.pdf`.

`xml2ly` and some of the specific examples presented in this document started as this author's contribution to `libmusicxml2`, and was later moved to a separate GitHub repository for practical reasons.

Chapter 2

About this document

This document is organized in four parts:

- the part II lets the user discover the library, as well as its architecture, see section 3, [The MusicFormats architecture], page 9;
- then the options and help so-called OAH infrastructure provided by the library is presented in part III;
- the part IV is dedicated to the handling or warnings and errors;
- the part V presents the multiple languages support provided by MusicFormats;
- parts VI to IX show the specific features of the various converter;
- and finally, there is a comprehensive set of indexes.

The use of MusicFormats through its APIs is described in a specific documentation, to be found at `MusicFormatsAPIUserGuide.pdf`. This is intended for users who create applications such as Web sites that do not use command line commands, but call functions provided by the library instead. The exact same functionality is available this way.

In fact, the command line versions of the services merely use these API functions.

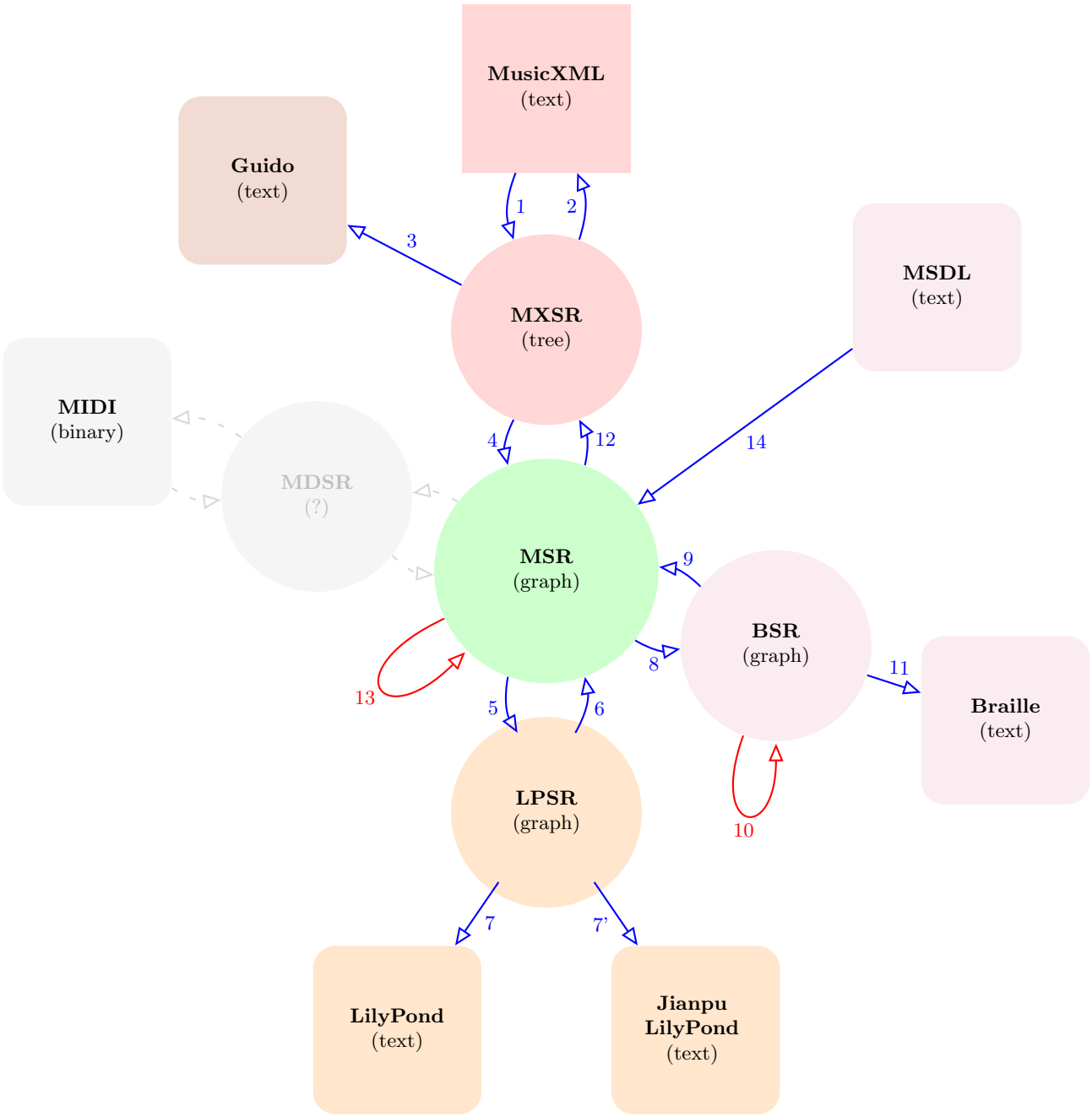
Part II

Discovering MusicFormats

Chapter 3

The MusicFormats architecture

Figure 3.1: The MusicFormats architecture



The picture at figure 3.1, [Architecture], page 9, shows how MusicFormats is structured:

- central to MusicFormats is MSR, an internal fine-grained representation of the musical contents of music scores;
- immediately around it, the round boxes are other (internal) representations used by various formats unitary conversions;
- the outermost square boxes are the (external) formats that MusicFormats supports;
- the numbered arrows are conversion steps between formats and/or representations. The numbers indicate roughly the order in which they were added to the library.

  Some conversions are two-way, such as that of MXSR to MSR and back. Others are one-way, such as the conversion of LSPR to LilyPond text;
- the red arrows are conversions of a representation to the same one. These are meant to offer options to modify the contents of those representations;
- the dimmed, dashed boxes and arrows indicate items not yet available or supported.

Decomposing the conversion work into successive steps has many advantages:

- each step concentrates on a subset of the tasks to be performed without interfering with the others. For example, converting MusicXML text to MSR has nothing to do with LilyPond;
- development and debugging is therefore much easier than with a single, huge bulk of code;
- most important still, this architecture allows the *reuse* of the steps, which are combined to assemble the higher-level converters;
- icing on the cake, the options and help associated with the various steps are combined to obtain the options and help for the converters and generators.

Technically, the conversion steps are called *passes*, a term that comes from the compiler writing field. We shall use it throughout this document.

Chapter 4

A first example

Before presenting the MusicFormats library in detail, let's get an idea of what it has to offer. The commands used in this chapter will be explained in later chapters.

4.1 Raw `xml2ly` usage

MusicXML is a textual representation of music scores, that can be produced by scanning score images or exported from GUI scoring applications. It has been designed to facilitate the sharing of scores across applications, which represent scores their own way.

MusicFormats provides `xml2ly` that converts MusicXML data to LilyPond code.

In computer science, the simplest example one can write with a language is often named 'Hello World'. MusicFormats abides to this rule, suppling basic/HelloWorld.xml:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/HelloWorld.xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC
    "-//Recordare//DTD MusicXML 3.0 Partwise//EN"
    "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise version="3.0">
  <work>
    <work-title>Hello World!</work-title>
    </work>
  <!-- A very minimal MusicXML example -->
  <part-list>
    <score-part id="P1">
      <part-name>Music</part-name>
    </score-part>
  </part-list>
  <part id="P1">
<!--========================================================-->
    <measure number="1">
  <!-- A very minimal MusicXML example, part P1, measure 1 -->
      <attributes>
        <divisions>1</divisions>
        <key>
          <fifths>0</fifths>
        </key>
        <time>
          <beats>4</beats>
          <beat-type>4</beat-type>
        </time>
        <clef>
          <sign>G</sign>
          <line>2</line>
```

```
        </clef>
      </attributes>
  <!-- A very minimal MusicXML example, part P1, measure 1, before first note -->
      <note>
        <pitch>
          <step>C</step>
          <octave>4</octave>
        </pitch>
        <duration>4</duration>
        <type>whole</type>
      </note>
    </measure>
<!--=========================================================-->
  </part>
</score-partwise>
```

4.2 Redirecting the output and error messages to files

By default, the standard output and error streams are directed to the terminal in which the `xml2ly` command has been submitted.

Let's consider the case of basic/UnknownMaintainerIdentification.xml:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly basic/
    UnknownMaintainerIdentification.xml
*** MusicXML warning *** basic/UnknownMaintainerIdentification.xml:11: creator type "
    maintainer" is unknown
\version "2.22.0"

% Pick your choice from the next two lines as needed
%myBreak = { \break }
myBreak = {}

% Pick your choice from the next two lines as needed
%myPageBreak = { \pageBreak }
myPageBreak = {}

\header {
    title                = "Hello World!"
    workTitle            = "Hello World!"
    title                = "Hello World!"
}

\paper {
}

\layout {
    \context {
      \Score
      autoBeaming = ##f % to display tuplets brackets
    }
    \context {
      \Voice
    }
}

Part_POne_Staff_One_Voice_One = \absolute {
    \language "nederlands"
    \key c \major
    \numericTimeSignature \time 4/4

    \clef "treble"
    c'1 | % 2
```

```
    \barNumberCheck #2
    | % 2
    \barNumberCheck #2
}

\book {
    \score {
        <<

            \new Staff = "Part_POne_Staff_One"
            \with {
            }
            <<
                \context Voice = "Part_POne_Staff_One_Voice_One" <<
                    \Part_POne_Staff_One_Voice_One
                >>
            >>

        >>

        \layout {
            \context {
              \Score
              autoBeaming = ##f % to display tuplets brackets
            }
            \context {
              \Voice
            }
        }

        \midi {
            \tempo 16 = 360
        }
    }

}
Warning message(s) were issued for input line 11
```

The standard output and error streams are merged into the terminal window, which may not be satisfactory. This behaviour can be changed using the shell's redirection operators:

- `>`: redirects the standard output stream to a file;
- `2>`: redirects the standard error stream to a file.

Thus, after executing:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly basic/
    UnknownMaintainerIdentification.xml > output.ly 2> error.txt
```

`output.ly` contains the LilyPond code produced, and `error.txt` contains:

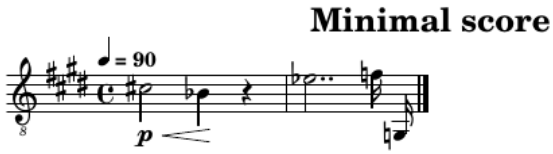
```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat error.txt
*** MusicXML warning *** basic/UnknownMaintainerIdentification.xml:11: creator type "
    maintainer" is unknown
Warning message(s) were issued for input line 11
```

4.3 The need for options

Using `xml2ly` as in the previous section is somewhat limited. This command:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly basic/
    MinimalScore.xml > MinimalScore.ly
```

leads to this score:



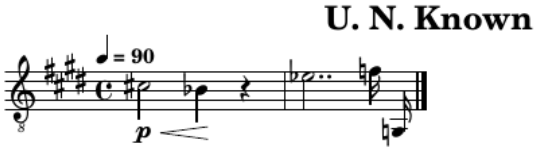
What if we want to change the title in the LilyPond output? This is where options come into play. One of them is option `-query`:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -query title
--- Help for atom "title" in subgroup "Header"
    -title STRING
          Set 'title' to STRING in the LilyPond code \header.
```

Using it this way:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly basic/
    MinimalScore.xml -title "U. N. Known" > MinimalScore.ly
```

produces that score:



The effect of the `-title` option is for `xml2ly` to generate this at the beginning of the LilyPond output:

```
\version "2.22.0"

% ... ... ...

\header {
    title                   = "U. N. Known"
    % ... ... ...
}

% ... ... ...
```

We could of course add this `title` setting by hand after `xml2ly` has produced LilyPond code, but all MusicFormats is about is to *automate* such things as much as possible.

This is why there are many options to the MusicFormats tools, which in turn explains why OAH, a powerfull options and help handling infrastructure, is provided as part of the library.

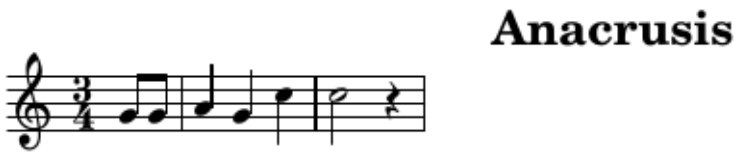
4.4 The passes at work

The passes involved in a conversion can be seen with suitable options:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -auto-output-file
    -name -trace-passes -display-cpu-usage basic/Anacrusis.xml

%--------------------------------------------------------------
  Handle the options and arguments from argc/argv
%--------------------------------------------------------------
This is xml2ly v0.9.52 (November 29, 2021) from MusicFormats v0.9.60 (January 25, 2022)
Launching the conversion of "basic/Anacrusis.xml" to LilyPond
Time is Thursday 2022-02-03 @ 14:19:32 CET
The command line is:
  xml2ly -auto-output-file-name -trace-passes -cpu basic/Anacrusis.xml
or with options long names:
  xml2ly -auto-output-file-name -trace-passes -display-cpu-usage basic/Anacrusis.xml
or with options short names:
   -aofn -tpasses -cpu basic/Anacrusis.xml
LilyPond code will be written to Anacrusis.ly
The command line options and arguments have been analyzed

%--------------------------------------------------------------
  Pass 1: Create an MXSR reading a MusicXML file
%--------------------------------------------------------------
% MusicXML data uses UTF-8 encoding

%--------------------------------------------------------------
  Pass 2a: Create an MSR skeleton from the MXSR
%--------------------------------------------------------------

%--------------------------------------------------------------
  Pass 2b: Populate the MSR skeleton from MusicXML data
%--------------------------------------------------------------

<!--=== part "P1", line 60 ===-->

%--------------------------------------------------------------
  Pass 3: Convert the first MSR into a second MSR
%--------------------------------------------------------------

%--------------------------------------------------------------
  Pass 4: Convert the second MSR into an LPSR
%--------------------------------------------------------------

Opening file 'Anacrusis.ly' for writing

%--------------------------------------------------------------
  Pass 5: Convert the LPSR score to LilyPond code
%--------------------------------------------------------------
Timing information:

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

          Handle the options and arguments from argc/argv      mandatory     0.02798
Pass 1    Create an MXSR reading a MusicXML file               mandatory     0.00420
Pass 2a   Create an MSR skeleton from the MXSR                 mandatory     0.00191
Pass 2b   Populate the MSR skeleton from MusicXML data         mandatory     0.00314
Pass 3    Convert the first MSR into a second MSR              mandatory     0.00069
Pass 4    Convert the second MSR into an LPSR                  mandatory     0.00090
Pass 5    Convert the LPSR score to LilyPond code              mandatory     0.00156

Total (sec)  Mandatory  Optional
-----------  ---------  ---------
0.04037      0.04037    0.00000
```

The optional passes are those that display MusicFormats internal data. They are triggered by suitable options, see section 16.6, [Displaying MusicFormats internal data], page 64.

The resulting `Anacrusis.ly` file leads to this score where submitted to LilyPond:



Chapter 5

More examples

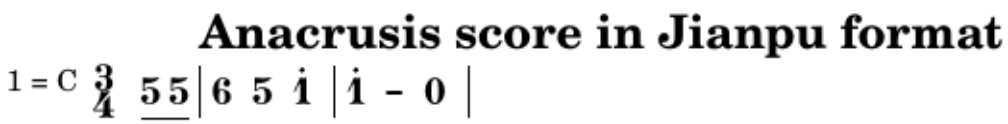
5.1 Jianpu output

`xml2ly` can be used to produce scores in the Jianpu numeric notation format, in which the notes piches are numbers relative to the scale instead of graphic elements:

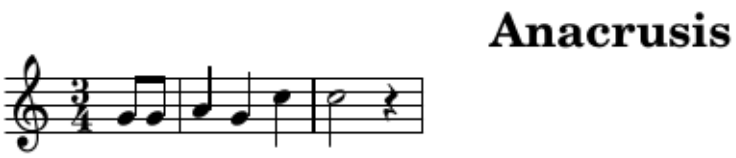
```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly basic/Anacrusis.xml -output-file-name Anacrusis_Jianpu.ly -jianpu -title "Anacrusis score in Jianpu format"
```

This option needs lilypond-Jianpu to be accessible to LilyPond. This is available at https://github.com/nybbs2003/lilypond-Jianpu/jianpu10a.ly.

The key in this example is C major. The resulting `MinimalScore_Jianpu.ly` leads to:



This is to be compared with:

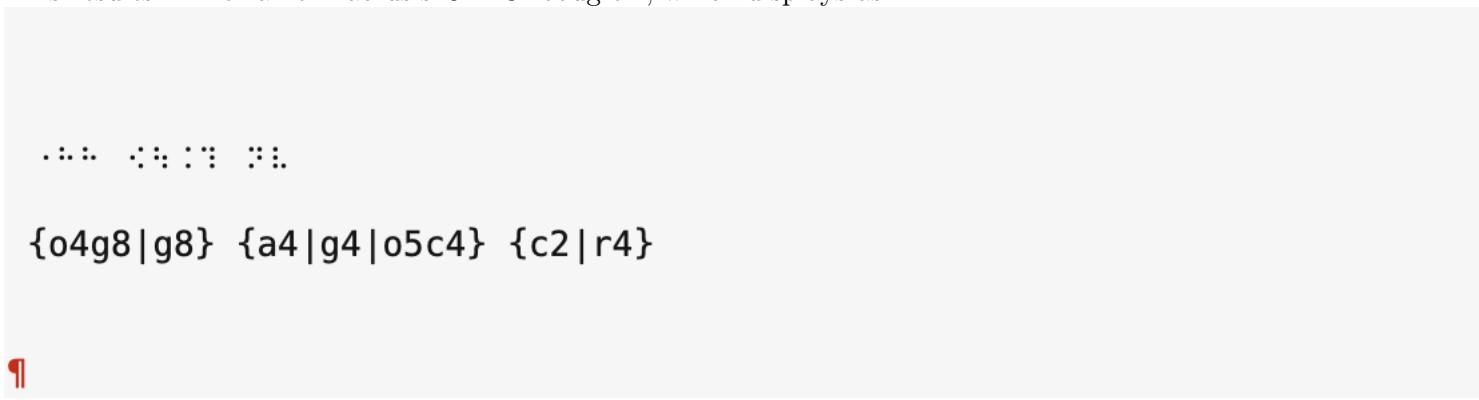


5.2 Braille output

The same score can also be produced in braille, with an interpretation of the 6-doc cells for debug in this case, by `xml2brl`:

