

MusicFormats user guide

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

v0.9.62 – March 5, 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++17 and provides a set of music scores representations and converters between various textual music scores formats. Building it only requires a C++17 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 [MusicFormatsUserGuide.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.

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

Minimal score



Contents

I	Preamble	6
1	Acknowledgements	7
2	About this document	8
II	Discovering MusicFormats	9
3	The MusicFormats architecture	10
4	A first example	12
4.1	Raw xml2ly usage	12
4.2	Redirecting the output and error messages to files	13
4.3	The need for options	15
4.4	The passes at work	16
5	More examples	18
5.1	Jianpu output	18
5.2	Braille output	18
5.3	Guido output	19
5.4	MusicXML output	20
6	Multiple files conversion	22
6.1	Using the <code>find</code> shell command	22
6.2	Using a <code>Makefile</code>	22
7	The MusicFormats repository	25
8	Library components	26
8.1	Formats	27
8.2	Representations	28
8.3	Passes	28
8.4	Generators	29
8.5	General use converters	31
8.6	Specific converters	32
8.7	MusicFormats services	33
8.8	Other tools	33
9	Music score view	34
10	Versions numbering	36

III	Shell basics	37
11	Shell basics	38
11.1	Basic shell builtins	38
11.2	Commands	39
11.3	Obtaining help about the shell commands	39
11.4	Multiple commands in a line	40
11.5	Paths	40
11.6	Input/Output redirection and pipes	42
11.7	Functions	42
11.8	MusicFormatsBashDefinitions.bash	42
11.9	Scripts	43
11.10	Multi-line commands	43
11.11	Some useful shell commands	44
11.11.1	for	44
11.11.2	find	44
IV	Installing MusicFormats	46
12	MusicFormats installation modes	47
13	Using a distribution	48
13.1	MacOS™ distribution	48
13.1.1	Security issue in recent MacOS™ versions	49
13.2	Ubuntu distribution	51
13.3	Windows™ distribution	51
14	Full installation.	53
14.1	Cloning the repository.	53
14.2	make and cmake definitions	54
14.3	Building the library on Mac OS™ and Linux-like systems	54
14.4	Installing the library on Mac OS™ and Linux-like systems	56
14.4.1	User specific installation on Mac OS™	56
14.4.2	Global installation on Mac OS™	56
14.5	Building the library on Windows™	56
V	Options and help (OAH)	57
15	Options and help design principles	58
16	Options use.	59
16.1	Options characteristics	60
16.2	The -insider option	61
16.3	Early options	62
17	Options and help introspection.	63
17.1	Restricting help to a given group or subgroup	63
17.2	Querying about options by name	63
17.3	Searching the help for a string	64
17.4	Displaying help about options usage	64
17.5	Displaying a help summary	65

18	Options examples.	66
18.1	Boolean options	66
18.2	Options simple values	66
18.3	Options more complex values	67
18.4	More complex options	68
18.5	Displaying the options values	69
18.6	Displaying MusicFormats internal data	71
19	Including options and arguments from a file	73
19.1	An options and arguments file example	73
19.2	Options values and arguments in included files	74
19.3	Multi-level includes	75
19.4	Multi-level includes overflow	76
19.5	An include file example	77
20	Non-musical options	79
20.1	Timing measurements	79
20.2	Chords structure	80
21	Trace options	82
VI	Warnings and errors (WAE)	84
22	Warnings and errors (WAE)	85
VII	Multiple languages support	86
23	Multiple languages support.	87
VIII	The MusicFormats Scripting Language (MFSL)	89
24	MFSL (MusicFormats Scripting Language	90
24.1	A MusicXML example	91
24.2	A first, minimal MFSL script example	91
24.3	Launching an MFSL script	92
24.4	Interactive use of the MFSL interpreter.	93
24.5	The structure of MFSL scripts	95
24.6	Options to the MFSL interpreter	95
24.7	Options in an MFSL script body	96
24.8	Using choices in an MFSL script	97
24.9	Running the tool for one choice label	100
24.10	Running the tool for all the choice labels	100
24.11	MFSL language details	100
24.12	Error handling in MFSL scripts.	100
24.13	Behind the scenes...	100
IX	xml2ly	101
25	xml2ly	102
25.1	Why xml2ly?	102
25.2	What xml2ly does	102
25.3	Useful options to xml2ly	104

X	xml2brl	105
26	xml2brl	106
26.1	Why xml2brl?	106
26.2	What xml2brl does	107
XI	xml2xml	108
27	xml2xml	109
27.1	Why xml2xml?	109
27.2	What xml2xml does	109
XII	xml2gmn	111
28	xml2gmn	112
28.1	Why xml2gmn?	112
28.2	What xml2gmn does	113
XIII	Indexes	114

List of Figures

3.1	The MusicFormats architecture	10
9.1	The MusicFormats score view	35
24.1	The score created with <code>minimal.mfsl</code>	94
25.1	xml2ly architecture	103
26.1	xml2brl architecture	106
27.1	xml2xml architecture	109
28.1	xml2gmn architecture	112

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 [MusicFormatsMaintenanceGuide.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 [10](#);;
- 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 [MusicFormatsAPIGuide.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 10,, 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, supplying `basic/HelloWorld.xml`:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/HelloWorld.xml
2 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
3 <!DOCTYPE score-partwise PUBLIC
4     "-//Recordare//DTD MusicXML 3.0 Partwise//EN"
5     "http://www.musicxml.org/dtds/partwise.dtd">
6 <score-partwise version="3.0">
7   <work>
8     <work-title>Hello World!</work-title>
9   </work>
10  <!-- A very minimal MusicXML example -->
11  <part-list>
12    <score-part id="P1">
13      <part-name>Music</part-name>
14    </score-part>
15  </part-list>
16  <part id="P1">
17  <!-- =====>
18    <measure number="1">
19  <!-- A very minimal MusicXML example, part P1, measure 1 -->
20    <attributes>
21      <divisions>1</divisions>
22      <key>
23        <fifths>0</fifths>
24      </key>
25      <time>
26        <beats>4</beats>
27        <beat-type>4</beat-type>
28      </time>
29      <clef>
30        <sign>G</sign>
31        <line>2</line>

```

```

32     </clef>
33   </attributes>
34   <!-- A very minimal MusicXML example, part P1, measure 1, before first note -->
35   <note>
36     <pitch>
37       <step>C</step>
38       <octave>4</octave>
39     </pitch>
40     <duration>4</duration>
41     <type>whole</type>
42   </note>
43 </measure>
44 <!-- =====>
45 </part>
46 </score-partwise>

```

4.2 Redirecting the output and error messages to files

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

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

```

1  jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly basic/
   UnknownMaintainerIdentification.xml
2  *** MusicXML warning *** basic/UnknownMaintainerIdentification.xml:11: creator type "
   maintainer" is unknown
3  \version "2.22.0"
4
5  % Pick your choice from the next two lines as needed
6  %myBreak = { \break }
7  myBreak = {}
8
9  % Pick your choice from the next two lines as needed
10 %myPageBreak = { \pageBreak }
11 myPageBreak = {}
12
13 \header {
14   title           = "Hello World!"
15   workTitle       = "Hello World!"
16   title           = "Hello World!"
17 }
18
19 \paper {
20 }
21
22 \layout {
23   \context {
24     \Score
25     autoBeaming = ##f % to display tuplets brackets
26   }
27   \context {
28     \Voice
29   }
30 }
31
32 Part_P0ne_Staff_One_Voice_One = \absolute {
33   \language "nederlands"
34   \key c \major
35   \numericTimeSignature \time 4/4
36
37   \clef "treble"
38   c'1 | % 2

```

```

39   \barNumberCheck #2
40   | % 2
41   \barNumberCheck #2
42 }
43
44 \book {
45   \score {
46     <<
47
48     \new Staff = "Part_POne_Staff_One"
49     \with {
50       }
51     <<
52     \context Voice = "Part_POne_Staff_One_Voice_One" <<
53       \Part_POne_Staff_One_Voice_One
54     >>
55   >>
56
57   >>
58
59   \layout {
60     \context {
61       \Score
62       autoBeaming = ##f % to display tuplets brackets
63     }
64     \context {
65       \Voice
66     }
67   }
68
69   \midi {
70     \tempo 16 = 360
71   }
72 }
73
74 }
75 Warning message(s) were issued for input line 11

```

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

Thus, after executing:

```

1 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:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat error.txt
2 *** MusicXML warning *** basic/UnknownMaintainerIdentification.xml:11: creator type "
  maintainer" is unknown
3 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:

```
1 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`:

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

Using it this way:

```
1 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:

```
1 \version "2.22.0"
2
3 % ... ..
4
5 \header {
6   title = "U. N. Known"
7   % ... ..
8 }
9
10 % ... ..
```