```
jacquesmenu@macmini > xml2brl basic/Anacrusis.xml -auto-output-file-name -utf8d --use-encoding-in-file-name
```

This results in fileNameAnacrusis_UTF8Debug.brf, which displays as:



The `o*` indicate the octave, and notes pitches and rests use LilyPond syntax.

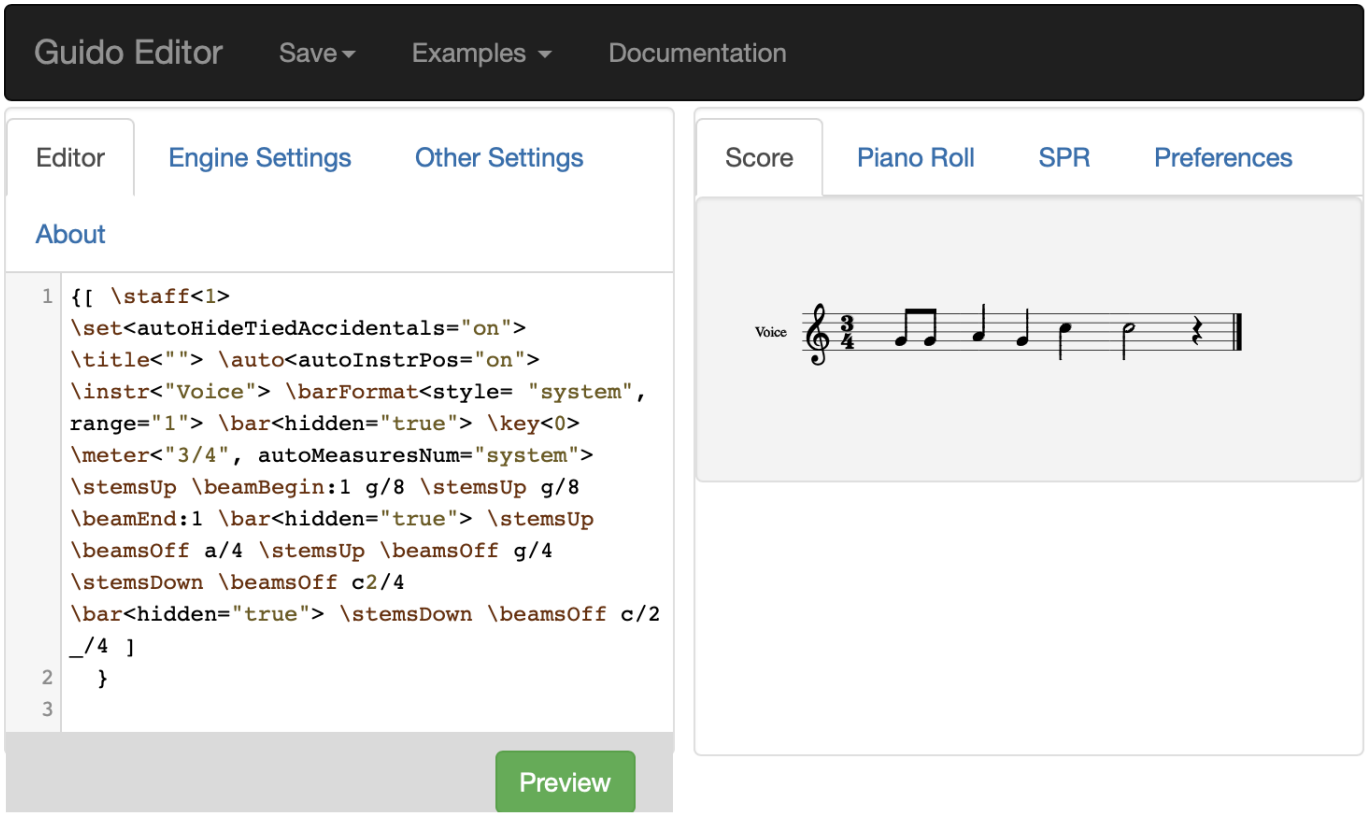
5.3 Guido output

Guido is a textual representation of music scores. Converting MusicXML to Guido is the reason why Dominique Fober created `libmusicxml2` in the first place.

MusicFormats's `xml2gmn` is a multi-pass converter when `xml2guido` a part of `libmusicxml2`, has two passes and only usea a MXSR representation.

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2gmn basic/Anacrusis.
    xml
{[ \staff<1> \set<autoHideTiedAccidentals="on"> \title<""> \auto<autoInstrPos="on"> \instr
    <"Voice"> \barFormat<style= "system", range="1"> \bar<hidden="true"> \key<0> \meter<"
    3/4", autoMeasuresNum="system"> \stemsUp \beamBegin:1 g/8 \stemsUp g/8 \beamEnd:1 \bar
    <hidden="true"> \stemsUp \beamsOff a/4 \stemsUp \beamsOff g/4 \stemsDown \beamsOff c2
    /4 \bar<hidden="true"> \stemsDown \beamsOff c/2 _/4 ]
  }
```

This can be viewed and edited on Dominique Fober's https://guidoeditor.grame.fr/#:



5.4 MusicXML output

`xml2xml` is meant for applying transformations to MusicXML data. For example, `basic/Anacrusis.xml` contains:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/Anacrusis.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.0 Partwise//EN" "http://www.
    musicxml.org/dtds/partwise.dtd">
<score-partwise>
  <work>
    <work-title>Anacrusis</work-title>
    </work>

  <!-- ... ... ... />

  </score-partwise>
```

We can obtain another MusicXML file with this command, changing the work title, adding a work number and using an alto clef instead of a treble clef:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2xml basic/Anacrusis.
    xml -output-file-name Anacrusis_From_xml2xml.xml -msr-replace-clef treble=alto -work-
    title "Anacrusis from xml2ml with alto clef" -work-number 317
```

The resulting file `Anacrusis_From_xml2xml.xml` contains:

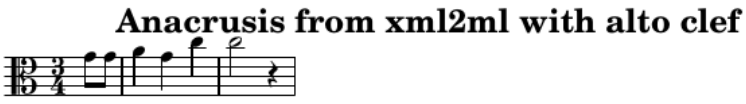
```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat
    Anacrusis_From_xml2xml.xml
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
      "http://www.musicxml.org/dtds/partwise.dtd">
```

```
<score-partwise version="3.1">
    <!--
================================================
Generated by xml2xml v0.9.5 (October 6, 2021)
on Monday 2022-02-14 @ 08:05:54 CET
from "basic/Anacrusis.xml"
================================================
-->
    <work>
        <work-number>317</work-number>
        <work-title>Anacrusis from xml2ml with alto clef</work-title>
    </work>

  <!-- ... ... ... />

    <part id="P1">
        <measure number="0">
            <attributes>
                <divisions>2</divisions>
                <key>
                    <fifths>0</fifths>
                </key>
                <time>
                    <beats>3</beats>
                    <beat-type>4</beat-type>
                </time>
                <clef>
                    <sign>C</sign>
                    <line>3</line>
                </clef>
            </attributes>

  <!-- ... ... ... />

  </score-partwise>
```

Let's convert this to LilyPond with `xml2ly` into `Anacrusis_From_xml2xml.ly`:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly
    Anacrusis_From_xml2xml.xml -auto-output-file-name
```

The resulting score is:



Chapter 6

Multiple files conversion

6.1 Using the `find` shell command

On Mac OS™ and Linux, converting all the MusicXML files in a given folder can be achieved with a `find` command, here using `xml2ly`:

```
find . -name "*.xml" -exec xml2ly {} \;
```

6.2 Using a `Makefile`

MusicFormats supplies a `Makefile` in `files/musicxmlfiles`, which can be copied to any folder at will to convert all the `.xml` files in it.

Let's show how to proceed with the MusicXML files provided with MusicFormats. To convert all the MusicXML files in a sub-folder of `files/musicxmlfiles`, one can:

- `cd` to this sub-folder;

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cd keys/
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys >
```

- create a *symbolic link* to this `Makefile` in the given sub-folder – this is to be done only once:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > ln -s ../
    Makefile

jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > ls -sal
total 24
 0 drwxr-xr-x    5 jacquesmenu  staff   160 Feb 25 14:12 .
 0 drwxr-xr-x  108 jacquesmenu  staff  3456 Feb 25 07:46 ..
16 -rw-r--r--@   1 jacquesmenu  staff  6148 Feb 25 12:51 .DS_Store
 8 -rw-r--r--    1 jacquesmenu  staff  2634 Apr 22  2021 HumdrumScotKeys.xml
 0 lrwxr-xr-x    1 jacquesmenu  staff    11 Feb 25 14:12 Makefile -> ../Makefile
```

  Using a symbolic link avoids copying the `Makefile` into multiple sub-folders.
  That also ensures that any change to the `Makefile` in `libmusicxml/src/files/musicxmlfiles` is taken into account in the sub-folders.

- use this `Makefile` with the `make` command as shown below.

This Makefile can provide help about its use:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > make help
Makefile for converting MusicXML files to LilyPond scores.
Supplied as part of the MusicFormats library, thanks to Dom Fober for providing it.
Output files are written in sub-folders of the current folder.

Available targets are:
  'all' (default) : call the 'lily' target.

  'lily'          : converts the set of MusicXML files to lilypond in folder

  'pnglily'       : converts the output of 'lily' target to png in folder pnglily
  'pdflily'       : converts the output of 'lily' target to pdf in folder pdflily
  'midilily'      : converts the output of 'lily' target to pdf in folder midilily

  'clean'         : removes the sub-folders containing the results

Options:
  'XML2LY=/path/to/xml2ly
  'OPTIONS=...
  'LILYPOND=/path/to/lilypond
  When those options are not specified, the tools are taken from the current PATH variable
    ,
  and OPTIONS contains '-q' (quiet mode) for xml2ly.
```

The conversion to the 'lilypond' default can be achieved by:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > make
XML2LY   = xml2ly
OPTIONS  = -q
LILYPOND = /Applications/LilyPond/lilypond-2.23.6/bin/lilypond

xml2ly version is:
Command line version of musicxml2lilypond converter v0.9.53 (February 21, 2022)

lilypond version is:
GNU LilyPond 2.23.6 (running Guile 2.2)

xml2ly -q -o "lilypond/HumdrumScotKeys.ly" "HumdrumScotKeys.xml"
/Applications/LilyPond/lilypond-2.23.6/bin/lilypond --pdf -l NONE -s -o "pdflily/
    HumdrumScotKeys" "lilypond/HumdrumScotKeys.ly"

Contents of lilypond sub-folder:
8 -rw-r--r--  1 jacquesmenu  staff  1852 Feb 25 15:42 lilypond/HumdrumScotKeys.ly

Contents of pdflily sub-folder:
88 -rw-r--r--  1 jacquesmenu  staff  42795 Feb 25 15:42 pdflily/HumdrumScotKeys.pdf
```

The resulting sub-folder contents is:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > ls -salR
total 24
 0 drwxr-xr-x    7 jacquesmenu  staff   224 Feb 25 15:42 .
 0 drwxr-xr-x  108 jacquesmenu  staff  3456 Feb 25 07:46 ..
16 -rw-r--r--@   1 jacquesmenu  staff  6148 Feb 25 12:51 .DS_Store
 8 -rw-r--r--    1 jacquesmenu  staff  2634 Apr 22  2021 HumdrumScotKeys.xml
 0 lrwxr-xr-x    1 jacquesmenu  staff    11 Feb 25 14:12 Makefile -> ../Makefile
 0 drwxr-xr-x    3 jacquesmenu  staff    96 Feb 25 15:42 lilypond
 0 drwxr-xr-x    4 jacquesmenu  staff   128 Feb 25 15:42 pdflily

./lilypond:
total 8
0 drwxr-xr-x  3 jacquesmenu  staff    96 Feb 25 15:42 .
0 drwxr-xr-x  7 jacquesmenu  staff   224 Feb 25 15:42 ..
8 -rw-r--r--  1 jacquesmenu  staff  1852 Feb 25 15:42 HumdrumScotKeys.ly
```

```
16
17 ./pdflily:
18 total 96
19  0 drwxr-xr-x  4 jacquesmenu  staff    128 Feb 25 15:42 .
20  0 drwxr-xr-x  7 jacquesmenu  staff    224 Feb 25 15:42 ..
21  8 -rw-r--r--  1 jacquesmenu  staff    202 Feb 25 15:42 HumdrumScotKeys.midi
22 88 -rw-r--r--  1 jacquesmenu  staff  42795 Feb 25 15:42 HumdrumScotKeys.pdf
```

The sub-folder can be cleaned-up with:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > make clean
2
3 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > ls -sal
4 total 24
5  0 drwxr-xr-x    5 jacquesmenu  staff   160 Feb 25 15:45 .
6  0 drwxr-xr-x  108 jacquesmenu  staff  3456 Feb 25 07:46 ..
7 16 -rw-r--r--@   1 jacquesmenu  staff  6148 Feb 25 12:51 .DS_Store
8  8 -rw-r--r--    1 jacquesmenu  staff  2634 Apr 22  2021 HumdrumScotKeys.xml
9  0 lrwxr-xr-x    1 jacquesmenu  staff    11 Feb 25 14:12 Makefile -> ../Makefile
```

Chapter 7

The MusicFormats repository

The MusicFormats repository contains several versions:

- the defaultmaster version, to be found at https://github.com/jacques-menu/musicformats, is where changes are pushed by the maintainers of MusicFormats. It is the most up to date;
- the `v....` versions are the master versions frozen at some point in time.

This document mentions sample files in various formats. They can be seen online on the `dev` version, whose name and URL never change, at https://github.com/jacques-menu/musicformats/tree/dev/files.

These examples are mentioned in this document as they appear in the MusicFormats repository, in subdirectories of the `files` directory. Currently, there are:

- `musicxmlfiles` for MusicXML files;
- `msdlfiles` for MSDL files.

A typical example is basic/HelloWorld.xml, which stands for the following, with the `dev` cloned locally in the `musicformats-git-dev` directory:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev > ls -sal files/musicxmlfiles/basic/HelloWorld
    .xml
2 8 -rw-r--r--@ 1 jacquesmenu  staff  1266 Apr 22  2021 files/musicxmlfiles/basic/HelloWorld
    .xml
```

Chapter 8

Library components

MusicFormats uses the following terminology for its components:

- a *format* is a description of music scores in textual of binary form, used in the field of music score applications, thus outside of the library;
- a *representation* in an internal data structure describing a music score. As of this writing, the supported representations are:
  - MXSR (MusicXML Score Representation);
  - MSR (Music Score Representation);
  - LSPR (LilyPond Score Representation);
  - BSR (Braille Score Representation).

  There is another, non-musical, representation in MusicFormats: OAH contains a description of the options and help provided by the library and its 'musical' components.
- the formats known to MusicFormats can be seen as external representations of music scores, while representations are internal to the library;
- a *pass* performs a unitary conversion between a format and/or a representation to another such, as a *single step*. This term comes from the compiler writing field: it means that the whole music score description is traversed to produce another description;
- a *converter* is a sequence of two or more passes. Each one converts a representation, either external or internal, into another that is used by the next pass in a pipeline way, at the higher level.
  Such converters are thus said to be *multi-pass*.

  The first one, provided by the `libmusicxml2` library, was `xml2guido`.

  Other converters provided by MusicFormats were added later by this author, currently: `xml2ly` `xml2brl` `xml2xml` and `xml2guido`.

  For example:

```
jacquesmenu@macmini > xml2xml -about
What xml2xml does:

    This multi-pass converter basically performs 6 passes:
        Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
                 and converts it to a MusicXML tree;
        Pass 2a: converts that MusicXML tree into
                 a first Music Score Representation (MSR) skeleton;
        Pass 2b: populates the MSR skeleton from the MusicXML tree
                 to get a full MSR;
        Pass 3:  converts the first MSR into a second MSR, to apply options;
```

```
        Pass 4:  converts the second MSR into a second MusicXML tree;
        Pass 5:  converts the second MusicXML tree to MusicXML code
                 and writes it to standard output.

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

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

- a *generator* is a multi-pass converter that creates the first represention of a score in the sequence *ex-nihilo*, without reading any input file. The ones provided by MusicFormats are `Mikrokosmos3Wandering` and `LilyPondIssue34`:

```
jacquesmenu@macmini > Mikrokosmos3Wandering -musicxml -about
What Mikrokosmos3Wandering does:

    This multi-pass generator creates a textual representation
    of Zoltán Kodály's Mikrokosmos III Wandering score.
    It basically performs 4 passes when generating MusicXML output:

        Pass 1:  generate a first MSR for the Mikrokosmos III Wandering score
        Pass 2:  converts the first MSR a second MSR, to apply options;
        Pass 3:  converts the second MSR into an MusicXML tree;
        Pass 4:  converts the MusicXML tree to MusicXML code
                 and writes it to standard output.