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 powerful 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:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -auto-output-file
  -name -trace-passes -display-cpu-usage basic/Anacrusis.xml
2
3 %-----
4   Handle the options and arguments from argc/argv
5 %-----
6 This is xml2ly v0.9.52 (November 29, 2021) from MusicFormats v0.9.60 (January 25, 2022)
7 Launching the conversion of "basic/Anacrusis.xml" to LilyPond
8 Time is Thursday 2022-02-03 @ 14:19:32 CET
9 The command line is:
10  xml2ly -auto-output-file-name -trace-passes -cpu basic/Anacrusis.xml
11 or with options long names:
12  xml2ly -auto-output-file-name -trace-passes -display-cpu-usage basic/Anacrusis.xml
13 or with options short names:
14  -aofn -tpasses -cpu basic/Anacrusis.xml
15 LilyPond code will be written to Anacrusis.ly
16 The command line options and arguments have been analyzed
17
18 %-----
19   Pass 1: Create an MXSR reading a MusicXML file
20 %-----
21 % MusicXML data uses UTF-8 encoding
22
23 %-----
24   Pass 2a: Create an MSR skeleton from the MXSR
25 %-----
26
27 %-----
28   Pass 2b: Populate the MSR skeleton from MusicXML data
29 %-----
30
31 <!--== part "P1", line 60 ==-->
32
33 %-----
34   Pass 3: Convert the first MSR into a second MSR
35 %-----
36
37 %-----
38   Pass 4: Convert the second MSR into an LPSR
39 %-----
40
41 Opening file 'Anacrusis.ly' for writing
42
43 %-----
44   Pass 5: Convert the LPSR score to LilyPond code
45 %-----
46 Timing information:
47
48 Activity   Description                                     Kind      CPU (sec)
49 -----
50
51           Handle the options and arguments from argc/argv      mandatory  0.02798
52 Pass 1     Create an MXSR reading a MusicXML file                        mandatory  0.00420
53 Pass 2a    Create an MSR skeleton from the MXSR                          mandatory  0.00191
54 Pass 2b    Populate the MSR skeleton from MusicXML data                  mandatory  0.00314
55 Pass 3     Convert the first MSR into a second MSR                       mandatory  0.00069
56 Pass 4     Convert the second MSR into an LPSR                          mandatory  0.00090
57 Pass 5     Convert the LPSR score to LilyPond code                      mandatory  0.00156
58
59 Total (sec)  Mandatory  Optional
60 -----
61 0.04037      0.04037      0.00000

```

The optional passes are those that display MusicFormats internal data. They are triggered by suitable options, see section [18.6](#) [Displaying MusicFormats internal data], page [71](#),.

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

Anacrusis



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 pitches are numbers relative to the scale instead of graphic elements:

```
1 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:

Anacrusis score in Jianpu format

1 = C $\frac{3}{4}$ 55 | 6 5 $\dot{1}$ | $\dot{1}$ - 0 |

This is to be compared with:

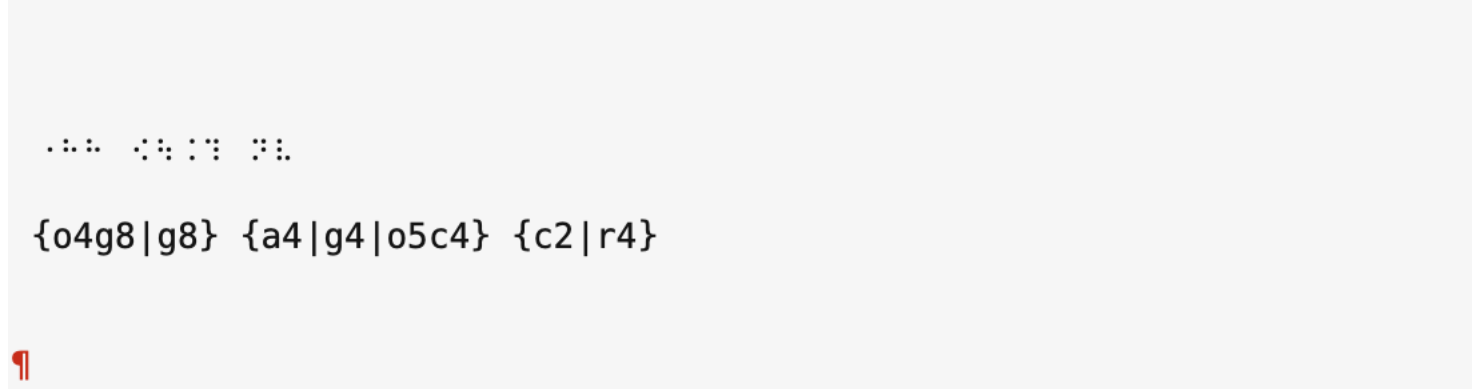


5.2 Braille output

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

```
1 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 `xml2gm` is a multi-pass converter when `xml2guido` a part of `libmusicxml2`, has two passes and only use a MXSR representation.

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2gm basic/Anacrusis.
  xml
2 {[ \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 ]
3 }
```

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

Guido Editor
Save ▾
Examples ▾
Documentation

Editor
Engine Settings
Other Settings

About

```

1  {[ \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 ]
2  }
3

```

Preview

Score
Piano Roll
SPR
Preferences

Guido Project - Copyright 2020 © Grame-CNCM

5.4 MusicXML output

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

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/Anacrusis.xml
2 <?xml version="1.0" encoding="UTF-8"?>
3 <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.0 Partwise//EN" "http://www.
  musicxml.org/dtds/partwise.dtd">
4 <score-partwise>
5   <work>
6     <work-title>Anacrusis</work-title>
7   </work>
8
9   <!-- ... .. />
10
11 </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:

```

1 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:

```

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

```

```

5 <score-partwise version="3.1">
6   <!--
7   =====
8   Generated by xml2xml v0.9.5 (October 6, 2021)
9   on Monday 2022-02-14 @ 08:05:54 CET
10  from "basic/Anacrusis.xml"
11  =====
12  -->
13    <work>
14      <work-number>317</work-number>
15      <work-title>Anacrusis from xml2ml with alto clef</work-title>
16    </work>
17
18    <!-- ... .. . />
19
20    <part id="P1">
21      <measure number="0">
22        <attributes>
23          <divisions>2</divisions>
24          <key>
25            <fifths>0</fifths>
26          </key>
27          <time>
28            <beats>3</beats>
29            <beat-type>4</beat-type>
30          </time>
31          <clef>
32            <sign>C</sign>
33            <line>3</line>
34          </clef>
35        </attributes>
36
37      <!-- ... .. . />
38
39    </score-partwise>

```

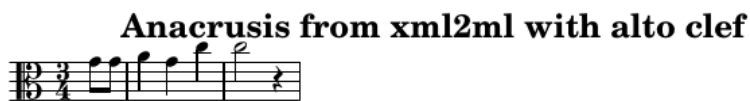
Let's convert this to LilyPond with `xml2ly` into `Anacrusis_From_xml2xml.ly`:

```

1 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 the `find` command.

The `files/musicxmlfiles/clefs` sub-folder initial contents is:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/clefs > ls -sal
2 total 64
3  0 drwxr-xr-x    5 jacquesmenu  staff    160 Feb 26 00:18 .
4  0 drwxr-xr-x  108 jacquesmenu  staff   3456 Feb 25 07:46 ..
5 16 -rw-r--r--@   1 jacquesmenu  staff   6148 Feb 26 00:18 .DS_Store
6 24 -rw-r--r--@   1 jacquesmenu  staff  12286 Jan  2 17:01 ClefChange.xml
7 24 -rw-r--r--@   1 jacquesmenu  staff   9120 Jan  2 17:02 Clefs.xml
```

`xml2ly` can be applied to all the MusicXML files in this folder with:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/clefs > find . -name "*.
2   xml" -exec xml2ly {} -auto-output-file-name \;
```

The contents of the sub-folder now contains the following:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/clefs > ls -sal
2 total 80
3  0 drwxr-xr-x    7 jacquesmenu  staff    224 Feb 26 00:19 .
4  0 drwxr-xr-x  108 jacquesmenu  staff   3456 Feb 25 07:46 ..
5 16 -rw-r--r--@   1 jacquesmenu  staff   6148 Feb 26 00:18 .DS_Store
6  8 -rw-r--r--    1 jacquesmenu  staff   3168 Feb 26 00:19 ClefChange.ly
7 24 -rw-r--r--@   1 jacquesmenu  staff  12286 Jan  2 17:01 ClefChange.xml
8  8 -rw-r--r--    1 jacquesmenu  staff   3045 Feb 26 00:19 Clefs.ly
9 24 -rw-r--r--@   1 jacquesmenu  staff   9120 Jan  2 17:02 Clefs.xml
```

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;

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cd keys/
2 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:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > ln -s ../
  Makefile
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 14:12 .
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

```

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:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > make help
2 Makefile for converting MusicXML files to LilyPond scores.
3 Supplied as part of the MusicFormats library, thanks to Dom Fober for providing it.
4 Output files are written in sub-folders of the current folder.
5
6 Available targets are:
7   'all' (default) : call the 'lily' target.
8
9   'lily'           : converts the set of MusicXML files to LilyPond in folder lilypond
10
11  'pdflylily'       : converts the output of 'lily' target to pdf in folder pdflylily
12  'svglylily'       : converts the output of 'lily' target to svg in folder svglylily
13  'pnglylily'       : converts the output of 'lily' target to png in folder pnglylily
14  'pslylily'        : converts the output of 'lily' target to ps in folder pslylily
15  'epslylily'       : converts the output of 'lily' target to eps in folder epslylily
16
17  'clean'           : removes the sub-folders containing the results
18
19 Make variables:
20   XML2LY=/path/to/xml2ly
21   LILYPOND=/path/to/lilypond
22   When XML2LY and/or LILYPOND are not set,
23   these tools are looked up in the PATH variable,
24
25   OPTIONS=...
26   By default, OPTIONS contains '-q' (quiet mode), for use by xml2ly.

```

The conversion using the 'lily' default target can be achieved by:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > make
2 XML2LY    = xml2ly
3 OPTIONS   = -q
4 LILYPOND   = /Applications/LilyPond/lilypond-2.23.6/bin/lilypond
5
6 xml2ly version is:
7 Command line version of musicxml2lilypond converter v0.9.53 (February 21, 2022)
8
9 lilypond version is:
10 GNU LilyPond 2.23.6 (running Guile 2.2)
11

```



```

12 xml2ly -q -o "lilypond/HumdrumScotKeys.ly" "HumdrumScotKeys.xml"
13 /Applications/LilyPond/lilypond-2.23.6/bin/lilypond --pdf -l NONE -s -o "pdf Lily/
    HumdrumScotKeys" "lilypond/HumdrumScotKeys.ly"
14
15 Contents of lilypond sub-folder:
16 8 -rw-r--r-- 1 jacquesmenu staff 1852 Feb 25 15:42 lilypond/HumdrumScotKeys.ly
17
18 Contents of pdf Lily sub-folder:
19 88 -rw-r--r-- 1 jacquesmenu staff 42795 Feb 25 15:42 pdf Lily/HumdrumScotKeys.pdf

```

The resulting sub-folder contents is:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/keys > ls -salR
2 total 24
3 0 drwxr-xr-x 7 jacquesmenu staff 224 Feb 25 15:42 .
4 0 drwxr-xr-x 108 jacquesmenu staff 3456 Feb 25 07:46 ..
5 16 -rw-r--r--@ 1 jacquesmenu staff 6148 Feb 25 12:51 .DS_Store
6 8 -rw-r--r-- 1 jacquesmenu staff 2634 Apr 22 2021 HumdrumScotKeys.xml
7 0 lrwxr-xr-x 1 jacquesmenu staff 11 Feb 25 14:12 Makefile -> ../Makefile
8 0 drwxr-xr-x 3 jacquesmenu staff 96 Feb 25 15:42 lilypond
9 0 drwxr-xr-x 4 jacquesmenu staff 128 Feb 25 15:42 pdf Lily
10
11 ./lilypond:
12 total 8
13 0 drwxr-xr-x 3 jacquesmenu staff 96 Feb 25 15:42 .
14 0 drwxr-xr-x 7 jacquesmenu staff 224 Feb 25 15:42 ..
15 8 -rw-r--r-- 1 jacquesmenu staff 1852 Feb 25 15:42 HumdrumScotKeys.ly
16
17 ./pdf Lily:
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 or binary form, used in the field of music score applications, thus outside of the library;
- a *representation* is 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:

```
1 jacquesmenu@macmini > xml2xml -about
2 What xml2xml does:
3
4     This multi-pass converter basically performs 6 passes:
5         Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
6                 and converts it to a MusicXML tree;
7         Pass 2a: converts that MusicXML tree into
8                 a first Music Score Representation (MSR) skeleton;
9         Pass 2b: populates the MSR skeleton from the MusicXML tree
10                to get a full MSR;
11         Pass 3:  converts the first MSR into a second MSR, to apply options;
```

```

12      Pass 4:  converts the second MSR into a second MusicXML tree;
13      Pass 5:  converts the second MusicXML tree to MusicXML code
14              and writes it to standard output.
15
16      Other passes are performed according to the options, such as
17      displaying views of the internal data or printing a summary of the score.
18
19      The activity log and warning/error messages go to standard error.

```

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

```

1  jacquesmenu@macmini > Mikrokosmos3Wandering -musicxml -about
2  What Mikrokosmos3Wandering does:
3
4      This multi-pass generator creates a textual representation
5      of Zoltán Kodály's Mikrokosmos III Wandering score.
6      It basically performs 4 passes when generating MusicXML output:
7
8      Pass 1:  generate a first MSR for the Mikrokosmos III Wandering score
9      Pass 2:  converts the first MSR a second MSR, to apply options;
10     Pass 3:  converts the second MSR into an MusicXML tree;
11     Pass 4:  converts the MusicXML tree to MusicXML code
12             and writes it to standard output.
13
14     Other passes are performed according to the options, such as
15     displaying views of the internal data or printing a summary of the score.
16
17     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 containing markups such as <code><part-list/></code> , <code><time/></code> and <code><note/></code> ;
Guido	a text containing markups such as <code>\barFormat</code> , <code>\tempo</code> and <code>\crescEnd</code> ;
LilyPond	a text containing commands such as <code>\header</code> , <code>\override</code> and <code>\transpose</code> ;
Jianpu LilyPond	a text containing LilyPond commands and the use of <code>lilypond-Jianpu</code> (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 <code>lilypondstuff/jianpu</code> ;
Braille	a text containing 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 <code><part-list/></code> , <code><time/></code> and <code><note/></code> ;
LPSR	LilyPond Score Representation, i.e. MSR plus LilyPond-specific items such as <code>\score</code> blocks;
BSR	Braille Score Representation, with pages, lines and 6-dots cells;
MDSR	MIDI Score Representation, to be designed.

8.3 Passes

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

Arrow	Pass name	Description
1	mxm2mxsr	reads MusicXML data from a file or from standard input if '-' is supplied as the file name, and creates an MXSR representation containing the same data;
2	mxsr2mxm	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 an 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 fills this with all the rest;
5	mxsr2lpsr	converts an MSR representation into an LPSR representation, which contains an MSR component built from the original MSR (pass 5). The LPSR contains LilyPond-specific formats such as <code>\layout</code> , <code>\paper</code> , and <code>\score</code> blocks;
6	lpsr2msr	converts an LPSR representation into an MSR representation. There is nothing to do, since the former contains the latter as a component;
7	lpsr2lilypond	converts an LPSR representation into LilyPond text code, and writes it to standard output;
7'	lpsr2lilypond	converts an LPSR representation into LilyPond text code using <code>lilypond-Jianpu</code> , and writes it to standard output. This pass is run with <code>xml2ly -jianpu</code> ;

8	<code>msr2bsr</code>	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	<code>bsr2msr</code>	converts a BSRrepresentation into an MSR representation. There is nothing to do, since the former contains the latter as a component;
10	<code>bsr2bsr</code>	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	<code>bsr2braille</code>	converts a BSRrepresentation into Braille text, and writes it to standard output;
12	<code>msr2mxsr</code>	converts an MSR representation into an MXSR representation;
13	<code>msr2msr</code>	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 it has been scanned and exported as MusicXML data, or apply options;
14	<code>msdl2msr</code>	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:

```

1 jacquesmenu@macmini > MusicAndHarmonies | more
2 <?xml version="1.0" encoding="UTF-8" standalone="no"?>
3 <!DOCTYPE score-partwise PUBLIC "-//Recordare//DTD MusicXML 3.1 Partwise//EN"
4 "http://www.musicxml.org/dtds/partwise.dtd">
5 <score-partwise>
6   <movement-title>Random Music</movement-title>
7   <identification>
8     <creator type="Composer">Georg Chance</creator>
9     <encoding>
10      <software>MusicFormats Library's MusicAndHarmonies generator</software>
11    </encoding>
12  </identification>
13  <part-list>
14    <score-part id="P1">
15      <part-name>Part name</part-name>
16      <score-instrument id="I1">
17        <instrument-name>Any instr.</instrument-name>
18      </score-instrument>

```