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

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

- MusicFormats also provides various examples and tools to create and manipulate music scores.

At the command line level, only the converters, generators and OAH are available to the user. The other components are used behind the scenes by the latter.

The MusicFormats APIs, on the other hand, give full access to all the components, more about this in Part V.

8.1 Formats

The formats supported by MusicFormats 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. This file is provided in `lilypondstuff/jianpu`;
Braille	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.

8.2 Representations

The representations used by MusicFormats are:

Representation	Description
MSR	Music Score Representation, in terms of part groups, parts, staves, voices, notes, etc. This is the heart of the multi-format converters provided by MusicFormats;
MXSR	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.

8.3 Passes

In the picture, the arrows show the available passes. They are:

Arrow	Pass name	Description
1	mxml2mxsr	reads MusicXML data from a file or from standard input is '-' is supplied as the file name, and creates an MXSR representation containg the same data;
2	mxsr2mxml	converts an MXSR representation into MusicXML data. This is a mere 'print()' operation;
3	mxsr2guido	converts an MXSR representation into Guido text code, and writes it to standard output;
4	mxsr2msr	converts an MXSR representation 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	mxsr2lpsr	converts an MSR representation into an LSPR representation, which contains an MSR component build from the original MSR (pass 5). The BSR contains LilyPond-specific formats such as `\layout`, `\paper`, and `\score` blocks;
6	lpsr2msr	converts an LSPR representation into an MSR representation. There is nothing to do, since the former contains the latter as a component;
7	lpsr2lilypond	converts an LSPR representation into LilyPond text code, and writes it to standard output;
7'	lpsr2lilypond	converts an LSPR representation into LilyPond text code using `lilypond-Jianpu`, and writes it to standard output. This pass is run with `xml2ly -jianpu`;

8	msr2bsr	converts an MSR representation into a BSRrepresentation, which contains an MSR component built from the original MSR. The BSR contains Braille-specific formats such as pages, lines and 6-dot cells. The lines and pages are virtual, i.e. not limited in length. This the pass where skip (invisible) notes are added wherever needed to avoid the LilyPond #34 issue;
9	bsr2msr	converts a BSRrepresentation into an MSR representation. There is nothing to do, since the former contains the latter as a component;
10	bsr2bsr	converts a BSRrepresentation 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	bsr2braille	converts a BSRrepresentation into Braille text, and writes it to standard output;
12	msr2mxsr	converts an MSR representation into an MXSR representation;
13	msr2msr	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 apply options;
14	msdl2msr	converts an MSDL score description into an MSR representation.

8.4 Generators

The ones generators by `libmusicxml2` create an MXSR representation and output it as MusicXML text:

- libmusicxml/samples/RandomMusic.cpp
  generates an MXSR representation containing random music, and writes it as MusicXML to standard output;
- libmusicxml/samples/RandomChords.cpp:
  generates an MXSR representation containing random two-note chords, and writes it as MusicXML to standard output;

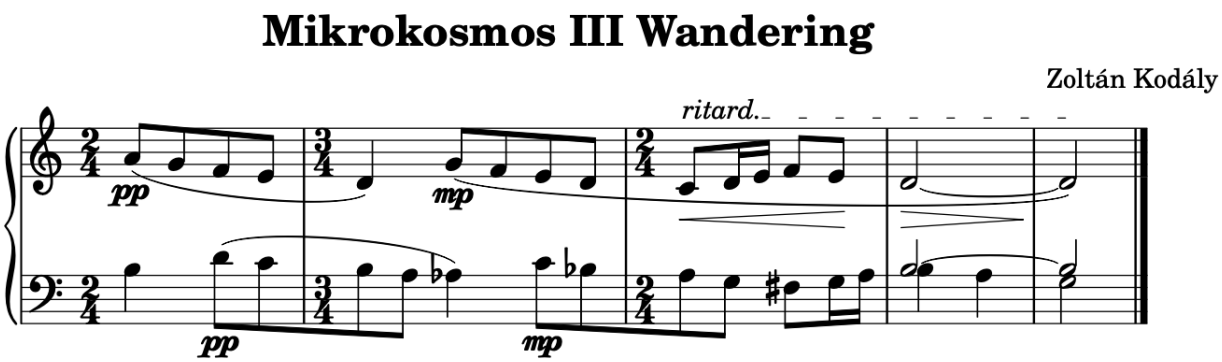
MusicFormats supplies its own generators to demonstrate the use of its APIs: These generators are:

- src/clisamples/MusicAndHarmonies.cpp:
  builds an MXSR representation containing notes and harmonies, and writes it as MusicXML to standard output:

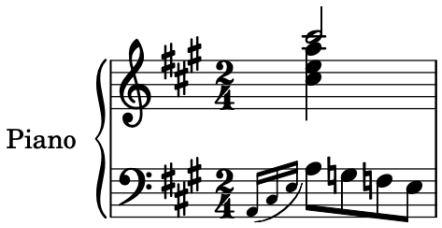
```
jacquesmenu@macmini > MusicAndHarmonies | more
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
                        "http://www.musicxml.org/dtds/partwise.dtd">
<score-partwise>
    <movement-title>Random Music</movement-title>
    <identification>
        <creator type="Composer">Georg Chance</creator>
        <encoding>
            <software>MusicFormats Library's MusicAndHarmonies generator</software>
        </encoding>
    </identification>
    <part-list>
        <score-part id="P1">
            <part-name>Part name</part-name>
            <score-instrument id="I1">
                <instrument-name>Any instr.</instrument-name>
            </score-instrument>
```

```
        </score-part>
    </part-list>
    <part id="P1">
        <measure number="1">
            <attributes>
                <divisions>4</divisions>
                <time>
                    <beats>4</beats>
                    <beat-type>4</beat-type>
                </time>
                <clef>
                    <sign>G</sign>
                    <line>2</line>
                </clef>
            </attributes>
            <harmony>
                <root>
                    <root-step>C</root-step>
                </root>
                <kind text="FOO">major</kind>
                <staff>1</staff>
            </harmony>
            <note>
                <pitch>
                    <step>F</step>
                    <octave>5</octave>
                </pitch>
                <duration>4</duration>
                <type>quarter</type>
            </note>

<!-- ... ... ... -->
```

- src/clisamples/Mikrokosmos3Wandering.cpp:
  creates an MSR graph representing Bartok's Mikrokosmos III Wandering score, and then produces LilyPond, Braille, MusicXML or Guido from it. The LilyPond output gives:



- src/clisamples/LilyPondIssue34.cpp:
  aims at creating an LSPR graph representing the score below, and then produces LilyPond, Braille, MusicXML or Guido from it. Currently, the code is the same as that of `Mikrokosmos3Wandering`, though:



8.5 General use converters

The available MusicXML converters available in MusicFormats are:

Converter	Description
`xml2guido`	supplied by `libmusicxml2`, converts MusicXML data to Guido code, using passes: 1 ⇒ 3
`xml2ly`	performs the 4 passes from MusicXML to LilyPond to translate the former into the latter, using these passes: 1 ⇒ 4 ⇒ 13 ⇒ 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 ⇒ 13 ⇒ 5 ⇒ 7'
`xml2brl`	performs the 5 passes from MusicXML to Braille to translate the former into the latter (draft); 1 ⇒ 4 ⇒ 13 ⇒ 8 ⇒ 10 ⇒ 11
`xml2xml`	converts MusicXML data to MSR and back in 5 passes. This is useful to modify MusicXML 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

The passes used by the converters are shown by their `-about, -a` option. For example:

```
jacquesmenu@macmini > xml2xml -about
What xml2xml does:

    This multi-pass converter basically performs 6 passes:
        Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
                 and converts it to a MusicXML tree;
        Pass 2a: converts that MusicXML tree into
                 a first Music Score Representation (MSR) skeleton;
        Pass 2b: populates the MSR skeleton from the MusicXML tree
                 to get a full MSR;
        Pass 3:  converts the first MSR into a second MSR, to apply options;
        Pass 4:  converts the second MSR into a second MusicXML tree;
        Pass 5:  converts the second MusicXML tree to MusicXML code
                 and writes it to standard output.

    Other passes are performed according to the options, such as
```

```
    displaying views of the internal data or printing a summary of the score.

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

Since the generators may produce various output formats, one should be specified:

```
jacquesmenu@macmini > Mikrokosmos3Wandering -about
What Mikrokosmos3Wandering does:

    This multi-pass generator creates a textual representation
    of Zoltán Kodály's Mikrokosmos III Wandering score.
    It performs various passes depending on the output generated,
    which should be specified a '-lilypond', '-braille', '-musicxml' or '-guido' option.

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

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

Adding option `-braille`, for example we get:

```
jacquesmenu@macmini > Mikrokosmos3Wandering -braille -about
What Mikrokosmos3Wandering does:

    This multi-pass generator creates a textual representation
    of Zoltán Kodály's Mikrokosmos III Wandering score.
    It basically performs 4 passes when generating braille output:

        Pass 1:  generate a first MSR for the Mikrokosmos III Wandering score
        Pass 2:  converts the first MSR a second MSR, to apply options;
        Pass 3:  converts the second MSR into a
                 Braille Score Representation (BSR)
                 containing one Braille page per MusicXML page;
        Pass 4:  converts the BSRinto another BSR
                 with as many Braille pages as needed
                 to fit the line and page lengthes;
        Pass 5:  converts the BSR to Braille text
                 and writes it to standard output.)

    In this preliminary version, pass 3 merely clones the BSR it receives.

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

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

8.6 Specific converters

MusicFormats provides only one compiler in the usual software meaning, namely `msdlconverter`.

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. MusicFormatssupplies `msdl`, a compiler converting MSDL into Guido LilyPond, Brailleor MusicXML to standard output, depending on the '`-generated-code-kind`' option.

Translator	Description

`msdlconverter -lilypond`	performs the 4 passes from MusicXML to LilyPond to translate the former into the latter, using these passes: 1 ⇒ 4 ⇒ 13 ⇒ 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 ⇒ 13 ⇒ 5 ⇒ 7'
`msdlconverter -braille`	performs the 5 passes from MusicXML to Braille to translate the former into the latter (draft); 1 ⇒ 4 ⇒ 13 ⇒ 8 ⇒ 10 ⇒ 11
`msdlconverter -musicxml`	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
`msdlconverter -guido`	converts MusicXML data to Guido code, using passes: 1 ⇒ 4 ⇒ 13 ⇒ 12 ⇒ 3

8.7 MusicFormats services

The MusicFormats library provides *services* to the user. As of this writing, they are:

- generators;
- converters.

Other services may be provided in the future, such as music score analyzers.

This is why this documentation uses the term *service* for the current generators and converters.

8.8 Other 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.

Part III

Shell basics

Chapter 9

Shell basics

Since this document is about using MusicFormats from the command line by musicians, let's start by a short presentation of shell usage. This chapter can be skipped of course by shell-savvy users.

A shell is an application that reads commands and executes them. In the early ages of physical *terminal*s, they were typically typed on a keyboard. With GUI nowadays, they can be typed in a so-called terminal window. A

The syntax of shell commands is meant to be simple, without complex structuring features such as those found in programming languages.

A number of shell have been used over the years. Most of the ones used at the time of this writing belong to the `sh` family, among then Bash and Z shell (`zsh`. The commands we use in this document can be run on any shell in this family.

9.1 Basic shell builtins

Many builtins have very short names for ease of interactive use. Many vowels were left out to minimize typing. For example, there are:

- `pwd` to show the current working directory;
- `cd` to change directory;
- `echo` to produce output in the terminal window.

A so-called *prompt* is displayed by a shell when is it ready to read a command and execute it. This document uses two kinds of prompts:

- one contains only the user name and machine name, such as:

```
jacquesmenu@macmini: ~ >
```

- the other one displays the current working directory: it is used when the latter has to be set at a specific value for the command:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles >
```

When a shell is launched, a directory is chosen as the current *working directory*, usually the user's *home directory*.

```
jacquesmenu@macmini: ~ > pwd
/Users/jacquesmenu

jacquesmenu@macmini: ~ > cd musicformats-git-dev

jacquesmenu@macmini: ~/musicformats-git-dev > pwd
/Users/jacquesmenu/musicformats-git-dev
jacquesmenu@macmini: ~/musicformats-git-dev >
```

9.2 Commands

A command name is either provided by the shell itself, a so-called *builtin*, or the name of a piece of software that can be executed.

In this example, the command name is xml2lyy:

```
jacquesmenu@macmini > xml2lyy +sdf 45
-bash: xml2lyy: command not found
```

The shell can be queried about a command name:

```
jacquesmenu@macmini > type cd
cd is a shell builtin

jacquesmenu@macmini > type xml2lyy
-bash: type: xml2lyy: not found

jacquesmenu@macmini > type xml2ly
xml2ly is hashed (/Users/jacquesmenu/musicformats-git-dev/build/bin/xml2ly)
```

9.3 Paths

The files on a computer are organized as file-systems. A path is a way to access a file on a file system:

- on Unixlike system, there is a single tree of so-called *directories*, the root being named `/`. A sub-directory is preceded by `/` in the paths;
- on Windows™ systems, there is a set of trees, their roots being the physical or virtual drives, such as `C:`. A \ is used to indicate a sub-directory.

This document uses Unixlike pathes.

9.4 Quoting, variables and aliases

Shell commands are submitted as a sequence of words separated by spaces. If a word, such a file name, contains *spaces*, it has to be surrounded by quotes or double quotes in order to be seen by the shell as a single word:

```
jacquesmenu@macmini > xml2ly Nice file.xml
Several input file names supplied, only one can be used
The arguments vector contains 2 elements:
   0: "Nice"
   1: "file.xml"

jacquesmenu@macmini > xml2ly 'Nice file.xml'
can't open file Nice file.xml
### Conversion from MusicXML to LilyPond failed ###

jacquesmenu@macmini > xml2ly "Nice file.xml"
can't open file Nice file.xml
### Conversion from MusicXML to LilyPond failed ###
```

Note that if a quote or double quote is part of word, the word should be inclosed by the other such:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -find "tuplet's"
0 occurrence of string "tuplet's" has been found

jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles >
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -find 'tuplet"s'
0 occurrence of string "tuplet"s" has been found
```

A shell *variable* is a name for a piece of text, called its *value*, that can be used instead of that text in commands. The value of the variable can be seen in the terminal with the `echo` command:

```
jacquesmenu@macmini: ~/musicformats-git-dev > DOC_DIR=documentation

jacquesmenu@macmini: ~/musicformats-git-dev > echo $DOC_DIR
documentation
```

Variables can be used surrounded by curly brackets, too:

```
jacquesmenu@macmini: ~/musicformats-git-dev/documentation > echo ${DOC_DIR}
documentation
```

This notation provides further possibilities such as string replacement, which are out of the scope of this document.

Using variables is interesting when there are several uses of its value: changing the value at one place causing the new value to be used at every such use:

```
jacquesmenu@macmini: ~/musicformats-git-dev > ls $DOC_DIR
CommonLaTeXFiles    MusicFormatsCLIUserGuide  presentation
IntroductionToMusicXML    MusicFormatsMaintainanceGuide
MusicFormatsAPIUserGuide  graphics

jacquesmenu@macmini: ~/musicformats-git-dev > cd $DOC_DIR

jacquesmenu@macmini: ~/musicformats-git-dev/documentation > pwd
/Users/jacquesmenu/musicformats-git-dev/documentation
```

The difference between quotes and double quotes is how variables are handled:

- the characters between quotes are used literally;

- variables occurring between double quotes are replaced by their value.

```
jacquesmenu@macmini: ~/musicformats-git-dev > DOC_DIR=documentation

jacquesmenu@macmini: ~/musicformats-git-dev > cd '$DOC_DIR'
-bash: cd: $DOC_DIR: No such file or directory

jacquesmenu@macmini: ~/musicformats-git-dev > cd "$DOC_DIR"

jacquesmenu@macmini: ~/musicformats-git-dev/documentation > pwd
/Users/jacquesmenu/musicformats-git-dev/documentation
```

Here is an example combining quotes and double quotes:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > DOC_DIR=documentation
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > echo "DOC_DIR's value is
    : ${DOC_DIR}"
DOC_DIR's value is: documentation
```

9.5 Functions

The shells allow the creation of *functions*, that contain several commands under a single name. An example is function `checkVersions ()`, which displays the versions of the main MusicFormats services:

```
function checkVersions ()
{
#  set -x

  xml2ly -v
  xml2brl -v
  xml2xml -v
  xml2gmn -v

  Mikrokosmos3Wandering -v

  msdlconverter -v
#  set +x
}
```

9.6 MusicFormatsBashDefinitions.bash

`MusicFormatsBashDefinitions.bash` contains a set of variables, aliases and function definitions used by this author. One of them is function `checkVersions ()` above.

Feel free to use them, adapt them or ignore them depending on your taste.

Some settings we use in this document are:

```
jacquesmenu@macmini > type ll
ll is a function
ll ()
{
    ls -salGTF $*
}
```

The options to `ls` may vary depending the on the operating system.