```

19     </score-part>
20 </part-list>
21 <part id="P1">
22     <measure number="1">
23         <attributes>
24             <divisions>4</divisions>
25             <time>
26                 <beats>4</beats>
27                 <beat-type>4</beat-type>
28             </time>
29             <clef>
30                 <sign>G</sign>
31                 <line>2</line>
32             </clef>
33         </attributes>
34         <harmony>
35             <root>
36                 <root-step>C</root-step>
37             </root>
38             <kind text="F00">major</kind>
39             <staff>1</staff>
40         </harmony>
41         <note>
42             <pitch>
43                 <step>F</step>
44                 <octave>5</octave>
45             </pitch>
46             <duration>4</duration>
47             <type>quarter</type>
48         </note>
49
50     <!-- ... .. -->

```

- [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:

Mikrokosmos III Wandering

Zoltán Kodály

- [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:

Piano Sonata in A Major

Wolfgang Amadeus Mozart



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 \Rightarrow 3
xml2ly	performs the 4 passes from MusicXML to LilyPond to translate the former into the latter, using these passes: 1 \Rightarrow 4 \Rightarrow 13 \Rightarrow 5 \Rightarrow 7 The <code>-jianpu</code> option is supplied to create Jianpu (numbered) scores, in which the notes are represented by numbers instead of graphics, using passes: 1 \Rightarrow 4 \Rightarrow 13 \Rightarrow 5 \Rightarrow 7'
xml2brl	performs the 5 passes from MusicXML to Braille to translate the former into the latter (draft); 1 \Rightarrow 4 \Rightarrow 13 \Rightarrow 8 \Rightarrow 10 \Rightarrow 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 \Rightarrow 4 \Rightarrow 13 \Rightarrow 12 \Rightarrow 2
xml2gmn	converts MusicXML data to Guido code, using passes: 1 \Rightarrow 4 \Rightarrow 13 \Rightarrow 12 \Rightarrow 3

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

```
1 jacquesmenu@macmini > xml2xml -about
2 What xml2xml does:
3
4 This multi-pass converter basically performs 6 passes:
5   Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
6             and converts it to a MusicXML tree;
7   Pass 2a: converts that MusicXML tree into
8             a first Music Score Representation (MSR) skeleton;
9   Pass 2b: populates the MSR skeleton from the MusicXML tree
10            to get a full MSR;
11   Pass 3:  converts the first MSR into a second MSR, to apply options;
12   Pass 4:  converts the second MSR into a second MusicXML tree;
13   Pass 5:  converts the second MusicXML tree to MusicXML code
14            and writes it to standard output.
15
16 Other passes are performed according to the options, such as
```



```

17 displaying views of the internal data or printing a summary of the score.
18
19 The activity log and warning/error messages go to standard error.

```

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

```

1 jacquesmenu@macmini > Mikrokosmos3Wandering -about
2 What Mikrokosmos3Wandering does:
3
4 This multi-pass generator creates a textual representation
5 of Zoltán Kodály's Mikrokosmos III Wandering score.
6 It performs various passes depending on the output generated,
7 which should be specified a '-lilypond', '-braille', '-musicxml' or '-guido' option.
8
9 Other passes are performed according to the options, such as
10 displaying views of the internal data or printing a summary of the score.
11
12 The activity log and warning/error messages go to standard error.

```

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

```

1 jacquesmenu@macmini > Mikrokosmos3Wandering -braille -about
2 What Mikrokosmos3Wandering does:
3
4 This multi-pass generator creates a textual representation
5 of Zoltán Kodály's Mikrokosmos III Wandering score.
6 It basically performs 4 passes when generating braille output:
7
8 Pass 1: generate a first MSR for the Mikrokosmos III Wandering score
9 Pass 2: converts the first MSR a second MSR, to apply options;
10 Pass 3: converts the second MSR into a
11 Braille Score Representation (BSR)
12 containing one Braille page per MusicXML page;
13 Pass 4: converts the BSR into another BSR
14 with as many Braille pages as needed
15 to fit the line and page lengths;
16 Pass 5: converts the BSR to Braille text
17 and writes it to standard output.)
18
19 In this preliminary version, pass 3 merely clones the BSR it receives.
20
21 Other passes are performed according to the options, such as
22 displaying views of the internal data or printing a summary of the score.
23
24 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
------------	-------------

<code>msdlconverter -lilypond</code>	performs the 4 passes from MusicXML to LilyPond to translate the former into the latter, using these passes: $1 \Rightarrow 4 \Rightarrow 13 \Rightarrow 5 \Rightarrow 7$ The <code>-jianpu</code> option is supplied to create Jianpu (numbered) scores, in which the notes are represented by numbers instead of graphics, using passes: $1 \Rightarrow 4 \Rightarrow 13 \Rightarrow 5 \Rightarrow 7'$
<code>msdlconverter -braille</code>	performs the 5 passes from MusicXML to Braille to translate the former into the latter (draft); $1 \Rightarrow 4 \Rightarrow 13 \Rightarrow 8 \Rightarrow 10 \Rightarrow 11$
<code>msdlconverter -musicxml</code>	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: $1 \Rightarrow 4 \Rightarrow 13 \Rightarrow 12 \Rightarrow 2$
<code>msdlconverter -guido</code>	converts MusicXML data to Guido code, using passes: $1 \Rightarrow 4 \Rightarrow 13 \Rightarrow 12 \Rightarrow 3$

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.

Chapter 9

Music score view

It may be useful for MusicFormats users to have an insight of what is behind the scenes. The picture at figure 9.1 [The MusicFormats score view], page 35,, shows how it views a music score, i.e. how it represents it in MSR.

The background colors are used as follows:

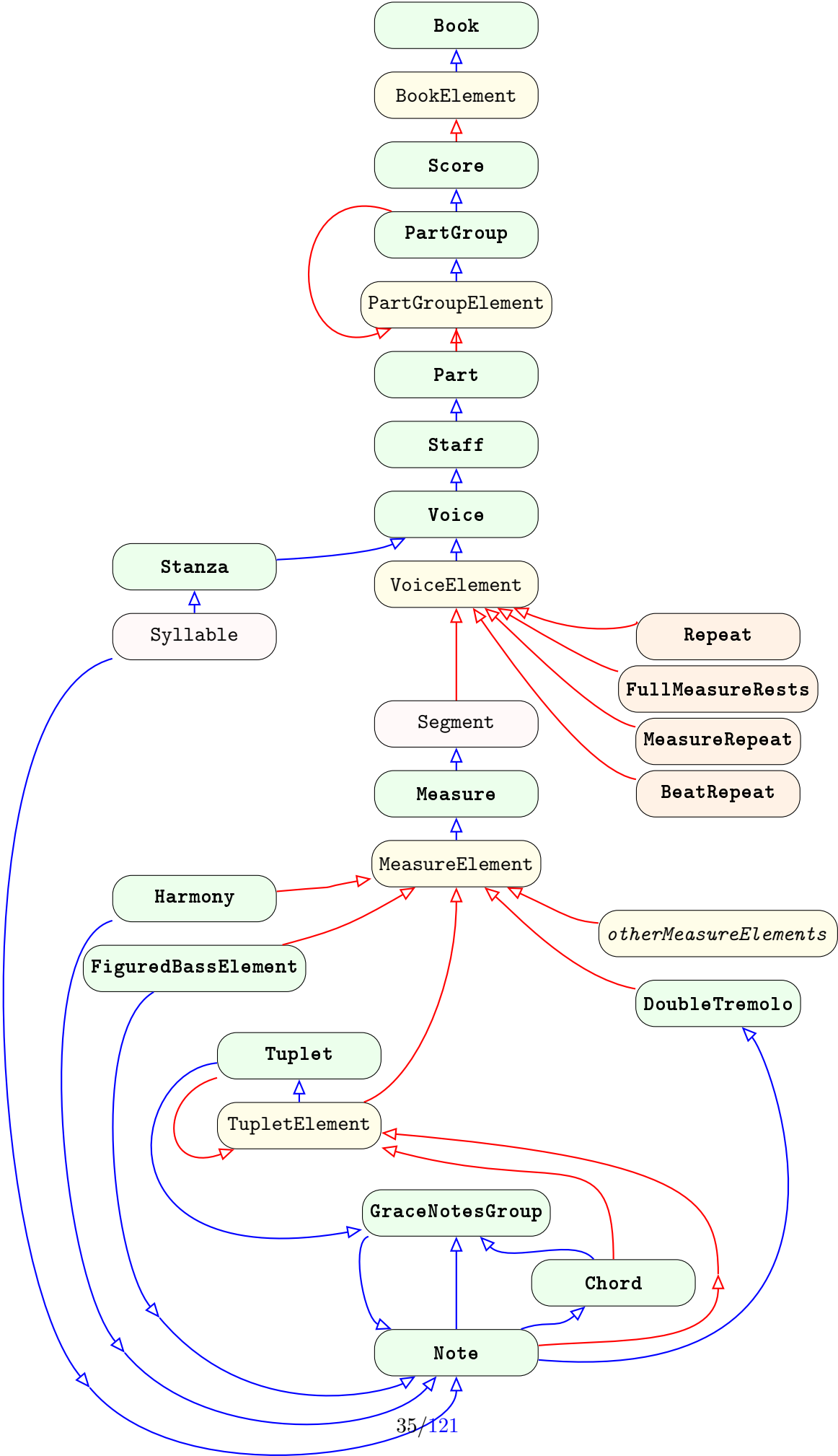
- **green**: an element that is expected to be found in a score representation, such as Staff and Chord;
- **pink**: a element needed internally in MSR to structure the representation, such as Segment and Syllable. Those are not known by musicians, and specific to MusicFormats;
- **yellow**: elements that can be used in other ones, such as a Voice containing VoiceElements. Those are specific to MusicFormats too.

The arrows colors have the following meaning:

- **blue**: can be contained in. For example, an Tuplet can be contained in a GraceNotesGroup, which can be contained in (attached to) a Note.
- **red**: a particular case of. For example, Part is a particular case of PartGroupElement, and a PartGroup is a particular case of PartGroupElement, meaning that part groups can be *nested*.

This view is inspired by both MusicXML and LilyPond. The specific MusicFormats specific aspects are there due to implementation considerations.

Figure 9.1: The MusicFormats score view



Chapter 10

Versions numbering

MusicFormats uses *semantic* version numbering, such as `v0.9.61`:

- the library itself gets a new number right after a new branch as been created for it. Branching to "v0.9.60" causes the library to be numbered "v0.9.61" with `SetMusicFormatsVersionNumber.bash`;
- each representation, converter or pass that is modified this new branch has been created gets a new history element with the same number as the library.

Thus all the executable tools have the same version number as the library, but their components may have older version numbers. This can be seen with options `-mf-version`, `-mfv`, `-version`, `-v` and `-version-full`, `-vf` :

```

1 MusicFormats library v0.9.61 (February 28, 2022)
2
3 jacquesmenu@macmini > xml2ly -version
4 Command line version of musicxml2lilypond converter v0.9.61 (February 28, 2022)
5
6 jacquesmenu@macmini > xml2ly -version-full
7 Command line version of musicxml2lilypond converter v0.9.61 (February 28, 2022)
8
9 Representations versions:
10  MXSR
11    v0.9.50 (October 6, 2021)
12  MSR
13    v0.9.52 (November 27, 2021)
14  LPSR
15    v0.9.50 (October 6, 2021)
16
17 Passes versions:
18  mxsr2msr
19    v0.9.60 (February 21, 2022)
20  msr2msr
21    v0.9.51 (November 15, 2021)
22  msr2lpsr
23    v0.9.50 (October 6, 2021)
24  lpsr2lilypond
25    v0.9.60 (February 21, 2022)

```

Part III

Shell basics

Chapter 11

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 *terminals*, 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.

A shell maintains a so-called *current working directory*, which a directory in the files system.

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:

```
1 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:

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

11.1 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.

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

```

1 jacquesmenu@macmini: ~ > pwd
2 /Users/jacquesmenu
3
4 jacquesmenu@macmini: ~ > cd musicformats-git-dev
5
6 jacquesmenu@macmini: ~/musicformats-git-dev > pwd
7 /Users/jacquesmenu/musicformats-git-dev
8 jacquesmenu@macmini: ~/musicformats-git-dev >

```

11.2 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`:

```

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

```

The shell can be queried about a command name:

```

1 jacquesmenu@macmini > type cd
2 cd is a shell builtin
3
4 jacquesmenu@macmini > type xml2lyy
5 -bash: type: xml2lyy: not found
6
7 jacquesmenu@macmini > type xml2ly
8 xml2ly is hashed (/Users/jacquesmenu/musicformats-git-dev/build/bin/xml2ly)

```

11.3 Obtaining help about the shell commands

The `man` command (Manual page) can be used for that:

```

1 jacquesmenu@macmini > man
2 What manual page do you want?

```

It can be used to obtain help about itself, too:

```

1 jacquesmenu@macmini > man man

```

produces:

```

1 man(1)                                     General Commands Manual
2
3                                     man(1)
4
5 NAME
6     man - format and display the on-line manual pages
7
8 SYNOPSIS
9     man [-acdfFhkKtwW] [--path] [-m system] [-p string] [-C config_file] [-M pathlist]
10    [-P pager] [-B browser] [-H
    htmlpager] [-S section_list] [section] name ...

```



```

11      man formats and displays the on-line manual pages.  If you specify section, man
12      only looks in that section of
13      the manual.  name is normally the name of the manual page, which is typically the
14      name of a command, function,
15      or file.  However, if name contains a slash (/) then man interprets it as a file
16      specification, so that you can
17      do man ./foo.5 or even man /cd/foo/bar.1.gz.
18
19      See below for a description of where man looks for the manual page files.
20
21      ... ..

```

11.4 Multiple commands in a line

One can submit several commands in a single line, separating them with a `;`:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cd Clefs; ls -sal
2 total 88
3  0 drwxr-xr-x   8 jacquesmenu  staff    256 Feb 26 09:03 .
4  0 drwxr-xr-x 108 jacquesmenu  staff   3456 Feb 25 07:46 ..
5 16 -rw-r--r--@   1 jacquesmenu  staff   6148 Feb 26 00:23 .DS_Store
6  8 -rw-r--r--   1 jacquesmenu  staff   3168 Feb 26 00:19 ClefChange.ly
7 24 -rw-r--r--@   1 jacquesmenu  staff  12286 Jan  2 17:01 ClefChange.xml
8  8 -rw-r--r--   1 jacquesmenu  staff   1662 Apr 22 2021 ClefChanges.xml
9  8 -rw-r--r--   1 jacquesmenu  staff   3045 Feb 26 00:19 Clefs.ly
10 24 -rw-r--r--@   1 jacquesmenu  staff   9120 Jan  2 17:02 Clefs.xml
11 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/Clefs >

```

This is handy for interactive use, which avoids the use of multi-line commands, presented at section 11.10 [Multi-line commands], page 43,.

11.5 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.
- `.` is the current working directory;
- `..` is the directory containing the current working directory, i.e. one level up in the directories hierarchy;

This document uses Unixlike `.%-----` 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:

```

1 jacquesmenu@macmini > xml2ly Nice file.xml
2 Several input file names supplied, only one can be used
3 The arguments vector contains 2 elements:
4   0: "Nice"
5   1: "file.xml"
6
7 jacquesmenu@macmini > xml2ly 'Nice file.xml'
8 can't open file Nice file.xml
9 ### Conversion from MusicXML to LilyPond failed ###
10
11 jacquesmenu@macmini > xml2ly "Nice file.xml"
12 can't open file Nice file.xml
13 ### 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:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -find "tuplet's"
2 0 occurrence of string "tuplet's" has been found
3
4 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles >
5 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -find 'tuplet"s'
6 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:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > DOC_DIR=documentation
2
3 jacquesmenu@macmini: ~/musicformats-git-dev > echo $DOC_DIR
4 documentation

```

Variables can be used surrounded by curly brackets, too:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/documentation > echo ${DOC_DIR}
2 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:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > ls $DOC_DIR
2 CommonLaTeXFiles      MusicFormatsUserGuide presentation
3 IntroductionToMusicXML MusicFormatsMaintainanceGuide
4 MusicFormatsAPIGuide   graphics
5
6 jacquesmenu@macmini: ~/musicformats-git-dev > cd $DOC_DIR
7
8 jacquesmenu@macmini: ~/musicformats-git-dev/documentation > pwd
9 /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.

```

1 jacquesmenu@macmini: ~/musicformats-git-dev > DOC_DIR=documentation
2
3 jacquesmenu@macmini: ~/musicformats-git-dev > cd '$DOC_DIR'
4 -bash: cd: $DOC_DIR: No such file or directory
5
6 jacquesmenu@macmini: ~/musicformats-git-dev > cd "$DOC_DIR"
7
8 jacquesmenu@macmini: ~/musicformats-git-dev/documentation > pwd
9 /Users/jacquesmenu/musicformats-git-dev/documentation

```

Here is an example combining quotes and double quotes:

```

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

```

11.6 Input/Output redirection and pipes

Shells can read commands from text files named *scripts* through input redirection, output redirection and pipes.