9.7 Scripts

A shell script is a text file that can be executed using its name as a command. The first line tells which shell should be used to execute the commands in the remainder of the file, `sh` by default.

This author uses scripts as the one below as handy interactive short-cuts. It groups commands to copy a MusicXML file exported after scanning a PDF file to another file, convert the latter to LilyPond with `xml2ly` and open the result with fresco to produce the PDF score:

```
jacquesmenu@macmini > cat doBethena_SaxTenor.bash
#/bin/bash

cp -p Bethena_SaxTenor_original.xml Bethena_SaxTenor.xml

xml2ly -include  Bethena_SaxTenor_OptionsAndArguments.txt

open Bethena_SaxTenor.ly
```

After creating the script file, make it executable:

```
jacquesmenu@macmini > chmod +x doBethena_SaxTenor.bash

jacquesmenu@macmini > ls -sal doBethena_SaxTenor.bash
8 -rwxr-xr-x@ 1 jacquesmenu  staff  154 Feb 20 07:46 doBethena_SaxTenor.bash
```

The script can then be executed with:

```
jacquesmenu@macmini > ./doBethena_SaxTenor.bash
```

Part IV

Installing MusicFormats

Chapter 10

MusicFormats installation modes

There is no GUI installer available yet, so users have to install the library at a lower level, sorry for that....

How to install MusicFormats depends on the operating system. Linux users often build the software they use themselves, while those of Windows™ and Mac OS™ are accustomed to install in much simpler ways.

Depending on the needs, users may wish to install the *whole* MusicFormats with source code and examples, or to use a *distribution*, that contains only the *libraries* if relevant, the command line executables and the documentation PDF files.

The following chapters show the details.

Chapter 11

Using a distribution

The MusicFormats repository is hosted by GitHub and use so-called *actions* to build the library on Mac OS™, Ubuntu and Windows™. The resulting files are then uploaded to the repository, where they are available to create the distributions for these there operating systems.

These distributions are in the form of Zip files. They are are available from https://github.com/jacques-menu/musicformats/tree/master/distrib, as well as the documentation PDF files:

```
jacquesmenu@macmini: ~/musicformats_local_clone/distrib > ls -sal
total 150088
     0 drwxr-xr-x   8 jacquesmenu  staff       256 Feb 16 07:47 .
     0 drwxr-xr-x  22 jacquesmenu  staff       704 Feb 16 07:47 ..
  1672 -rw-r--r--   1 jacquesmenu  staff    854294 Feb 16 07:47 IntroductionToMusicXML.pdf
  1976 -rw-r--r--   1 jacquesmenu  staff   1008702 Feb 16 07:47 MusicFormatsCLIUserGuide.
    pdf
108168 -rw-r--r--   1 jacquesmenu  staff  55378423 Feb 16 07:47 MusicFormatsForMacOS.zip
 34080 -rw-r--r--   1 jacquesmenu  staff  17445663 Feb 16 07:47 MusicFormatsForUbuntu.zip
  4184 -rw-r--r--   1 jacquesmenu  staff   2139537 Feb 16 07:47 MusicFormatsForWindows.zip
     8 -rw-r--r--   1 jacquesmenu  staff         6 Feb 16 07:47 MusicFormatsVersionNumber.
    txt
```

These distribution Zip archive are the ones for the current, most recent version of MusicFormats. To use earlier versions, the library has to be cloned locally, see **??**.

11.1 MacOS™ distribution

Mac OS™ software is usually distributed as DMG files. Due to file size limitations on GitHub, the Mac OS™ distribution has to be compacted. This is done with Zip, and placing that in a DMG archive would not add any value. Only the Zip archive is thus provided.

After downloading and uncompressing `MusicFormatsForMacOS.zip`, we get:

```
jacquesmenu@macmini: ~/Downloads/MusicFormatsForMacOS > ls -sal *
8 -rw-r--r--@ 1 jacquesmenu  staff  6 Feb 14 14:20 MusicFormatsVersionNumber.txt

bin:
total 661992
     0 drwxr-xr-x@ 25 jacquesmenu  staff       800 Feb 14 14:20 .
     0 drwxr-xr-x@  4 jacquesmenu  staff       128 Feb 15 17:23 ..
74864 -rwxr-xr-x@  1 jacquesmenu  staff  38326752 Feb 14 14:20 LilyPondIssue34
74864 -rwxr-xr-x@  1 jacquesmenu  staff  38329824 Feb 14 14:20 Mikrokosmos3Wandering
 8432 -rwxr-xr-x@  1 jacquesmenu  staff   4314896 Feb 14 14:20 MusicAndHarmonies
 8432 -rwxr-xr-x@  1 jacquesmenu  staff   4314880 Feb 14 14:20 RandomChords
 8432 -rwxr-xr-x@  1 jacquesmenu  staff   4314880 Feb 14 14:20 RandomMusic
 8624 -rwxr-xr-x@  1 jacquesmenu  staff   4414944 Feb 14 14:20 countnotes
```

```
16528 -rwxr-xr-x@  1 jacquesmenu  staff    8459424 Feb 14 14:20 displayMusicformatsHistory
16528 -rwxr-xr-x@  1 jacquesmenu  staff    8459424 Feb 14 14:20 displayMusicformatsVersion
79200 -rwxr-xr-x@  1 jacquesmenu  staff   40546384 Feb 14 14:20 msdlconverter
12480 -rwxr-xr-x@  1 jacquesmenu  staff    6387232 Feb 14 14:20 partsummary
 8848 -rwxr-xr-x@  1 jacquesmenu  staff    4528736 Feb 14 14:20 readunrolled
64000 -rwxr-xr-x@  1 jacquesmenu  staff   32764496 Feb 14 14:20 xml2brl
66872 -rwxr-xr-x@  1 jacquesmenu  staff   34236240 Feb 14 14:20 xml2gmn
17160 -rwxr-xr-x@  1 jacquesmenu  staff    8781984 Feb 14 14:20 xml2guido
67552 -rwxr-xr-x@  1 jacquesmenu  staff   34583840 Feb 14 14:20 xml2ly
12392 -rwxr-xr-x@  1 jacquesmenu  staff    6342528 Feb 14 14:20 xml2midi
59720 -rwxr-xr-x@  1 jacquesmenu  staff   30574528 Feb 14 14:20 xml2xml
 9104 -rwxr-xr-x@  1 jacquesmenu  staff    4657200 Feb 14 14:20 xmlclone
 9256 -rwxr-xr-x@  1 jacquesmenu  staff    4735296 Feb 14 14:20 xmlfactory
 8800 -rwxr-xr-x@  1 jacquesmenu  staff    4504976 Feb 14 14:20 xmliter
 8680 -rwxr-xr-x@  1 jacquesmenu  staff    4442496 Feb 14 14:20 xmlread
11976 -rwxr-xr-x@  1 jacquesmenu  staff    6129744 Feb 14 14:20 xmltranspose
 9248 -rwxr-xr-x@  1 jacquesmenu  staff    4734368 Feb 14 14:20 xmlversion
```

Mac OS™ executables are self-sufficient and can be placed anywhere on a disk except the trash. Usually, there are placed in the `/Applications` directory.

11.1.1 Security issue in recent MacOS™ versions

Mac OS™ gets more and more stringent over time regarding security. The operating system part in charge of this is named Gatekeeper.

When installing MusicFormats from the repository on versions up to 10 (High Sierra), the executables in `bin` are usable alright.

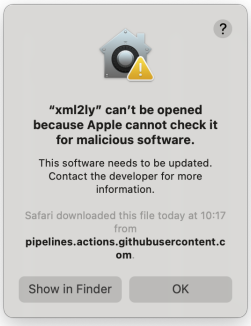
From version 11 (Catalina) on, though, the executables you get are not executable actually, because their developer is unknown to the operating system, and actions have to be taken for them to be usable.

The screenshot below has been made with Mac OS™ Monterey 12.0.1 with english as the user interface language. The texts vary of course depending on the language used.

When launching one of these executables for the first time, such as:

```
jacquesmenu@macmini: ~/Downloads/MusicFormatsForMacOS/bin > ./xml2ly
```

we get a alert telling that it cannot be opened, because the developper is not known to the operating system:



Clicking in either buttons in this dialog kill the process:

```
Killed: 9
```

The trouble is that these executables are in *quarantine* by default. To make them usable, they have to quit quarantine, which is done by removing one of their attributes:

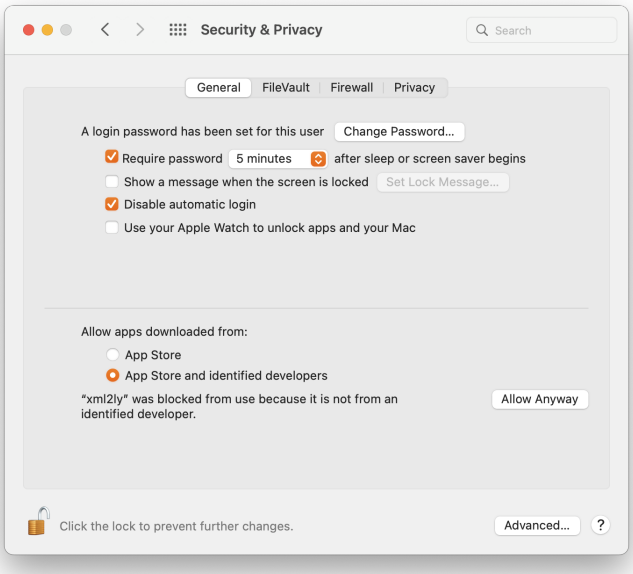
```
jacquesmenu@macmini: ~/Downloads/MusicFormatsForMacOS/bin > xattr -d com.apple.quarantine
    *
```

From then on, the MusicFormats executables can be used seamlessly on the given machine.

Having to perform the preceding task for each executable is the price to pay for security. And it has to be perfomed again when installing new versions...

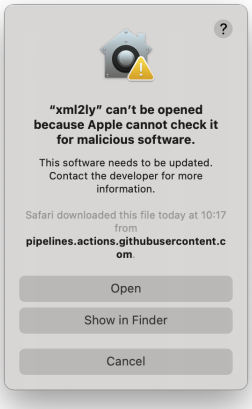
The above can be done in the GUI file by file too. Right after you got the message above:

- open *System Preferences*, choose the *Security & Privacy* tab, and there click on the *General button*;
- click on the lock at the bottom left of the dialog to make changes:



- click on the *Allow Anyway* button.

Re-execute the executable from the command line. This pops-up a dialog to confirm you actually want to use this software:



Click on the *Open* button to register the executable in Gatekeeper and go ahead.

11.2 Ubuntu distribution

After downloading, we get:

```
jacquesmenu@macmini: ~/Downloads/MusicFormatsForUbuntu > ls -sal *
8 -rw-r--r--@ 1 jacquesmenu  staff  6 Feb 14 14:33 MusicFormatsVersionNumber.txt

bin:
total 2296
  0 drwxr-xr-x@ 25 jacquesmenu  staff     800 Feb 14 18:22 .
  0 drwxr-xr-x@  6 jacquesmenu  staff     192 Feb 16 08:45 ..
 96 -rw-r--r--@  1 jacquesmenu  staff   49000 Feb 14 14:33 LilyPondIssue34
 96 -rw-r--r--@  1 jacquesmenu  staff   49040 Feb 14 14:33 Mikrokosmos3Wandering
 96 -rw-r--r--@  1 jacquesmenu  staff   47224 Feb 14 14:33 MusicAndHarmonies
 96 -rw-r--r--@  1 jacquesmenu  staff   47216 Feb 14 14:33 RandomChords
 96 -rw-r--r--@  1 jacquesmenu  staff   47216 Feb 14 14:33 RandomMusic
 72 -rw-r--r--@  1 jacquesmenu  staff   33800 Feb 14 14:33 countnotes
 40 -rw-r--r--@  1 jacquesmenu  staff   17648 Feb 14 14:33 displayMusicformatsHistory
 40 -rw-r--r--@  1 jacquesmenu  staff   17648 Feb 14 14:33 displayMusicformatsVersion
104 -rw-r--r--@  1 jacquesmenu  staff   50616 Feb 14 14:33 msdlconverter
544 -rw-r--r--@  1 jacquesmenu  staff  275976 Feb 14 14:33 partsummary
 88 -rw-r--r--@  1 jacquesmenu  staff   43720 Feb 14 14:33 readunrolled
 80 -rw-r--r--@  1 jacquesmenu  staff   39200 Feb 14 14:33 xml2brl
 88 -rw-r--r--@  1 jacquesmenu  staff   43336 Feb 14 14:33 xml2gmn
 48 -rw-r--r--@  1 jacquesmenu  staff   23112 Feb 14 14:33 xml2guido
 80 -rw-r--r--@  1 jacquesmenu  staff   39056 Feb 14 14:33 xml2ly
 88 -rw-r--r--@  1 jacquesmenu  staff   42880 Feb 14 14:33 xml2midi
 88 -rw-r--r--@  1 jacquesmenu  staff   43344 Feb 14 14:33 xml2xml
 88 -rw-r--r--@  1 jacquesmenu  staff   43368 Feb 14 14:33 xmlclone
 48 -rw-r--r--@  1 jacquesmenu  staff   22616 Feb 14 14:33 xmlfactory
168 -rw-r--r--@  1 jacquesmenu  staff   83488 Feb 14 14:33 xmliter
 56 -rw-r--r--@  1 jacquesmenu  staff   28424 Feb 14 14:33 xmlread
 56 -rw-r--r--@  1 jacquesmenu  staff   28656 Feb 14 14:33 xmltranspose
 40 -rw-r--r--@  1 jacquesmenu  staff   17360 Feb 14 14:33 xmlversion

lib:
total 157792
     0 drwxr-xr-x@ 4 jacquesmenu  staff       128 Feb 14 18:22 .
     0 drwxr-xr-x@ 6 jacquesmenu  staff       192 Feb 16 08:45 ..
113224 -rw-r--r--@ 1 jacquesmenu  staff  57968176 Feb 14 14:33 libmusicxml2.a
 44568 -rw-r--r--@ 1 jacquesmenu  staff  22818696 Feb 14 14:33 libmusicxml2.so
```

Move the `MusicFormatsForUbuntu` directory to a suitable place and set your `PATH` and `LIBRARY_PATH` environment variables accordingly.

11.3 Windows™ distribution

After downloading, we get:

```
jacquesmenu@macmini: ~/Downloads/MusicFormatsForWindows > ls -sal *
8 -rw-r--r--@ 1 jacquesmenu  staff  6 Feb 14 14:53 MusicFormatsVersionNumber.txt

bin:
total 1232
  0 drwxr-xr-x@ 25 jacquesmenu  staff    800 Feb 14 18:22 .
  0 drwxr-xr-x@  6 jacquesmenu  staff    192 Feb 16 08:49 ..
 80 -rw-r--r--@  1 jacquesmenu  staff  38400 Feb 14 14:53 LilyPondIssue34.exe
 80 -rw-r--r--@  1 jacquesmenu  staff  38400 Feb 14 14:53 Mikrokosmos3Wandering.exe
 56 -rw-r--r--@  1 jacquesmenu  staff  26112 Feb 14 14:53 MusicAndHarmonies.exe
 56 -rw-r--r--@  1 jacquesmenu  staff  25088 Feb 14 14:53 RandomChords.exe
 56 -rw-r--r--@  1 jacquesmenu  staff  25088 Feb 14 14:53 RandomMusic.exe
 32 -rw-r--r--@  1 jacquesmenu  staff  14848 Feb 14 14:53 countnotes.exe
 24 -rw-r--r--@  1 jacquesmenu  staff  10752 Feb 14 14:53 displayMusicformatsHistory.exe
```

```
 24 -rw-r--r--@  1 jacquesmenu  staff   10752 Feb 14 14:53 displayMusicformatsVersion.exe
 80 -rw-r--r--@  1 jacquesmenu  staff   39936 Feb 14 14:53 msdlconverter.exe
112 -rw-r--r--@  1 jacquesmenu  staff   56832 Feb 14 14:53 partsummary.exe
 40 -rw-r--r--@  1 jacquesmenu  staff   18432 Feb 14 14:53 readunrolled.exe
 64 -rw-r--r--@  1 jacquesmenu  staff   32768 Feb 14 14:53 xml2brl.exe
 72 -rw-r--r--@  1 jacquesmenu  staff   33280 Feb 14 14:53 xml2gmn.exe
 64 -rw-r--r--@  1 jacquesmenu  staff   29184 Feb 14 14:53 xml2guido.exe
 64 -rw-r--r--@  1 jacquesmenu  staff   32768 Feb 14 14:53 xml2ly.exe
 40 -rw-r--r--@  1 jacquesmenu  staff   17920 Feb 14 14:53 xml2midi.exe
 72 -rw-r--r--@  1 jacquesmenu  staff   33280 Feb 14 14:53 xml2xml.exe
 32 -rw-r--r--@  1 jacquesmenu  staff   14848 Feb 14 14:53 xmlclone.exe
 32 -rw-r--r--@  1 jacquesmenu  staff   15360 Feb 14 14:53 xmlfactory.exe
 40 -rw-r--r--@  1 jacquesmenu  staff   19456 Feb 14 14:53 xmliter.exe
 56 -rw-r--r--@  1 jacquesmenu  staff   27136 Feb 14 14:53 xmlread.exe
 32 -rw-r--r--@  1 jacquesmenu  staff   14848 Feb 14 14:53 xmltranspose.exe
 24 -rw-r--r--@  1 jacquesmenu  staff   12288 Feb 14 14:53 xmlversion.exe

lib:
total 37368
    0 drwxr-xr-x@ 4 jacquesmenu  staff        128 Feb 14 18:22 .
    0 drwxr-xr-x@ 6 jacquesmenu  staff        192 Feb 16 08:49 ..
14696 -rw-r--r--@ 1 jacquesmenu  staff    7521356 Feb 14 14:53 musicxml2.exp
22672 -rw-r--r--@ 1 jacquesmenu  staff   11604836 Feb 14 14:53 musicxml2.lib
```

Move the `MusicFormatsForUbuntu` directory to a suitable place such as `C:\ Program Files` and set your `PATH` environment variable accordingly.