11.7 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:

```

1 function checkVersions ()
2 {
3     # set -x
4
5     xml2ly -v
6     xml2brl -v
7     xml2xml -v
8     xml2gmn -v
9
10    Mikrokosmos3Wandering -v
11
12    msdlconverter -v
13    # set +x
14 }

```

11.8 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:

```

1 jacquesmenu@macmini > type ll
2 ll is a function
3 ll ()
4 {
5     ls -salGTF $*
6 }

```

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

11.9 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:

```

1 jacquesmenu@macmini > cat doBethena_SaxTenor.bash
2 #!/bin/bash
3
4 cp -p Bethena_SaxTenor_original.xml Bethena_SaxTenor.xml
5
6 xml2ly -include Bethena_SaxTenor_OptionsAndArguments.txt
7
8 open Bethena_SaxTenor.ly

```

After creating the script file, make it executable:

```

1 jacquesmenu@macmini > chmod +x doBethena_SaxTenor.bash
2
3 jacquesmenu@macmini > ls -sal doBethena_SaxTenor.bash
4 8 -rwxr-xr-x@ 1 jacquesmenu staff 154 Feb 20 07:46 doBethena_SaxTenor.bash

```

The script can then be executed with:

```

1 jacquesmenu@macmini > ./doBethena_SaxTenor.bash

```

11.10 Multi-line commands

A command can be split into several lines if it is long or to enhance readability. A `\` character is used for that:

```

1 jacquesmenu@macmini > xml2brl \
2 > -trace-passes \
3 > foo.xml
4
5 %-----
6     Handle the options and arguments from argc/argv
7 %-----
8 This is xml2brl v0.9.51 (October 12, 2021) from MusicFormats v0.9.61 (February 18, 2022)
9 Launching the conversion of "foo.xml" to Braille
10 Time is Saturday 2022-02-26 @ 09:53:43 CET
11 The command line is:
12     xml2brl -trace-passes foo.xml
13 or with options long names:
14     xml2brl -trace-passes foo.xml
15 or with options short names:

```

```

16 -tpasses foo.xml
17 Braille will be written to standard output
18 The command line options and arguments have been analyzed
19
20 %-----
21 Pass 1: Create an MXSR reading a MusicXML file
22 %-----
23 can't open file foo.xml
24 ### Conversion from MusicXML to Braille failed ###

```

This feature is most useful in scripts.

11.11 Some useful shell commands

11.11.1 for

The `for` command is analogous to the same in programming languages, with features specific to shell usage.

For example, MusicFormats comes with some MusicXML files whose name contains 'Stanza':

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/lyrics > for FILE in $(ls
   *Stanza*.xml); do echo ${FILE}; done
2 MultipleStanzas.xml
3 StanzaMaster.xml

```

They can be converted to LilyPond with this `for` command. Note the use of `$(ls *Stanza*.xml)` to execute the `ls` command to obtain the files names to be handled:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/lyrics > for FILE in $(ls
   *Stanza*.xml); do echo "---> converting ${FILE} to LilyPond"; xml2ly -auto-output-file
   -name ${FILE}; done
2 ---> converting MultipleStanzas.xml to LilyPond
3 ---> converting StanzaMaster.xml to LilyPond

```

The resulting files are:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/lyrics > ls -sal *Stanza*.
   ly
2 8 -rw-r--r--  1 jacquesmenu  staff   3659 Feb 26 10:38 MultipleStanzas.ly
3 8 -rw-r--r--  1 jacquesmenu  staff   2236 Feb 26 10:38 StanzaMaster.ly

```

Such long `for` commands are not easy to type as a single line interactively.

11.11.2 find

The `find` command looks for files in directory satisfying some criterion. One of them is a name pattern:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > find . -name *Mozart*.
   xml
2 ./xmlsamples3.1/MozartTrio.xml
3 ./challenging/MozartPianoSonata.xml
4 ./multistaff/MozartAnChloeK524Sample.xml
5 ./multistaff/MozartAdagioKV410.xml
6 ./multistaff/MozartSecondTrio.xml

```

find applies actions to the files that mach the criterion, **-print** by default. There are other actions, such as **-exec**.

Each file found in turn is designated by '{}', and the command must be terminated by '\;':

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > find . -name *Mozart*.
  xml -exec ls -sal {} \;
2 184 -rw-r--r--  1 jacquesmenu  staff   92127 Apr 22   2021 ./xmlsamples3.1/MozartTrio.xml
3 56  -rw-r--r--@ 1 jacquesmenu  staff   25664 Apr 22   2021 ./challenging/MozartPianoSonata.
  xml
4 152 -rw-r--r--  1 jacquesmenu  staff   76231 Apr 22   2021 ./multistaff/
  MozartAnChloeK524Sample.xml
5 320 -rw-r--r--  1 jacquesmenu  staff   163503 Apr 22   2021 ./multistaff/MozartAdagioKV410.
  xml
6 120 -rw-r--r--  1 jacquesmenu  staff   60833 Apr 22   2021 ./multistaff/MozartSecondTrio.xml

```

Note that, as can be seen above, **find** searches the whose set of files contained in its first argument, including in sub-directories.

More informations is available with **man find**.

Part IV

Installing MusicFormats

Chapter 12

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 13

Using a distribution

MusicFormats comes with distributions for Mac OS™, Ubuntu and Windows™ in the form of Zip files., including the user documentation PDF file. `MusicFormatsVersionNumber.txt` contains the library's version number:

```
1 jacquesmenu@macmini: ~/musicformats_local_clone/distrib > ls -sal
2 total 150088
3      0 drwxr-xr-x   8 jacquesmenu  staff      256 Feb 16 07:47 .
4      0 drwxr-xr-x  22 jacquesmenu  staff      704 Feb 16 07:47 ..
5    1672 -rw-r--r--   1 jacquesmenu  staff    854294 Feb 16 07:47 IntroductionToMusicXML.pdf
6    1976 -rw-r--r--   1 jacquesmenu  staff    1008702 Feb 16 07:47 MusicFormatsUserGuide.pdf
7 108168 -rw-r--r--   1 jacquesmenu  staff    55378423 Feb 16 07:47 MusicFormatsForMacOS.zip
8   34080 -rw-r--r--   1 jacquesmenu  staff    17445663 Feb 16 07:47 MusicFormatsForUbuntu.zip
9    4184 -rw-r--r--   1 jacquesmenu  staff    2139537 Feb 16 07:47 MusicFormatsForWindows.zip
10      8 -rw-r--r--   1 jacquesmenu  staff         6 Feb 16 07:47 MusicFormatsVersionNumber.txt
```

The distributions Zip archives are supplied with all versions, i.e. the current, most recent version of MusicFormats(the default branch in the repository), and the earlier versions such as the v0.9.60 branch.

13.1 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:

```
1 jacquesmenu@macmini: ~/Downloads/MusicFormatsForMacOS > ls -sal *
2 8 -rw-r--r--@ 1 jacquesmenu  staff    6 Feb 14 14:20 MusicFormatsVersionNumber.txt
3
4 bin:
5 total 661992
6      0 drwxr-xr-x@ 25 jacquesmenu  staff      800 Feb 14 14:20 .
7      0 drwxr-xr-x@  4 jacquesmenu  staff     128 Feb 15 17:23 ..
8 74864 -rwxr-xr-x@  1 jacquesmenu  staff   38326752 Feb 14 14:20 LilyPondIssue34
9 74864 -rwxr-xr-x@  1 jacquesmenu  staff   38329824 Feb 14 14:20 Mikrokosmos3Wandering
10 8432 -rwxr-xr-x@  1 jacquesmenu  staff    4314896 Feb 14 14:20 MusicAndHarmonies
11 8432 -rwxr-xr-x@  1 jacquesmenu  staff    4314880 Feb 14 14:20 RandomChords
12 8432 -rwxr-xr-x@  1 jacquesmenu  staff    4314880 Feb 14 14:20 RandomMusic
13 8624 -rwxr-xr-x@  1 jacquesmenu  staff    4414944 Feb 14 14:20 countnotes
14 16528 -rwxr-xr-x@  1 jacquesmenu  staff    8459424 Feb 14 14:20 displayMusicformatsHistory
15 16528 -rwxr-xr-x@  1 jacquesmenu  staff    8459424 Feb 14 14:20 displayMusicformatsVersion
16 79200 -rwxr-xr-x@  1 jacquesmenu  staff   40546384 Feb 14 14:20 msdlconverter
17 12480 -rwxr-xr-x@  1 jacquesmenu  staff    6387232 Feb 14 14:20 partsummary
```

```

18 8848 -rwxr-xr-x@ 1 jacquesmenu staff 4528736 Feb 14 14:20 readunrolled
19 64000 -rwxr-xr-x@ 1 jacquesmenu staff 32764496 Feb 14 14:20 xml2brl
20 66872 -rwxr-xr-x@ 1 jacquesmenu staff 34236240 Feb 14 14:20 xml2gmn
21 17160 -rwxr-xr-x@ 1 jacquesmenu staff 8781984 Feb 14 14:20 xml2guido
22 67552 -rwxr-xr-x@ 1 jacquesmenu staff 34583840 Feb 14 14:20 xml2ly
23 12392 -rwxr-xr-x@ 1 jacquesmenu staff 6342528 Feb 14 14:20 xml2midi
24 59720 -rwxr-xr-x@ 1 jacquesmenu staff 30574528 Feb 14 14:20 xml2xml
25 9104 -rwxr-xr-x@ 1 jacquesmenu staff 4657200 Feb 14 14:20 xmlclone
26 9256 -rwxr-xr-x@ 1 jacquesmenu staff 4735296 Feb 14 14:20 xmlfactory
27 8800 -rwxr-xr-x@ 1 jacquesmenu staff 4504976 Feb 14 14:20 xmliter
28 8680 -rwxr-xr-x@ 1 jacquesmenu staff 4442496 Feb 14 14:20 xmlread
29 11976 -rwxr-xr-x@ 1 jacquesmenu staff 6129744 Feb 14 14:20 xmltranspose
30 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, they are placed in the `/Applications` directory.

13.1.1 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.

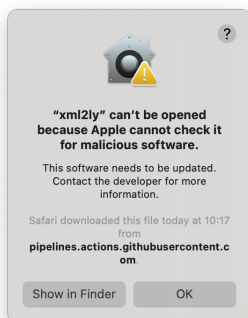
From version 11 (Catalina) on, though, the executables you get are not executable actually, because their developer is unknown to the operating system, and actions have to be taken for them to be usable.

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:

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

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



Clicking in either buttons in this dialog kill the process:

```
1 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:

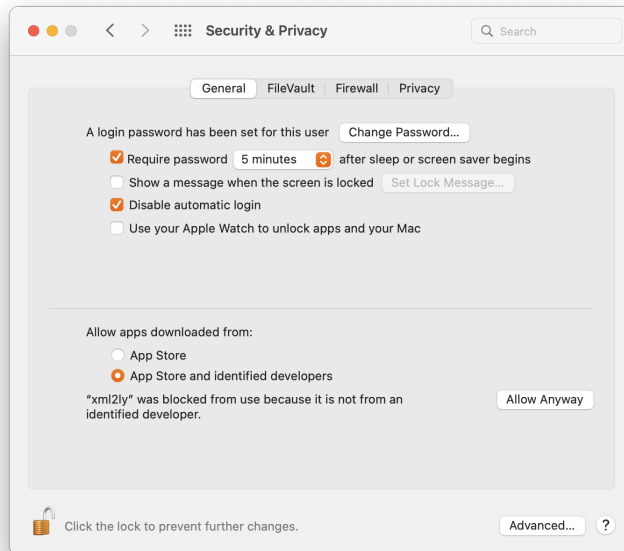
```
1 jacquesmenu@macmini: ~/Downloads/MusicFormatsForMacOS/bin > xattr -d com.apple.quarantine *
```

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 performed again when installing new versions...

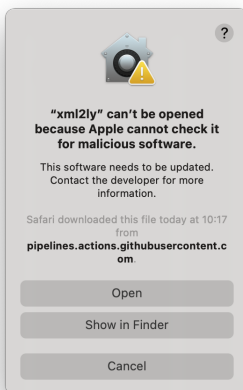
The above can be done in the GUI file by file too. Right after you got the message above:

- 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.

13.2 Ubuntu distribution

After downloading, we get:

```

1 jacquesmenu@macmini: ~/Downloads/MusicFormatsForUbuntu > ls -sal *
2 8 -rw-r--r--@ 1 jacquesmenu staff 6 Feb 14 14:33 MusicFormatsVersionNumber.txt
3
4 bin:
5 total 2296
6 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Feb 14 18:22 .
7 0 drwxr-xr-x@ 6 jacquesmenu staff 192 Feb 16 08:45 ..
8 96 -rw-r--r--@ 1 jacquesmenu staff 49000 Feb 14 14:33 LilyPondIssue34
9 96 -rw-r--r--@ 1 jacquesmenu staff 49040 Feb 14 14:33 Mikrokosmos3Wandering
10 96 -rw-r--r--@ 1 jacquesmenu staff 47224 Feb 14 14:33 MusicAndHarmonies
11 96 -rw-r--r--@ 1 jacquesmenu staff 47216 Feb 14 14:33 RandomChords
12 96 -rw-r--r--@ 1 jacquesmenu staff 47216 Feb 14 14:33 RandomMusic
13 72 -rw-r--r--@ 1 jacquesmenu staff 33800 Feb 14 14:33 countnotes
14 40 -rw-r--r--@ 1 jacquesmenu staff 17648 Feb 14 14:33 displayMusicformatsHistory
15 40 -rw-r--r--@ 1 jacquesmenu staff 17648 Feb 14 14:33 displayMusicformatsVersion
16 104 -rw-r--r--@ 1 jacquesmenu staff 50616 Feb 14 14:33 msdlconverter
17 544 -rw-r--r--@ 1 jacquesmenu staff 275976 Feb 14 14:33 partsummary
18 88 -rw-r--r--@ 1 jacquesmenu staff 43720 Feb 14 14:33 readunrolled
19 80 -rw-r--r--@ 1 jacquesmenu staff 39200 Feb 14 14:33 xml2brl
20 88 -rw-r--r--@ 1 jacquesmenu staff 43336 Feb 14 14:33 xml2gmn
21 48 -rw-r--r--@ 1 jacquesmenu staff 23112 Feb 14 14:33 xml2guido
22 80 -rw-r--r--@ 1 jacquesmenu staff 39056 Feb 14 14:33 xml2ly
23 88 -rw-r--r--@ 1 jacquesmenu staff 42880 Feb 14 14:33 xml2midi
24 88 -rw-r--r--@ 1 jacquesmenu staff 43344 Feb 14 14:33 xml2xml
25 88 -rw-r--r--@ 1 jacquesmenu staff 43368 Feb 14 14:33 xmlclone
26 48 -rw-r--r--@ 1 jacquesmenu staff 22616 Feb 14 14:33 xmlfactory
27 168 -rw-r--r--@ 1 jacquesmenu staff 83488 Feb 14 14:33 xmliter
28 56 -rw-r--r--@ 1 jacquesmenu staff 28424 Feb 14 14:33 xmlread
29 56 -rw-r--r--@ 1 jacquesmenu staff 28656 Feb 14 14:33 xmltranspose
30 40 -rw-r--r--@ 1 jacquesmenu staff 17360 Feb 14 14:33 xmlversion
31
32 lib:
33 total 157792
34 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Feb 14 18:22 .
35 0 drwxr-xr-x@ 6 jacquesmenu staff 192 Feb 16 08:45 ..
36 113224 -rw-r--r--@ 1 jacquesmenu staff 57968176 Feb 14 14:33 libmusicxml2.a
37 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.

13.3 Windows™ distribution

After downloading, we get:

```

1 jacquesmenu@macmini: ~/Downloads/MusicFormatsForWindows > ls -sal *
2 8 -rw-r--r--@ 1 jacquesmenu staff 6 Feb 14 14:53 MusicFormatsVersionNumber.txt
3
4 bin:
5 total 1232
6 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Feb 14 18:22 .
7 0 drwxr-xr-x@ 6 jacquesmenu staff 192 Feb 16 08:49 ..
8 80 -rw-r--r--@ 1 jacquesmenu staff 38400 Feb 14 14:53 LilyPondIssue34.exe
9 80 -rw-r--r--@ 1 jacquesmenu staff 38400 Feb 14 14:53 Mikrokosmos3Wandering.exe
10 56 -rw-r--r--@ 1 jacquesmenu staff 26112 Feb 14 14:53 MusicAndHarmonies.exe
11 56 -rw-r--r--@ 1 jacquesmenu staff 25088 Feb 14 14:53 RandomChords.exe
12 56 -rw-r--r--@ 1 jacquesmenu staff 25088 Feb 14 14:53 RandomMusic.exe
13 32 -rw-r--r--@ 1 jacquesmenu staff 14848 Feb 14 14:53 countnotes.exe
14 24 -rw-r--r--@ 1 jacquesmenu staff 10752 Feb 14 14:53 displayMusicformatsHistory.exe

```

```

15 24 -rw-r--r--@ 1 jacquesmenu staff 10752 Feb 14 14:53 displayMusicformatsVersion.exe
16 80 -rw-r--r--@ 1 jacquesmenu staff 39936 Feb 14 14:53 msdlconverter.exe
17 112 -rw-r--r--@ 1 jacquesmenu staff 56832 Feb 14 14:53 partsummary.exe
18 40 -rw-r--r--@ 1 jacquesmenu staff 18432 Feb 14 14:53 readunrolled.exe
19 64 -rw-r--r--@ 1 jacquesmenu staff 32768 Feb 14 14:53 xml2brl.exe
20 72 -rw-r--r--@ 1 jacquesmenu staff 33280 Feb 14 14:53 xml2gmn.exe
21 64 -rw-r--r--@ 1 jacquesmenu staff 29184 Feb 14 14:53 xml2guido.exe
22 64 -rw-r--r--@ 1 jacquesmenu staff 32768 Feb 14 14:53 xml2ly.exe
23 40 -rw-r--r--@ 1 jacquesmenu staff 17920 Feb 14 14:53 xml2midi.exe
24 72 -rw-r--r--@ 1 jacquesmenu staff 33280 Feb 14 14:53 xml2xml.exe
25 32 -rw-r--r--@ 1 jacquesmenu staff 14848 Feb 14 14:53 xmlclone.exe
26 32 -rw-r--r--@ 1 jacquesmenu staff 15360 Feb 14 14:53 xmlfactory.exe
27 40 -rw-r--r--@ 1 jacquesmenu staff 19456 Feb 14 14:53 xmliter.exe
28 56 -rw-r--r--@ 1 jacquesmenu staff 27136 Feb 14 14:53 xmlread.exe
29 32 -rw-r--r--@ 1 jacquesmenu staff 14848 Feb 14 14:53 xmltranspose.exe
30 24 -rw-r--r--@ 1 jacquesmenu staff 12288 Feb 14 14:53 xmlversion.exe
31
32 lib:
33 total 37368
34 0 drwxr-xr-x@ 4 jacquesmenu staff 128 Feb 14 18:22 .
35 0 drwxr-xr-x@ 6 jacquesmenu staff 192 Feb 16 08:49 ..
36 14696 -rw-r--r--@ 1 jacquesmenu staff 7521356 Feb 14 14:53 musicxml2.exp
37 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 14

Full installation

14.1 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`:

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

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