Chapter 12

Full installation

12.1 Cloning the repository

The library should be cloned locally, on the user's machine, with the command below. This creates a local copy (a *clone* in git's terminology) of the repository's contents, named here `musicformats_local_clone`:

```
jacquesmenu@macmini: ~ > git clone https://github.com/jacques-menu/musicformats.git
    musicformats_local_clone
Cloning into 'musicformats_local_clone'...
remote: Enumerating objects: 20619, done.
remote: Counting objects: 100% (15175/15175), done.
remote: Compressing objects: 100% (7546/7546), done.
remote: Total 20619 (delta 13189), reused 9420 (delta 7560), pack-reused 5444
Receiving objects: 100% (20619/20619), 107.32 MiB | 11.14 MiB/s, done.
Resolving deltas: 100% (15569/15569), done.
```

This creates a local copy idefault, master branch. Alternatively, a previous MusicFormats version can be cloned with something like:

```
jacquesmenu@macmini: ~ > VERSION_NUMBER=v0.0.60

jacquesmenu@macmini: ~ > git clone -b ${VERSION_NUMBER} https://github.com/jacques-menu/
    musicformats musicformats_local_clone-${VERSION_NUMBER}
```

The local clone contains:

```
jacquesmenu@macmini: ~ > cd musicformats_local_clone
jacquesmenu@macmini: ~/musicformats_local_clone > ls -sal
total 96
 0 drwxr-xr-x  22 jacquesmenu  staff    704 Feb  2 17:26 .
 0 drwxr-xr-x+ 80 jacquesmenu  staff   2560 Feb  2 17:26 ..
 0 drwxr-xr-x  12 jacquesmenu  staff    384 Feb  2 17:26 .git
 0 drwxr-xr-x   3 jacquesmenu  staff     96 Feb  2 17:26 .github
 8 -rwxr-xr-x   1 jacquesmenu  staff   1050 Feb  2 17:26 Build_libmusicformats.bash
 0 drwxr-xr-x   3 jacquesmenu  staff     96 Feb  2 17:26 KEEP
40 -rw-r--r--   1 jacquesmenu  staff  16725 Feb  2 17:26 LICENSE
 8 -rwxr-xr-x   1 jacquesmenu  staff   1055 Feb  2 17:26 README.md
 0 drwxr-xr-x   9 jacquesmenu  staff    288 Feb  2 17:26 build
 0 drwxr-xr-x  10 jacquesmenu  staff    320 Feb  2 17:26 docs
 0 drwxr-xr-x   9 jacquesmenu  staff    288 Feb  2 17:26 documentation
 0 drwxr-xr-x   6 jacquesmenu  staff    192 Feb  2 17:26 files
 0 drwxr-xr-x   5 jacquesmenu  staff    160 Feb  2 17:26 javascript
 0 drwxr-xr-x  21 jacquesmenu  staff    672 Feb  2 17:26 libmusicxml
 0 drwxr-xr-x  10 jacquesmenu  staff    320 Feb  2 17:26 midisharelight
40 -rw-r--r--   1 jacquesmenu  staff  18502 Feb  2 17:26 musicFormatsBashDefinitions.bash
 0 drwxr-xr-x   6 jacquesmenu  staff    192 Feb  2 17:26 packages
 0 drwxr-xr-x   8 jacquesmenu  staff    256 Feb  2 17:26 schemas
```

```
 0 drwxr-xr-x  12 jacquesmenu  staff    384 Feb  2 17:26 src
 0 drwxr-xr-x   7 jacquesmenu  staff    224 Feb  2 17:26 validation
 0 drwxr-xr-x  11 jacquesmenu  staff    352 Feb  2 17:26 web
 0 drwxr-xr-x   4 jacquesmenu  staff    128 Feb  2 17:26 win32
```

12.2 make and cmake definitions

make is used to build the library, while cmake implements the portability of MusicFormats to multiple operating systems and environments. Thanks to Dominique Fober for providing this setup libmusicxml2 in the first place. The respective settings are in build/Makefile and build/CMakeLists.txt.

The make file as a number of possibilities:

```
jacquesmenu@macmini: ~/musicformats_local_clone/build > make help
MusicFormats makefile - Targets are :
   all (default): build the MusicFormats library for the current platform,
                  build the MusicFormats tools,

Platform targets to build the MusicFormats library are:
   macos      build the library for macos
   windows    build 32 and 64 bits library for windows
   linux      build the library for linux
   android    build a static library for Android
   ios        build a static library for iOS
   msys       build on Windows using MSys
   js         build a javascript library
the platform targets is automatically evaluated by the default target.

Misc:
   cmake      re-generates the cmake project
   format     source code formatting using clang-format
   install    install library, tools and headers
   localinstall    install the tools to ~/bin
   package    create the musicformats-v0.9.60 package

Options:
   CMAKEOPT  cmake options passed to cmake by the 'cmake' target
   GENERATOR the cmake generator. Currently '-G Xcode'
   PDIR      the generation folder. Currently 'libdir'
   PREFIX    the install location prefix. Currently /usr/local'

CMake options:
   FMWK      [MacOS only] Generates a framework on MacOS. Default is on
   GDB     Activates ggdb3 option. Default is off
   LILY      Include lilypond part. Default is on
NOTE:  CMake options can be passed using CMAKEOPT, e.g.
       'make cmake CMAKEOPT=-DLILY=off'
```

12.3 Building the library on Mac OS™ and Linux-like systems

Mac OS™ and Linux have the same kind of tools behind the scenes for software development.

In order to build MusicFormats from source on your machine, you need:

- a C++11 compiler. Use Xcode on Mac OS™ ang GNU compilers on Unix-like machines;
- the cmake tool. It is available ready to install on Mac OS™ via MacPorts (https://www.macports.org).

The supported operating systems to build the library and run the command line tools are Linux, Windows and MacOS. Other systems may be fine but have not been tested.

MusicFormats requires C++11 at least. More recent versions are fine too.

Once in the local repository clone, just execute, here on Mac OS™:

```
jacquesmenu@macmini: ~/musicformats_local_clone > cd build
jacquesmenu@macmini: ~/musicformats_local_clone/build > make

... ... ...

** BUILD SUCCEEDED **

cd lib &&  [ -d musicformats.framework ] && tar czf musicformats.tgz musicformats.
    framework || echo "no framework"
no framework
```

The resulting executables are in `build/bin`:

```
jacquesmenu@macmini: ~/musicformats_local_clone/build > ls -sal bin
total 661992
    0 drwxr-xr-x@ 25 jacquesmenu  staff       800 Feb 16 09:17 .
    0 drwxr-xr-x  10 jacquesmenu  staff       320 Feb 16 09:15 ..
74864 -rwxr-xr-x   1 jacquesmenu  staff  38327184 Feb 16 09:17 LilyPondIssue34
74864 -rwxr-xr-x   1 jacquesmenu  staff  38330272 Feb 16 09:17 Mikrokosmos3Wandering
 8432 -rwxr-xr-x   1 jacquesmenu  staff   4314896 Feb 16 09:17 MusicAndHarmonies
 8432 -rwxr-xr-x   1 jacquesmenu  staff   4314880 Feb 16 09:17 RandomChords
 8432 -rwxr-xr-x   1 jacquesmenu  staff   4314880 Feb 16 09:17 RandomMusic
 8624 -rwxr-xr-x   1 jacquesmenu  staff   4414944 Feb 16 09:17 countnotes
16528 -rwxr-xr-x   1 jacquesmenu  staff   8459488 Feb 16 09:17 displayMusicformatsHistory
16528 -rwxr-xr-x   1 jacquesmenu  staff   8459488 Feb 16 09:17 displayMusicformatsVersion
79200 -rwxr-xr-x   1 jacquesmenu  staff  40546848 Feb 16 09:17 msdlconverter
12480 -rwxr-xr-x   1 jacquesmenu  staff   6387248 Feb 16 09:17 partsummary
 8848 -rwxr-xr-x   1 jacquesmenu  staff   4528752 Feb 16 09:17 readunrolled
64000 -rwxr-xr-x   1 jacquesmenu  staff  32764864 Feb 16 09:17 xml2brl
66872 -rwxr-xr-x   1 jacquesmenu  staff  34236560 Feb 16 09:17 xml2gmn
17160 -rwxr-xr-x   1 jacquesmenu  staff   8782048 Feb 16 09:17 xml2guido
67552 -rwxr-xr-x   1 jacquesmenu  staff  34584160 Feb 16 09:17 xml2ly
12392 -rwxr-xr-x   1 jacquesmenu  staff   6342560 Feb 16 09:17 xml2midi
59720 -rwxr-xr-x   1 jacquesmenu  staff  30574816 Feb 16 09:17 xml2xml
 9104 -rwxr-xr-x   1 jacquesmenu  staff   4657232 Feb 16 09:17 xmlclone
 9256 -rwxr-xr-x   1 jacquesmenu  staff   4735296 Feb 16 09:17 xmlfactory
 8800 -rwxr-xr-x   1 jacquesmenu  staff   4505008 Feb 16 09:17 xmliter
 8680 -rwxr-xr-x   1 jacquesmenu  staff   4442528 Feb 16 09:17 xmlread
11976 -rwxr-xr-x   1 jacquesmenu  staff   6129760 Feb 16 09:17 xmltranspose
 9248 -rwxr-xr-x   1 jacquesmenu  staff   4734384 Feb 16 09:17 xmlversion
```

The resulting librairies are in `build/bin`, here on MacOS:

```
jacquesmenu@macmini: ~/musicformats_local_clone/build > ls -sal lib
total 1283720
      0 drwxr-xr-x@  6 jacquesmenu  staff        192 Feb 16 09:17 .
      0 drwxr-xr-x  10 jacquesmenu  staff        320 Feb 16 09:15 ..
 107368 -rwxr-xr-x   1 jacquesmenu  staff   54970432 Feb 16 09:17 libmusicformats.0.9.60.
    dylib
      0 lrwxr-xr-x   1 jacquesmenu  staff         28 Feb 16 09:17 libmusicformats.0.dylib
    -> libmusicformats.0.9.60.dylib
1176352 -rw-r--r--   1 jacquesmenu  staff  592564120 Feb 16 09:16 libmusicformats.a
      0 lrwxr-xr-x   1 jacquesmenu  staff         23 Feb 16 09:17 libmusicformats.dylib ->
     libmusicformats.0.dylib
```

12.4 Installing the library on Mac OS™ and Linux-like systems

After building, MusicFormats can be installed either in the user's home directory or globally on the machine.

12.4.1 User specific installation on Mac OS™

This is done with `make localinstall`:

```
jacquesmenu@macmini: ~/musicformats_local_clone/build > make localinstall
cd bin && cp xml2midi xmlread xml2guido xml2ly xmlversion xml2brl /Users/jacquesmenu/bin
```

Make sure this `bin` directory is in your shell `PATH`, and there you are.

12.4.2 Global installation on Mac OS™

This installation, done with `make install`, requires administration privileges:

```
jacquesmenu@macmini: ~/musicformats_local_clone/build > sudo make install
```

12.5 Building the library on Windows™

*** Please contribute to this section, this author does not have any access to a Windows™ machine. ***

In order to build MusicFormats from source on your machine, you need:

- a C++11 compiler. Use Xcode on Mac OS™ ang GNU compilers on Unix-like machines;
- the `cmake` tool. It is available ready to install on Mac OS™ via MacPorts (https://www.macports.org).

The supported operating systems to build the library and run the command line tools are Linux, Windows and MacOS. Other systems may be fine but have not been used for tests.

MusicFormats requires C++11 at least. More recent versions are fine too.

Once in the local repository clone, just execute:

```
cd build

cmake --build <buildir> --target install
```

Part V

Options and help (OAH)

Chapter 13

Options and help design principles

MusicFormats having many services with many options makes options and help handling a challenge. This is why MusicFormats provides OAH (Options And Help), a full-fledged object-oriented options and help management infrastructure.

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? why not!"

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 converters then aggregate the OAH groups of the passes they are composed of to offer their options and help to the user.

MusicFormats 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*.

The `-query` option used through-out this document will be presented in detail at section 15.2, [Querying about options by name], page 56.

The term *command line* means that the user launches the MusicFormats services in a terminal window, using a so-called *shell*. A shell writes a so-called *prompt* in the window, indicating that is waits for user input at the keyboard, and performs a loop:

- it *reads* a line from the keyboard, made of a command name, options and arguments;
- the command is analyzed to check that it is *well-formed*;
- the command is *executed* if it is well-formed;
- the shell displays the prompt again and waits for the next user input.

When a terminal window is created, a shell is launched automatically, waiting for user in put in that window.

Various shell families have been created over time. The most widely used today is Bash (`https://www.gnu.org/software/bash/`). No worry though, the information presented in this section applies to all of them.

Chapter 14

Options use

THe OAH options are very easy to use. They are inspired by GNU options, with more power and flexibility:

- the options can be supplied in the command line as usual;
- they can also be supplied in a call to an API function such as `musicxmlfile2lilypond ()`, in an options and arguments argument.
  See the MusicFormatsAPIUserGuide for the details;
- options are introduced either by `-` or `--`, which can be used at will. Both ways are equivalent;
- all options have a long name, and some have a complementary short name. The latter is not provided if the long name is short enough, such as `-jianpu`, `-cubase`, `-ambitus` or `-custos`.
  Short and long names can be used and mixed at will in the command line and in option vectors (API). Apart from very common options such as `-o`, the short names are meant for interactive use. This document uses only long name, which are more explicit in general;
- some short option names are supplied as is usual in open sotfware, such as `-h` (help), and `-o` (output file name):

```
jacquesmenu@macmini > xml2ly -query o
--- Help for atom "o" in subgroup "Files"
    -output-file-name, -o FILENAME
          Write output to file FILENAME instead of standard output.
```

- options and arguments such as file names can be intermixed at will. Thus:

```
xml2ly --display-cpu-usage basic/HelloWorld.xml
```

  and

```
xml2ly basic/HelloWorld.xml -display-cpu-usage
```

  produce the exact same result;

- some options names, either long or short, share a common prefix. This allows them to be *contracted*, as in `-h=rests,notes`, which is equivalent to `-hrests, -hnotes`, and `-trace=voices,notes`, equivalent to `-trace-voices, -trace-notes`:

```
jacquesmenu@macmini > xml2ly -query h
--- Help for prefix "h" ---
        '-h=abc,wxyz' is equivalent to '-habc, -hwxyz'

--- Help for atom "h" in subgroup "Options and help"
    -help, -h
          Display xml2ly's full help.
```

- the single-character options can be *clustered*: `-vac` is equivalent to: `-v, -a, -c`:

```
jacquesmenu@macmini > xml2ly -va
Command line version of musicxml2lilypond converter v0.9.52 (November 29, 2021)
A member of MusicFormats v0.9.59 (January 4, 2022)

Representations versions:
  MXSR
    v0.9.5 (October 6, 2021)
  MSR
    v0.9.52 (November 27, 2021)
  LPSR
    v0.9.5 (October 6, 2021)

Passes versions:
  mxsr2msr
    v0.9.51 (November 27, 2021)
  msr2msr
    v0.9.51 (November 15, 2021)
  msr2lpsr
    v0.9.5 (October 6, 2021)
  lpsr2lilypond
    v0.9.52 (December 16, 2021)
What xml2ly does:

    This multi-pass converter 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
                 a first Music Score Representation (MSR) skeleton;
        Pass 2b: populates the first MSR skeleton from the MusicXML tree
                 to get a full MSR;
        Pass 3:  converts the first MSR into a second MSR to apply options
        Pass 4:  converts the second MSR into a
                 LilyPond Score Representation (LPSR);
        Pass 5:  converts the LPSR to LilyPond code
                 and writes it to standard output.

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

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

14.1 Options characteristics

There are various options in MusicFormats for various needs. Every option controls a feature of a component or specifies a value used in the operation of the library.

An option can be:

- pure help: it provides information to the user, but does not do anything musical, such as option `-contact, -c`:

```
jacquesmenu@macmini > xml2ly -contact
To contact the maintainers of xml2ly:
    Create an issue at https://github.com/jacques-menu/musicformats,
    describing the problem and any error messages you get if relevant.
    You should sign up for d GitHub for that.
```

- self-sufficient, such as option `-quiet, -q`:

```
jacquesmenu@macmini > xml2ly -query quiet
--- Help for atom "quiet" in subgroup "Warning and errors"
    -quiet, -q
          Don't issue any warning or error messages.
```

- expecting a value, which must be supplied right after the option name:

```
jacquesmenu@macmini > xml2ly -query msr-pitches-language
--- Help for atom "msr-pitches-language" in subgroup "Notes"
    -msr-pitches-language, -mplang LANGUAGE
          Use LANGUAGE to display note pitches in the MSR logs and text views.
          The 13 MSR pitches languages available are:
          arabic, catalan, deutsch, english, espanol, francais,
          italiano, nederlands, norsk, portugues, suomi, svenska and vlaams.
          The default is 'kQTPNederlands'.
```

- expecting an optional value, supplied with a '=' without any spaces: a default value is used if none is provided by the user, such as **-name-help, -nh**, presented in more detail at section 15.2, [Querying about options by name], page 56:

```
jacquesmenu@macmini > xml2ly -name-help=output-file-name
--- Help for atom "output-file-name" in subgroup "Files"
    -output-file-name, -o FILENAME
          Write output to file FILENAME instead of standard output.
```

Some options can be used *several times*, while the others can be used only once.

14.2 The -insider option

As mentioned above, the MusicFormats library components, i.e. representations, passes, converters and generators, have options and help attached to them. There are also other 'global' sets of options, independently of the individual components themselves.

MusicFormats has to 'modes' for options and help handling:

- in *regular* mode, the default, the options are grouped by subject, such as tuplets or chords. In other words, there are grouped in a user-oriented way;
- in *insider* mode, they are grouped as there are used internally by MusicFormats behind the scenes, in an implementation-oriented way, hence the name.

Switching from the default regular mode to the insider mode is done with the **-insider, -ins** option:

```
jacquesmenu@macmini > xml2ly -query insider
--- Help for atom "insider" in subgroup "Options and help"
    -insider, -ins
          Use the 'insider' mode for the options and help,
          in which the options are grouped as they are used internally by MusicFormats.
          In the 'regular' defaut mode, they are grouped by user-oriented topics,
          such a slurs, tuplets and figured bass.
```

In regular mode, the options are displayed in subgroups only. The groups containing them are not displayed for simplicity, because a three-level options hierarchy is not what users expect and are used to.

For example, the **-ignore-ornaments, -oorns** option is displayed this way in regular mode:

```
jacquesmenu@macmini > xml2ly -query ignore-ornaments
--- Help for atom "ignore-ornaments" in subgroup "Ornaments"
    -ignore-ornaments, -oorns
          Ignore ornaments in MusicXML data.
```

In insider mode, on the contrary, the full group-subgroup-atom hierarchy is visible, as well as the attachment of the options to the groups managed internally by MusicFormats:

```
jacquesmenu@macmini > xml2ly -query ignore-ornaments -insider
--- Help for atom "ignore-ornaments" in subgroup "Notes" of group "mxsr2msr" ---
    -ignore-ornaments, -oorns
          Ignore ornaments in MusicXML data.
```

To summarize things up, it can be said that the regular mode offers a user-oriented *view* of the options available in the insider mode.

14.3 Early options

A particular case of options is the *early options*, which are taken into account prior to the options being actually analyzed. This the case of option `-insider, -ins`, since the whole set of possible options depends on it being used or not.

An early option should be supplied in the command line itself. Having it in included options and arguments files prevents it from being applied early. See chapter 17, [Including options and arguments from a file], page 66 about this feature.

Chapter 15

Options and help introspection

15.1 Restricting help to a given group or subgroup

The OAH groups and subgroups can be displayed with their own options, such as option `-help-midi, -hmidi`:

```
jacquesmenu@macmini: ~ >  xml2ly -help-midi
--- Help for subgroup "MIDI" in group "MIDI group" ---
  MIDI group (-help-midi-group, -hmidi-group):
  --------------------------
    MIDI (-help-midi, -hmidi):
      -no-midi
            Generate the '\midi' block as a comment instead of active code.
      -midi-tempo, -mdtempo MIDI_TEMPO_SPEC
            Generate a '\tempo' command in the \midi block.
            MIDI_TEMPO_SPEC can be:
            'DURATION = PER_SECOND'
            or
            "DURATION = PER_SECOND" .
            DURATION is a string such as '8.', and PER_SECOND is an integer.
            The single or double quotes are used to allow spaces around the '=' sign,
            otherwise they can be dispensed with.
            Using double quotes allows for shell variables substitutions, as in:
            PER_SECOND=66
            xml2ly -midiTempo "8. ${PER_SECOND}" .
            The default is '8 = 180'.
```

15.2 Querying about options by name

One can obtain help on any specific group, sub-group or atom with the `-query` option:

```
jacquesmenu@macmini > xml2ly -query query
--- Help for atom "query" in subgroup "Options and help"
    -query OPTION_NAME
          Print help about OPTION_NAME.
```

```
jacquesmenu@macmini > xml2ly -query output-file-name
--- Help for atom "output-file-name" in subgroup "Files"
    -output-file-name, -o FILENAME
          Write output to file FILENAME instead of standard output.
```

Another option exists to obtain the same result: `-name-help, -nh` has an optional value:

```
jacquesmenu@macmini > xml2ly -name-help=output-file-name
--- Help for atom "output-file-name" in subgroup "Files"
    -output-file-name, -o FILENAME
          Write output to file FILENAME instead of standard output.
```

The default value if none is supplied is... `name-help` itself:

```
jacquesmenu@macmini > xml2ly -name-help
--- Help for atom "name-help" in subgroup "Options and help"
    -name-help, -nh OPTION_NAME
          Print help about OPTION_NAME.
          OPTION_NAME is optional, and the default value is 'name-help'.
```

Choosing one option of the other is a matter of taste. To be honest, `-name-help, -nh` has been created to illustrate optional values...

15.3 Searching the help for a string

The MusicFormats services have a great number of options. Option `-find` comes in handy to search the available help:

```
jacquesmenu@macmini > xml2ly -query find
--- Help for atom "find" in subgroup "Options and help"
    -find STRING
          Find string STRING in the help.
          The search is case insensitive, and a '-' is added in front of options names for
     clarity.
```

```
jacquesmenu@macmini > xml2ly -find output-file-name
2 occurrences of string "output-file-name" have been found:
   1:
    -output-file-name, -o FILENAME
    Write output to file FILENAME instead of standard output.
   2:
    -auto-output-file-name, -aofn
    This option can only be used when writing to a file.
                  Writethe output to a file in the current working directory.
                  The file name is derived from that of the input file:
                  any suffix after the '.' is replaced by one suited for the output
    format,
                  or such a suffix is adde if no '.' is present.
```

15.4 Displaying help about options usage

A minimal version of this chapter is displayed by the `--help-options-usage, -hou` option:

```
jacquesmenu@macmini > xml2ly -help-options-usage
xml2ly options usage:
  In xml2ly, '-' as an argument, represents standard input.

  Most options have a short and a long name for commodity.
  The long name may be empty if the short name is explicit enough.

  The options are organized in a group-subgroup-atom hierarchy.
  Help can be obtained for groups or subgroups at will,
  as well as for any option with the '-name-help, -nh' option.

  A subgroup can be showm as a header only, in which case its description is printed
```

```
  only when the corresponding short or long names are used.

  Both '-' and '--' can be used to introduce options,
  even though the help facility only shows them with '-'.

  There some prefixes to allow for shortcuts,
  such as '-t=voices,meas' for '-tvoices, -tmeas'.

  The options can be placed in any order,
  provided the values immediately follow the atoms that need them.

  Using options that attempt to create files, such as '-o, -output-file-name',
  leads to an error if the environment is read-only access,
  as is the case of https://libmusicxml.grame.fr .
```

15.5 Displaying a help summary

This can be done with the `-help-summary, -hs` option:

```
jacquesmenu@macmini > xml2ly -query help-summary
--- Help for atom "help-summary" in subgroup "Options and help"
    -help-summary, -hs
          Display xml2ly's help summary.
```

Chapter 16

Options examples

16.1 Boolean options

Most of the options are boolean : the feature they control is `false` by default, and is set to `true` when the option is used, such as:

```
jacquesmenu@macmini > xml2ly -query display-cpu-usage
--- Help for atom "display-cpu-usage" in subgroup "Informations"
    -display-cpu-usage, -cpu
          Write information about CPU usage to standard error.
```

16.2 Options simple values

There are options to supply value of various types to the services, such a strings, integers, floating numbers and rationals:

```
jacquesmenu@macmini: ~ > xml2ly -query page-count
--- Help for atom "page-count" in subgroup "Paper"
    -page-count PAGE_COUNT
          Set the LilyPond 'page-count' paper variable to PAGE_COUNT in the LilyPond code.
          PAGE_COUNT should be a positive integer.
          By default, this is left to LilyPond'.
```

```
jacquesmenu@macmini: ~ > xml2ly -query msr-ignore-musicxml-part-id
--- Help for atom "msr-ignore-musicxml-part-id" in subgroup "Parts"
    -msr-ignore-musicxml-part-id, -mompi PART_ID
          Ignore the part with ID PART_ID, which is a string.
          There can be several occurrences of this option.
          All the parts not ignored are kept.
          This option is incompatible with '-mkpi, -msr-keep-musicxml-part-id'.
```

```
jacquesmenu@macmini: ~ > xml2ly -query global-staff-size
--- Help for atom "global-staff-size" in subgroup "Layout"
    -global-staff-size, -gss NUMBER
          Set the LilyPond '#(set-global-staff-size ...)' to NUMBER in the LilyPond code.
          NUMBER should be a floating point or integer number.
          The default is '20.000000'.
```

```
jacquesmenu@macmini: ~ > xml2ly -query lilypond-lyrics-durations-style
--- Help for atom "lilypond-lyrics-durations-style" in subgroup "Lyrics"
    -lilypond-lyrics-durations-style, -ld STYLE
          The 2 LilyPond lyrics durations STYLEs available are:
          explicit and implicit.
          Using 'implicit' prevents the creation of lyrics attached to rests by LilyPond,
          use 'explicit' in such cases.
          The default is 'explicit'.
```

```
jacquesmenu@macmini: ~ > xml2ly -query delayed-ornaments-fraction
--- Help for atom "delayed-ornaments-fraction" in subgroup "Ornaments"
    -delayed-ornaments-fraction, -dof NUM/DENOM
          Place the delayed turn/reverseturn at the given fraction
          between the ornemented note and the next one.
          The default is '1/2'.
```

16.3 Options more complex values

There are options to supply value of various type to the services. Here are some examples:

```
jacquesmenu@macmini: ~ > xml2ly -query top-margin
--- Help for atom "top-margin" in subgroup "Paper"
    -top-margin MARGIN
          Set the LilyPond 'top-margin' paper variable to MARGIN in the LilyPond code.
          WIDTH should be a positive floating point or integer number,
          immediately followed by a unit name, i.e. 'in', 'mm' or 'cm'.
          By default, this is left to LilyPond'.
```

```
jacquesmenu@macmini: ~ > xml2ly -query msr-replace-clef
--- Help for atom "msr-replace-clef" in subgroup "Clefs"
    -msr-replace-clef, -mrc REPLACE_CLEF_SPEC
          Raplace clef ORIGINAL_CLEF by NEW_CLEF.
          REPLACE_CLEF_SPEC can be:
          'ORIGINAL_CLEF = NEW_CLEF'
          or
          "ORIGINAL_CLEF = NEW_CLEF"
          The single or double quotes are used to allow spaces in the clef names
          and around the '=' sign, otherwise they can be dispensed with.
          The 23 clefs available are:
          treble, soprano, mezzosoprano, alto, tenor, baritone, bass,
          treble1, treble-15, treble-8, treble+8, treble+15, bass-15, bass-8,
          bass+8, bass+15, varbaritone, tab4, tab5, tab6, tab7, percussion and
          jianpu.
          There can be several occurrences of this option.
```

```
jacquesmenu@macmini: ~ > xml2ly -query ledger-lines-color
--- Help for atom "ledger-lines-color" in subgroup "Staves"
    -ledger-lines-color, -llc RGB_COLOR
          Use RGB_COLOR for the ledger lines.
          RGB_COLOR should be of the form 'r,g,b',
          with r, g and b being float numbers between 0.0 and 1.0 inclusive.
```

```
jacquesmenu@macmini: ~ > xml2ly -query lilypond-transpose-part-name
--- Help for atom "lilypond-transpose-part-name" in subgroup "Parts"
    -lilypond-transpose-part-name, -lilytpn PART_TRANSPOSITION_SPEC
          Transpose part PART_NAME using TRANSPOSITION in the LilyPond code.
          PART_TRANSPOSITION_SPEC can be:
          'PART_NAME = TRANSPOSITION'
          or
          "PART_NAME = TRANSPOSITION"
          The single or double quotes are used to allow spaces in the names
```

```
          and around the '=' sign, otherwise they can be dispensed with.
          TRANSPOSITION should contain a diatonic pitch, followed if needed
          by a sequence of ',' or '\'' octave indications.
          Such indications cannot be mixed, and they are relative to c\', i.e. middle C.
          For example, 'a', 'f' and 'bes,' can be used respectively
          for instruments in 'a', 'f' and B flat respectively.
          Using double quotes allows for shell variables substitutions, as in:
          SAXOPHONE="bes,"
          EXECUTABLE -lilypond-transpose-part-name "P1 ${SAXOPHONE}" .
          There can be several occurrences of this option.
```

```
jacquesmenu@macmini: ~ > xml2ly -query relative
--- Help for atom "relative" in subgroup "Notes"
    -relative, -rel PITCH_AND_OCTAVE
          Use relative octave entry reference PITCH_AND_OCTAVE in the generated LilyPond
    code.
          PITCH_AND_OCTAVE is made of a diatonic pitch and
          an optional sequence of commas or single quotes.
          It should be placed between double quotes if it contains single quotes, such as:
            -rel "c''".
          The default is to use LilyPond's implicit reference 'f'.
```

```
jacquesmenu@macmini: ~ > xml2ly -query lilypond-accidental-style
--- Help for atom "lilypond-accidental-style" in subgroup "Notes"
    -lilypond-accidental-style, -as STYLE
          STYLE should be one of the 18 LilyPond accidental styles available:
          default, dodecaphonic, dodecaphonic-first,
          dodecaphonic-no-repeat, forget, modern, modern-cautionary, modern-voice,
          modern-voice-cautionary, neo-modern, neo-modern-cautionary, neo-modern-voice,
          neo-modern-voice-cautionary, no-reset, piano, piano-cautionary, teaching and
    voice.
          The default is 'default'.
```

```
jacquesmenu@macmini: ~ > xml2ly -query chords-display
--- Help for atom "chords-display" in subgroup "Chords"
    -chords-display, -chd SPECIFICATION
          Use SPECIFICATION to display chords using LilyPond's chordNameExceptions.
          SPECIFICATION should contain a chord contents such as '<c ees ges bes>',
          followed by code to display it, for example:
            '<c ees ges bes> \\super {"-7(" {\\small\\raise #0.5 \\flat} "5)"}'.
          The LilyPond code has to escape backslashes, thus use '\\' to obtain '\'.
          These two elements are passed over to LilyPond verbatim, without any check.
          This option can be used any number of times.
```

16.4 More complex options

The boolean options in MusicFormats can be combined:

```
jacquesmenu@macmini: ~ > xml2ly -query cubase
--- Help for atom "cubase" in subgroup "Input"
    -cubase:
          Useful settings for MusicXML data exported from Cubase.
          This combined option is equivalent to:
            -ignore-redundant-clefs, -irclefs:
              Ignore clefs that are the same as the current one.
            -ignore-redundant-keys, -irkeys:
              Ignore keys that are the same as the current one.
            -ignore-redundant-times, -irtimes:
              Ignore times that are the same as the current one.
jacquesmenu@macmini: ~ >
```

Options can also share a common prefix:

```
jacquesmenu@macmini: ~ > xml2ly -query trace-when-handling-musicxml-data -insider
--- Help for atom "trace-when-handling-musicxml-data" in subgroup "MXSR Trace" of group "
    Mxsr" ---
    -t<SHORT_NAME>, -trace--<LONG_NAME>
          Trace SHORT_NAME/LONG_NAME when handling MusicXML data.
    The 2 known SHORT_NAMEs are:
      enc and divs.
    The 2 known LONG_NAMEs are:
      encoding and divisions.
```

OAH offers *macro options*, such as:

```
--- Help for atom "auto-utf8d" in subgroup "Files"
    -auto-utf8d, -au8d:
          To ease the production of braille files.
          This macro option is equivalent to:
            -auto-output-file-name, -aofn:
              This option can only be used when writing to a file.
              Writethe output to a file in the current working directory.
              The file name is derived from that of the input file:
              any suffix after the '.' is replaced by one suited for the output format,
              or such a suffix is adde if no '.' is present.
            -use-encoding-in-file-name, -ueifn:
              Append a description of the encoding used
              and the presence of a BOM if any to the file name before the '.'.
```

And finally, this macro option can be used to obtain informations on the fly and write the output to a file automatically:

```
jacquesmenu@macmini: ~ > xml2ly -query debug
--- Help for atom "debug" in subgroup "Options and help"
    -debug:
          To help debugging musicxml2lilypond.
          This macro option is equivalent to:
            -trace-passes, -tpasses:
              Write a trace of the passes to standard error.
            -auto-output-file-name, -aofn:
              This option can only be used when writing to a file.
              Writethe output to a file in the current working directory.
              The file name is derived from that of the input file:
              any suffix after the '.' is replaced by one suited for the output format,
              or such a suffix is adde if no '.' is present.
            -display-cpu-usage, -cpu:
              Write information about CPU usage to standard error.
```

16.5 Displaying the options values

This can be done with the `-display-options-values, -dov` option:

```
jacquesmenu@macmini > xml2ly -query display-options-values
--- Help for atom "display-options-values" in subgroup "Options and help"
    -display-options-values, -dov
          Write the chosen options values to standard error.
```

Executing this command:

```
jacquesmenu@macmini > xml2ly -global-staff-size 30 -display-cpu-usage -display-options-
    values
  The options values for xml2ly are:
    Informations group (-help-informations-group, -hinfos-group), 1 atom chosen:
    -------------------------
      Informations (-help-informations, -hinfos), 1 atom chosen:
        fDisplayCPUusage                        : true, has been set by user

    Options and help group (-help-oah-group, -hoah-group), 1 atom chosen:
    -------------------------
      Options and help (-help-oah, -hoah), 1 atom chosen:
        fDisplayOptionsValues                   : true, has been set by user

    Layout group (-help-layout-group, -hlayout-group), 1 atom chosen:
    -------------------------
      Layout (-help-layout, -hlayout), 1 atom chosen:
        fGlobalStaffSize                        : 30, has been set by user

Input file name or '-' for standard input expected
```

A exhaustive display of all the options values, chosen by the user or not, can be obtained with `-display-options-valu` `-dova`: displays the whole set of options with their values, and whether they have been set by the user:

```
jacquesmenu@macmini > xml2ly -global-staff-size 30 -display-cpu-usage -display-options-
    values-all

  All the options values for xml2ly are:

    OAH Trace (-help-trace, -ht):
    -------------------------
      Other (-help-trace-other, -hto):
        fTraceComponents                        : false
        fTracePasses                            : false
        fTraceGeometry                          : false
        fTraceIdentification                    : false
        fTraceForTests                          : false

    ... ... ...

    Informations group (-help-informations-group, -hinfos-group):
    -------------------------
      Informations (-help-informations, -hinfos):
        fDisplayCPUusage                        : true, has been set by user

    Files group (-help-files-group, -hfiles-group):
    -------------------------
      Files (-help-files, -hfiles):
        fOutputFileName                         :
        fAutoOutputFileName                     : false

    Options and help group (-help-oah-group, -hoah-group):
    -------------------------
      Options and help (-help-oah, -hoah):
        insider                                 : fOptionHasBeenSelected: false
        fOahVerboseMode                         : false
        fReverseNamesDisplayOrder               : false
        fDisplayOptionsValues                       : true, has been set by user

    ... ... ...

    Staves group (-help-staves-group, -hstaves-group):
    -------------------------
      Staves (-help-staves, -hstaves):
        fCreateVoicesStaffRelativeNumbers       : false
        fLedgerLinesRGBColor                    : [0,0,0]
```

```
42
43      ... ... ...
44
45      Notes group (-help-notes-group, -hnotes-group):
46      -------------------------
47        Notes (-help-notes, -hnotes):
48          fMsrQuarterTonesPitchesLanguageKind      : kQTPNederlands
49          OctaveEntryVariable :
50            fOctaveEntryKind                         : kOctaveEntryAbsolute
51           : none
52          OctaveEntryVariable :
53            fOctaveEntryKind                         : kOctaveEntryAbsolute
54           : none
55          fWhiteNoteHeads                          : false
56          fGenerateStemsDirections                 : false
57          fGenerateCommentedOutVariables           : false
58          fGenerateLpsrVisitingInformation         : false
59          fAccidentalStyleKind                     : kAccidentalStyleDefault
60          fNonPrintNotesHeadRGBColor               : [0,0,0]
61
62      ... ... ...
63
64      Paper group (-help-paper-group, -hpaper-group):
65      -------------------------
66        Paper (-help-paper, -hpaper):
67          fPaperHeight                             : [297 kUnitMillimeter]
68          fPaperWidth                              : [210 kUnitMillimeter]
69          fPaperLeftMargin                         : [15 kUnitMillimeter]
70          fPaperRightMargin                        : [15 kUnitMillimeter]
71          fPaperTopMargin                          : [15 kUnitMillimeter]
72          fPaperBottomMargin                       : [15 kUnitMillimeter]
73          fRaggedBottom                            : false
74          fRaggedLast                              : false
75          fRaggedLastBottom                        : false
76          fRaggedRight                             : false
77          fPaperHorizontalShift                    : [0 kUnitMillimeter]
78          fPaperIndent                             : [0 kUnitMillimeter]
79          fPaperShortIndent                        : [0 kUnitMillimeter]
80          fMarkupSystemSpacingPadding              : [0 kUnitMillimeter]
81          fPageCount                               : 0
82          fSystemCount                             : 0
83
84      Layout group (-help-layout-group, -hlayout-group):
85      -------------------------
86        Layout (-help-layout, -hlayout):
87          fGlobalStaffSize                         : 30, has been set by user
88          fKeepStaffSize                           : false
89
90      ... ... ...
91
92      MIDI group (-help-midi-group, -hmidi-group):
93      -------------------------
94        MIDI (-help-midi, -hmidi):
95          fNoMidi                                  : false
96          fMidiTempo                               : [MidiTempo, midiTempoDuration = "8",
      midiTempoPerSecond = 180, line 0]
97
98  Input file name or '-' for standard input expected
```

16.6 Displaying MusicFormats internal data

MusicFormats provides many options to display its internals, including the representations it builds. Option `-find` can be used to see the various possibilities:

```
jacquesmenu@macmini > xml2ly -find display-
```

For example, consider `xml2ly`:

```
jacquesmenu@macmini > xml2ly -about
What xml2ly does:

    This multi-pass converter 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
                 a first Music Score Representation (MSR) skeleton;
        Pass 2b: populates the first MSR skeleton from the MusicXML tree
                 to get a full MSR;
        Pass 3:  converts the first MSR into a second MSR to apply options
        Pass 4:  converts the second MSR into a
                 LilyPond Score Representation (LPSR);
        Pass 5:  converts the LPSR to LilyPond code
                 and writes it to standard output.

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

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

The LSPR built in pass 4 and used in pass 5 to create the LilyPond output can be displayed with the following options:

```
jacquesmenu@macmini > xml2ly -find display-lpsr
3 occurrences of string "display-lpsr" have been found:
   1:
    -display-lpsr, -dlpsr
    Write the contents of the LPSR data with a summary of it MSR component to standard
    error.
   2:
    -display-lpsr-full, -dlpsrfull
    Write the contents of the LPSR data with its full MSR component to standard error.
   3:
    -display-lpsr-short, -dlpsrshort
    Write the contents of the LPSR data, short version, to standard error.
```

The resulting output is large of course, since LSPR represents the score in great detail. It can be used by curious users, and is a great help to the maintainers of MusicFormats.

Chapter 17

Including options and arguments from a file

MusicFormats converters have an `-include, -inc` option for this:

```
jacquesmenu@macmini > xml2ly -query include
--- Help for atom "include" in subgroup "Options and help"
    -include, -inc FILENAME
          Include the options and arguments contained in FILENAME.
          FILENAME is a string and should be a path to a text file.
          Such a file is expected to hold at most one option or argument per line.
          A '#' starts a comment that spans to the end of the line.
          Comments and empty lines are ignored and can be used at will.
          '-include, -inc' options may be used to include other files,
          up to a maximum level of 10.
          This is handy to share often used options in groups, for example.
```

Note that the current MusicFormats services can take at most one argument, that can be either a file name or '-', that designates the standard input stream.

17.1 An options and arguments file example

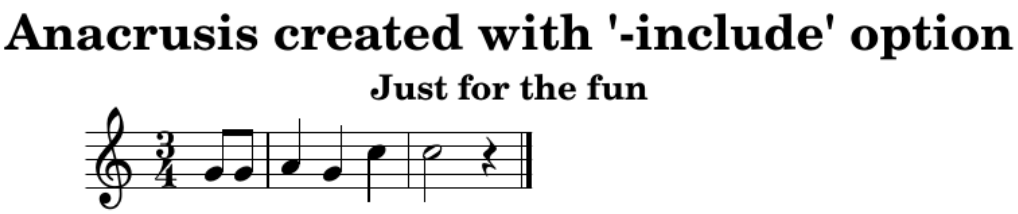
A file that be included with the option sample is basic/AnacrusisOptionsAndArguments.txt:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/
    AnacrusisOptionsAndArguments.txt
# some options

  # output file
  -auto-output-file-name

  # contents
  -title "Anacrusis created with '-include' option"
  -subtitle "Just for the fun"

  # layout
  -global-staff-size 30

  # non-musical
  -cpu

# the MusicXML file

  basic/Anacrusis.xml
```

Including this file with `xml2ly` gives:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -include basic/
    AnacrusisOptionsAndArguments.txt
Timing information:

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

           Handle the options and arguments from argc/argv            mandatory      0.03038
Pass 1     Create an MXSR reading a MusicXML file                     mandatory      0.00353
Pass 2a    Create an MSR skeleton from the MXSR                       mandatory      0.00071
Pass 2b    Populate the MSR skeleton from MusicXML data               mandatory      0.00139
Pass 3     Convert the first MSR into a second MSR                    mandatory      0.00037
Pass 4     Convert the second MSR into an LPSR                        mandatory      0.00039
Pass 5     Convert the LPSR score to LilyPond code                    mandatory      0.00088

Total (sec)  Mandatory  Optional
-----------  ---------  ---------
0.03766      0.03766    0.00000

jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > ls -sal Anacrusis.ly
8 -rw-r--r--@ 1 jacquesmenu  staff  1553 Feb  9 09:44 Anacrusis.ly
```

The resulting score is:



17.2 Options values and arguments in included files

As shown in section 9.4, [Quoting, variables and aliases], page 36, the shell identifies words in the command line. This is why options values and arguments have to be inclosed in quotes or double quotes when they contain spaces.

In included files, these values are merely extracted from a line, and taken verbatim. To ease copying/pasting from the command line though, any quotes or double quotes around the values are ignored.

For example, basic/QuotingInIncludedOptionsFiles.txt contains:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/basic > cat
    QuotingInIncludedOptionsFiles.txt
# contents
  -title This year's title
  -subtitle 'Last year's quoted multi-word subtitle'
  -subsubtitle "Double quoted multi-word subsubtitle"

# display
  -display-options-values

# LilyPond
  -lilypond-generation-infos

# output
  -auto-output-file-name
```

Including this file and displaying the options values, we get:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/basic > xml2ly -include
    QuotingInIncludedOptionsFiles.txt HelloWorld.xml
  The options values for xml2ly are:
    Files group (-help-files-group, -hfiles-group), 1 atom chosen:
    -------------------------
      Files (-help-files, -hfiles), 1 atom chosen:
        fAutoOutputFileName                     : true

    Options and help group (-help-oah-group, -hoah-group), 1 atom chosen:
    -------------------------
      Options and help (-help-oah, -hoah), 1 atom chosen:
        fDisplayOptionsValues                   : true

    Header group (-help-header-group, -hheader-group), 3 atoms chosen:
    -------------------------
      Header (-help-header, -hheader), 3 atoms chosen:
        fTitle                                  : This year's title
        fSubTitle                               : Last year's quoted multi-word subtitle
        fSubSubTitle                            : Double quoted multi-word subsubtitle

    Output group (-help-output-group, -houtput-group), 1 atom chosen:
    -------------------------
      Output (-help-output, -houtput), 1 atom chosen:
        fXml2lyInfos                            : true
```

17.3 Multi-level includes

A file containing options and argument may itself use the `-include, -inc` option, which allows for options to be shared easily for various uses of the services.

Note, however, that early options are detected *before* the files inclusion are performed. In particular, the `-insider, -ins` option should be in the command line itself, at the top level so to say, to be taken into account.

For example, basic/HelloWorldOptionsAndArguments_1.txt contains:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/
    HelloWorldOptionsAndArguments_1.txt
# output file
 -auto-output-file-name

# contents
  -title 'My title'
  -subtitle "  Nice subtitle"
  -subsubtitle "Subsubtitle from HelloWorldOptionsAndArguments_1.txt"

# layout
  -global-staff-size 30

# non-musical
  -display-cpu-usage

# the MusicXML file
  basic/HelloWorld.xml

# nested include
  -include basic/HelloWorldOptionsAndArguments_2.txt
```

The included basic/HelloWorldOptionsAndArguments_2.txt file contains:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/
    HelloWorldOptionsAndArguments_2.txt
# non-musical
  -subsubtitle "Subsubtitle from HelloWorldOptionsAndArguments_2.txt"

# cycle detection check
#  -include HelloWorldOptionsAndArguments_1.txt
```

Including basic/HelloWorldOptionsAndArguments_1.txt, we get:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -include basic/
    HelloWorldOptionsAndArguments_1.txt
Timing information:

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

           Handle the options and arguments from argc/argv          mandatory      0.02962
Pass 1     Create an MXSR reading a MusicXML file                   mandatory      0.00362
Pass 2a    Create an MSR skeleton from the MXSR                     mandatory      0.00185
Pass 2b    Populate the MSR skeleton from MusicXML data             mandatory      0.00288
Pass 3     Convert the first MSR into a second MSR                  mandatory      0.00092
Pass 4     Convert the second MSR into an LPSR                      mandatory      0.00090
Pass 5     Convert the LPSR score to LilyPond code                  mandatory      0.00143

Total (sec)  Mandatory  Optional
-----------  ---------  ---------
0.04122      0.04122    0.00000
```

The resulting score is:

17.4 Multi-level includes overflow

There are resources limitations on the machines MusicFormats is used on, and we should prevent them to be overflown. This could occur is including a file runs into a loop in which the same file is included again.

MusicFormats prevents this by limiting the level of such includes.

Let us uncomment the `-includeinc` option in basic/HelloWorldOptionsAndArguments_2.txt, leading to:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/
    HelloWorldOptionsAndArguments_2.txt
# non-musical
  -subsubtitle "Subsubtitle from HelloWorldOptionsAndArguments_2.txt"

# cycle detection check
  -include basic/HelloWorldOptionsAndArguments_1.txt
```

Now we get:

```
jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -include basic/
    HelloWorldOptionsAndArguments_1.txt
                        Including file [basic/HelloWorldOptionsAndArguments_1.txt]: more
    than 10 include levels, quitting
                        The include file names stack contains 10 elements:
                           1: [basic/HelloWorldOptionsAndArguments_2.txt]
                           2: [basic/HelloWorldOptionsAndArguments_1.txt]
                           3: [basic/HelloWorldOptionsAndArguments_2.txt]
                           4: [basic/HelloWorldOptionsAndArguments_1.txt]
                           5: [basic/HelloWorldOptionsAndArguments_2.txt]
                           6: [basic/HelloWorldOptionsAndArguments_1.txt]
                           7: [basic/HelloWorldOptionsAndArguments_2.txt]
                           8: [basic/HelloWorldOptionsAndArguments_1.txt]
                           9: [basic/HelloWorldOptionsAndArguments_2.txt]
                          10: [basic/HelloWorldOptionsAndArguments_1.txt]
```

17.5 An include file example

This example shows what can be done after exporting a scanned PDF score file to MusicXML. The piece is Scott Joplin's Bethena.

The scanner used by this author does not export tempo information as such, but as MusicXML `<words/>`. The `-convert-musicxml-words-to-msr-tempo` options fix this by converting the specific occurrences to MSR tempo elements in the `mxml2msr` pass.

Also, the dynamic wedges are exported as being above the staves, which is fixed by option `-all-wedges-below`.

The score is to be used by a musician who needs very readable scores with large note heads and alterations, hence the `-global-staff-size`, `-page-count`, `-ragged-last-bottom` and `-repeat-brackets` options.

And finally, the slurs in the original were wrong, i.e. not the *swing phrasé* as should be, hence the use of option `-ignore-slurs`:

```
jacquesmenu@macmini > cat Bethena_SaxTenor_OptionsAndArguments.txt
# argument
  Bethena_SaxTenor.xml

# output
  -auto-output-file-name

# informations
  -lilypond-generation-infos

# presentation
  -global-staff-size 23
  -page-count 3
  -ragged-last-bottom

# words conversions
  -convert-musicxml-words-to-msr-tempo "Valse tempo (Thema)"
  -convert-musicxml-words-to-msr-tempo "A tempo (Valse cantabile)"
  -convert-musicxml-words-to-msr-tempo "A tempo"
  -convert-musicxml-words-to-msr-tempo "a tempo"
  -convert-musicxml-words-to-msr-tempo "Finale"
  -convert-musicxml-words-to-msr-tempo "Andante"
  -convert-musicxml-words-to-msr-tempo "Tempo I"

# tweaking
  -repeat-brackets

  -all-wedges-below

```

```
30 -ignore-slurs
```

Chapter 18

Non-musical options

MusicFormats supplies options to obtain informations without inferering with the conversion activities in any way.

18.1 Timing measurements

There is a option `-cpu` option to see show much time is spent in the various translation activities. Note that the numbers obtained depend on the other activities on the machine. Also, on recent versions of Mac OS™, the first run of an executable may be a bit slower that subsequent runs, because the operating system loads the code in a cache for further use:

```
menu@macbookprojm > xml2ly -query 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 -display-cpu-usage 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
```

```
17
18 Total       Mandatory   Optional
19 -------     ---------   ---------
20 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
```

18.2 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 (semitones) pitch
34           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$^{\flat}$ 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

  Chord 'bes dommin9' inversion 2 inner intervals:
      f     -> aes   : minorThird          (perfectFifth         -> minorSeventh)
      f     -> ces   : diminishedFifth     (perfectFifth         -> minorNinth)
      f     -> bes   : perfectFourth       (perfectFifth         -> perfectUnison)
      f     -> d     : majorSixth          (perfectFifth         -> majorThird)

      aes   -> ces   : minorThird          (minorSeventh         -> minorNinth)
      aes   -> bes   : majorSecond         (minorSeventh         -> perfectUnison)
      aes   -> d     : augmentedFourth     (minorSeventh         -> majorThird)

      ces   -> bes   : majorSeventh        (minorNinth           -> perfectUnison)
      ces   -> d     : augmentedSecond     (minorNinth           -> majorThird)

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

Chapter 19

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 option `-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 for the maintainance of MusicFormats,
  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 '-trace-passes, -tpasses'.
--------------------------
  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 (gtracingOah->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.

Part VI

Warnings and errors (WAE)

Chapter 20

Warnings and errors (WAE)

Part VII

Multiple languages support

Chapter 21

Multiple languages support

The MusicFormats components support a number of languages, most of which being taken over from MusicXML and LilyPond.

For example, `xml2ly` offers several languages options:

```
jacquesmenu@macmini > xml2ly -find language
6 occurrences of string "language" have been found:
   1:
    -msr-pitches-language, -mplang LANGUAGE
    Use LANGUAGE to display note pitches in the MSR logs and text views.
                    The 13 MSR pitches languages available are:
                    arabic, catalan, deutsch, english, espanol, francais,
                    italiano, nederlands, norsk, portugues, suomi, svenska and vlaams.
                    The default is 'kQTPNederlands'.
   2:
    -lpsr-pitches-language, -lppl LANGUAGE
    Use LANGUAGE to display note pitches in the LPSR logs and views,
                    as well as in the generated LilyPond code.
                    The 13 LPSR pitches languages available are:
                    arabic, catalan, deutsch, english, espanol, francais,
                    italiano, nederlands, norsk, portugues, suomi, svenska and vlaams.
                    The default is 'kQTPNederlands'.
   3:
    -lpsr-chords-language, -lpcl LANGUAGE
    Use LANGUAGE to display chord names, their root and bass notes,
                    in the LPSR logs and views and the generated LilyPond code.
                    The 5 LPSR chords pitches languages available are:
                    french, german, ignatzek, italian and semiGerman.
                    'ignatzek' is Ignatzek's jazz-like, english naming used by LilyPond by
     default.
                    The default is 'kChordsIgnatzek'.
   4:
    -show-all-harmonies-contents, -sacc PITCH
    Write all harmonies contents for the given diatonic (semitones) PITCH,
                    supplied in the current language to standard output.
   5:
    -show-harmony-details, -scd HARMONY_SPEC
    Write the details of the harmony for the given diatonic (semitones) pitch
                    in the current language and the given harmony to standard output.
                    HARMONY_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-harmony-details "bes ${HARMONY}"
   6:
    -show-harmony-analysis, -sca HARMONY_SPEC
```

```
    Write an analysis of the harmony for the given diatonic (semitones) pitch
                    in the current language and the given harmony to standard output.
                    HARMONY_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-harmony-analysis "bes ${HARMONY} ${INVERSION}"
```

Part VIII

xml2ly

Chapter 22

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 active in the LilyPond community, found it too long, and they chose xml2ly among other names this author proposed to them.

22.1 Why xml2ly?

LilyPond comes with musicxml2ly, a converter 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 converter 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.

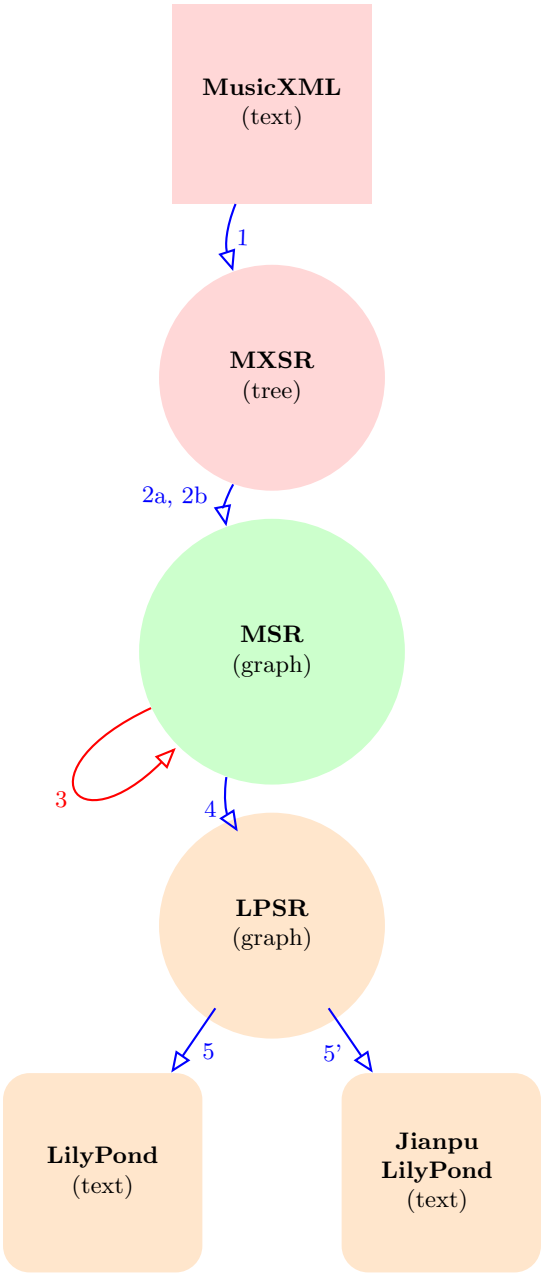
Speed was not an explicit goal, but as it turns out, xml2ly is not bad in this respect.

22.2 What xml2ly does

xml2ly performs the 5 steps from MusicXML to LilyPond to translate the former into the latter, as shown in figure 22.1, [xmlToLyArchitecture], page 84. Converting from MXSR to MSR is done in two sub-phases for implementation reasons.

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

Figure 22.1: xml2ly architecture



```
jacquesmenu@macmini > xml2ly -about
What xml2ly does:

    This multi-pass converter 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
                 a first Music Score Representation (MSR) skeleton;
        Pass 2b: populates the first MSR skeleton from the MusicXML tree
                 to get a full MSR;
        Pass 3:  converts the first MSR into a second MSR to apply options
        Pass 4:  converts the second MSR into a
                 LilyPond Score Representation (LPSR);
        Pass 5:  converts the LPSR to LilyPond code
                 and writes it to standard output.

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

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

Step 5' is merely step 5 plus the generation of a numbered score, which happens when the `-jianpu` option is used:

```
jacquesmenu@macmini > xml2ly -query jianpu
--- Help for atom "jianpu" in subgroup "Output"
    -jianpu
          Generate the score using the Jianpu (numbered) notation
          instead of the default western notation.
          This option needs lilypond-Jianpu to be accessible to LilyPond
          (https://github.com/nybbs2003/lilypond-Jianpu/jianpu10a.ly).
```

22.3 Useful options to xml2ly

Option `-avoid-msr2msr, -am2m` can be used to avoid running the src/passes/msr2msr/ pass:

```
jacquesmenu@macmini: ~ > xml2ly -query avoid-msr2msr
--- Help for atom "avoid-msr2msr" in subgroup "Rests"
    -avoid-msr2msr, -am2m
          Avoid the msr2msr pass, for TESTS.
```

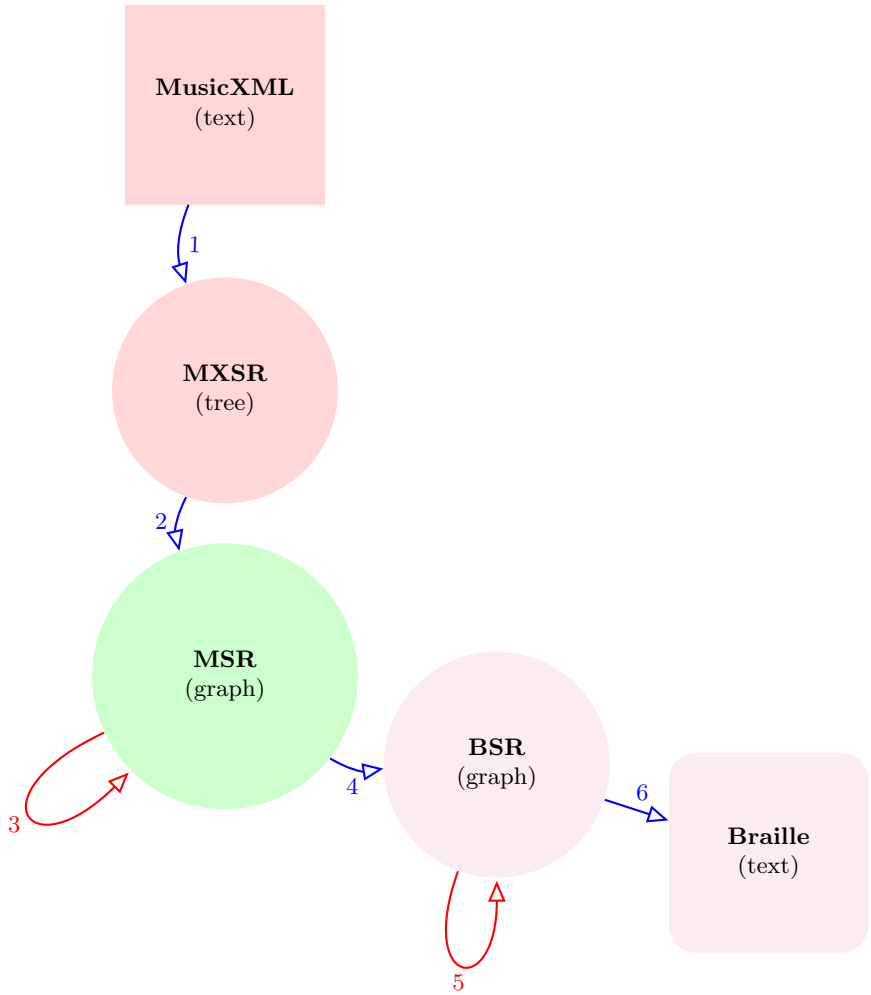
Part IX

xml2brl

Chapter 23

xml2brl

Figure 23.1: xml2brl architecture



23.1 Why xml2brl?

After first creating xml2ly, the design goals for xml2brl were:

- to experiment the re-use of MSR for other needs than generating LilyPond code;
- to provide a MusicXML to Braille transalator that might prove useful.

The first goal has been reached, but the second one has not at the time of this writing: nearly none of the individuals and bodies this author contacted to ask whom might help him with technical details about the generation of braille files answered.
So this whole effort got frozen at some point in time.

xml2brl is incomplete in that is does not support, by far, the full range of Braille complexities. Anyone interested may take over if needed, though, which is why this part of MusicFormats is presented in this document and detailed in the maintainance guide.

23.2 What xml2brl does

xml2brl performs the 5 steps from MusicXML to LilyPond to translate the former into the latter, as shown in figure 22.1, [xmlToLyArchitecture], page 84. Converting from MXSR to MSR is done in two sub-phases for implementation reasons.

The '-about' option to xml2brl details that somewhat:

```
jacquesmenu@macmini > xml2brl -about
What xml2brl does:

    This multi-pass converter basically performs 6 passes:
        Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
                 and converts it to a MusicXML tree;
        Pass 2a: converts that MusicXML tree into
                 a first Music Score Representation (MSR) skeleton;
        Pass 2b: populates the MSR skeleton from the MusicXML tree
                 to get a full MSR;
        Pass 3:  converts the first MSR into a second MSR, to apply options
        Pass 4:  converts the second MSR into
                 a first Braille Score Representation (BSR)
                 containing one Braille page per MusicXML page;
        Pass 5:  converts the first BSR into a second BSR
                 with as many Braille pages as needed
                 to fit the line and page lengthes;
        Pass 6:  converts the BSR to Braille text
                 and writes it to standard output.

    In this preliminary version, pass 3 merely clones the MSR it receives.

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

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

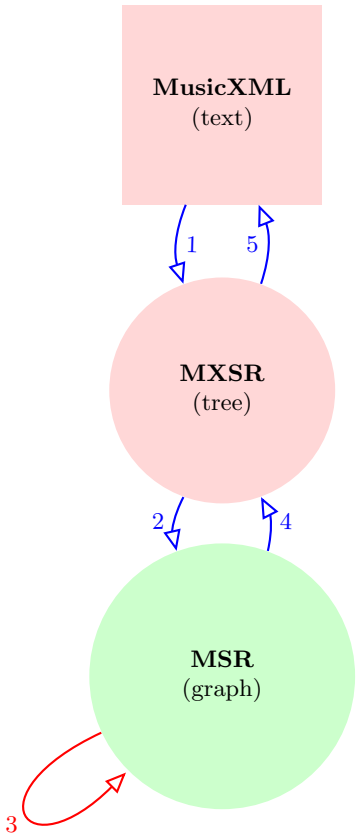
Part X

xml2xml

Chapter 24

xml2xml

Figure 24.1: xml2xml architecture



24.1 Why xml2xml?

xml2xml has been designed to operate on MusicXML data, applying options to apply the desired changes. It does a good job already, and will be completed as needed.

24.2 What xml2xml does

xml2xml performs the 5 steps from MusicXML to LilyPond to translate the former into the latter, as shown

in figure 22.1, [xmlToLyArchitecture], page 84. Converting from MXSR to MSR is done in two sub-phases for implementation reasons.

The '-about' option to xml2xml details that somewhat:

```
jacquesmenu@macmini > xml2xml -about
What xml2xml does:

    This multi-pass converter basically performs 6 passes:
        Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
                 and converts it to a MusicXML tree;
        Pass 2a: converts that MusicXML tree into
                 a first Music Score Representation (MSR) skeleton;
        Pass 2b: populates the MSR skeleton from the MusicXML tree
                 to get a full MSR;
        Pass 3:  converts the first MSR into a second MSR, to apply options;
        Pass 4:  converts the second MSR into a second MusicXML tree;
        Pass 5:  converts the second MusicXML tree to MusicXML code
                 and writes it to standard output.

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

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

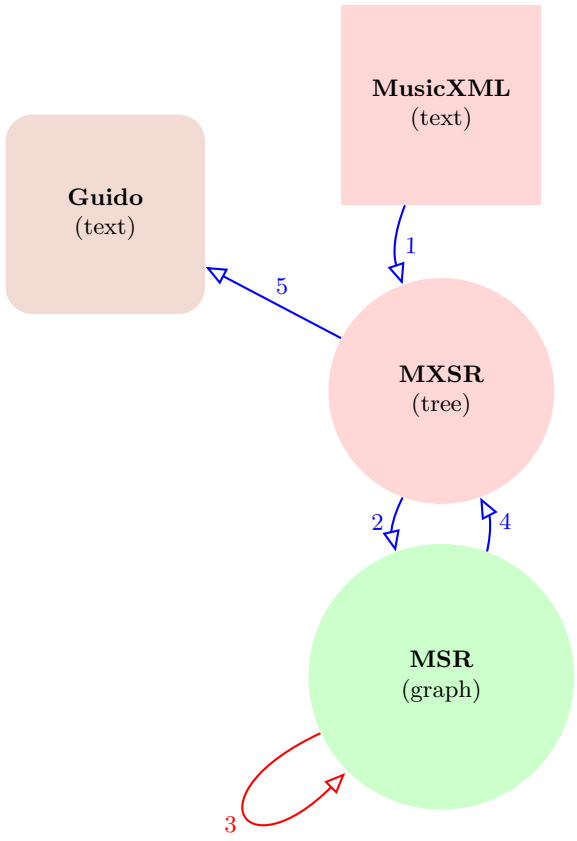
Part XI

xml2gmn

Chapter 25

xml2gmn

Figure 25.1: xml2gmn architecture



25.1 Why xml2gmn?

libmusicxml2 comes with xml2guido, a converter of MusicXML files to Guido syntax, which has some limitations. It is supplied as as sample of the library's use.

xml2gmn has been designed to complement libmusicxml2 features: it provides the same translation as xml2guido, with more options for flexibility.

Work remains to be done in the conversion of MSR to MXSR, but xml2gmn is already fairly complete.

25.2 What xml2gmn does

xml2gmn performs the 5 steps from MusicXML to LilyPond to translate the former into the latter, as shown in figure 22.1, [xmlToLyArchitecture], page 84. Converting from MXSR to MSR is done in two sub-phases for implementation reasons.

The '-about' option to xml2gmn details that somewhat:

```
jacquesmenu@macmini > xml2xml -about
What xml2xml does:

    This multi-pass converter basically performs 6 passes:
        Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
                 and converts it to a MusicXML tree;
        Pass 2a: converts that MusicXML tree into
                 a first Music Score Representation (MSR) skeleton;
        Pass 2b: populates the MSR skeleton from the MusicXML tree
                 to get a full MSR;
        Pass 3:  converts the first MSR into a second MSR, to apply options;
        Pass 4:  converts the second MSR into a second MusicXML tree;
        Pass 5:  converts the second MusicXML tree to MusicXML code
                 and writes it to standard output.

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

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

Part XII

Indexes

Files index

Options

Symbols

MusicXML index

Symbols

Main index

Symbols