```
1 jacquesmenu@macmini: ~ > VERSION_NUMBER=v0.9.60
2
3 jacquesmenu@macmini: ~ > git clone -b ${VERSION_NUMBER} https://github.com/jacques-menu/
   musicformats musicformats_local_clone_${VERSION_NUMBER}
```

The local clone contains:

```
1 jacquesmenu@macmini: ~ > cd musicformats_local_clone
2 jacquesmenu@macmini: ~/musicformats_local_clone > ls -sal
3 total 96
4 0 drwxr-xr-x 22 jacquesmenu staff 704 Feb 2 17:26 .
5 0 drwxr-xr-x+ 80 jacquesmenu staff 2560 Feb 2 17:26 ..
6 0 drwxr-xr-x 12 jacquesmenu staff 384 Feb 2 17:26 .git
7 0 drwxr-xr-x 3 jacquesmenu staff 96 Feb 2 17:26 .github
8 8 -rwxr-xr-x 1 jacquesmenu staff 1050 Feb 2 17:26 Build_libmusicformats.bash
9 0 drwxr-xr-x 3 jacquesmenu staff 96 Feb 2 17:26 KEEP
10 40 -rw-r--r-- 1 jacquesmenu staff 16725 Feb 2 17:26 LICENSE
11 8 -rwxr-xr-x 1 jacquesmenu staff 1055 Feb 2 17:26 README.md
12 0 drwxr-xr-x 9 jacquesmenu staff 288 Feb 2 17:26 build
13 0 drwxr-xr-x 10 jacquesmenu staff 320 Feb 2 17:26 docs
14 0 drwxr-xr-x 9 jacquesmenu staff 288 Feb 2 17:26 documentation
15 0 drwxr-xr-x 6 jacquesmenu staff 192 Feb 2 17:26 files
16 0 drwxr-xr-x 5 jacquesmenu staff 160 Feb 2 17:26 javascript
17 0 drwxr-xr-x 21 jacquesmenu staff 672 Feb 2 17:26 libmusicxml
18 0 drwxr-xr-x 10 jacquesmenu staff 320 Feb 2 17:26 midisharelight
19 40 -rw-r--r-- 1 jacquesmenu staff 18502 Feb 2 17:26 musicFormatsBashDefinitions.bash
20 0 drwxr-xr-x 6 jacquesmenu staff 192 Feb 2 17:26 packages
21 0 drwxr-xr-x 8 jacquesmenu staff 256 Feb 2 17:26 schemas
```

```

22 0 drwxr-xr-x 12 jacquesmenu staff 384 Feb 2 17:26 src
23 0 drwxr-xr-x 7 jacquesmenu staff 224 Feb 2 17:26 validation
24 0 drwxr-xr-x 11 jacquesmenu staff 352 Feb 2 17:26 web
25 0 drwxr-xr-x 4 jacquesmenu staff 128 Feb 2 17:26 win32

```

14.2 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:

```

1 jacquesmenu@macmini: ~/musicformats_local_clone/build > make help
2 MusicFormats makefile - Targets are :
3   all (default): build the MusicFormats library for the current platform,
4                   build the MusicFormats tools,
5
6 Platform targets to build the MusicFormats library are:
7   macos          build the library for macos
8   windows        build 32 and 64 bits library for windows
9   linux          build the library for linux
10  android         build a static library for Android
11  ios             build a static library for iOS
12  msys            build on Windows using MSys
13  js              build a javascript library
14 the platform targets is automatically evaluated by the default target.
15
16 Misc:
17   cmake          re-generates the cmake project
18   format         source code formatting using clang-format
19   install        install library, tools and headers
20   localinstall   install the tools to ~/bin
21   package        create the musicformats-v0.9.60 package
22
23 Options:
24   CMAKEOPT       cmake options passed to cmake by the 'cmake' target
25   GENERATOR      the cmake generator. Currently '-G Xcode'
26   PDIR           the generation folder. Currently 'libdir'
27   PREFIX         the install location prefix. Currently '/usr/local'
28
29 CMake options:
30   FMWK           [MacOS only] Generates a framework on MacOS. Default is on
31   GDB            Activates ggdb3 option. Default is off
32   LILY           Include lilypond part. Default is on
33 NOTE: CMake options can be passed using CMAKEOPT, e.g.
34   'make cmake CMAKEOPT=-DLILY=off'

```

14.3 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++17 compiler. Use Xcode on Mac OS™ and 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++17 at least. More recent versions are fine too.

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

```
1 jacquesmenu@macmini: ~/musicformats_local_clone > cd build
2 jacquesmenu@macmini: ~/musicformats_local_clone/build > make
3
4 ... ..
5
6 ** BUILD SUCCEEDED **
7
8 cd lib && [ -d musicformats.framework ] && tar czf musicformats.tgz musicformats.
   framework || echo "no framework"
9 no framework
```

The resulting executables are in build/bin:

```
1 jacquesmenu@macmini: ~/musicformats_local_clone/build > ls -sal bin
2 total 661992
3 0 drwxr-xr-x@ 25 jacquesmenu staff 800 Feb 16 09:17 .
4 0 drwxr-xr-x 10 jacquesmenu staff 320 Feb 16 09:15 ..
5 74864 -rwxr-xr-x 1 jacquesmenu staff 38327184 Feb 16 09:17 LilyPondIssue34
6 74864 -rwxr-xr-x 1 jacquesmenu staff 38330272 Feb 16 09:17 Mikrokosmos3Wandering
7 8432 -rwxr-xr-x 1 jacquesmenu staff 4314896 Feb 16 09:17 MusicAndHarmonies
8 8432 -rwxr-xr-x 1 jacquesmenu staff 4314880 Feb 16 09:17 RandomChords
9 8432 -rwxr-xr-x 1 jacquesmenu staff 4314880 Feb 16 09:17 RandomMusic
10 8624 -rwxr-xr-x 1 jacquesmenu staff 4414944 Feb 16 09:17 countnotes
11 16528 -rwxr-xr-x 1 jacquesmenu staff 8459488 Feb 16 09:17 displayMusicformatsHistory
12 16528 -rwxr-xr-x 1 jacquesmenu staff 8459488 Feb 16 09:17 displayMusicformatsVersion
13 79200 -rwxr-xr-x 1 jacquesmenu staff 40546848 Feb 16 09:17 msdlconverter
14 12480 -rwxr-xr-x 1 jacquesmenu staff 6387248 Feb 16 09:17 partsummary
15 8848 -rwxr-xr-x 1 jacquesmenu staff 4528752 Feb 16 09:17 readunrolled
16 64000 -rwxr-xr-x 1 jacquesmenu staff 32764864 Feb 16 09:17 xml2brl
17 66872 -rwxr-xr-x 1 jacquesmenu staff 34236560 Feb 16 09:17 xml2gmn
18 17160 -rwxr-xr-x 1 jacquesmenu staff 8782048 Feb 16 09:17 xml2guido
19 67552 -rwxr-xr-x 1 jacquesmenu staff 34584160 Feb 16 09:17 xml2ly
20 12392 -rwxr-xr-x 1 jacquesmenu staff 6342560 Feb 16 09:17 xml2midi
21 59720 -rwxr-xr-x 1 jacquesmenu staff 30574816 Feb 16 09:17 xml2xml
22 9104 -rwxr-xr-x 1 jacquesmenu staff 4657232 Feb 16 09:17 xmlclone
23 9256 -rwxr-xr-x 1 jacquesmenu staff 4735296 Feb 16 09:17 xmlfactory
24 8800 -rwxr-xr-x 1 jacquesmenu staff 4505008 Feb 16 09:17 xmliter
25 8680 -rwxr-xr-x 1 jacquesmenu staff 4442528 Feb 16 09:17 xmlread
26 11976 -rwxr-xr-x 1 jacquesmenu staff 6129760 Feb 16 09:17 xmltranspose
27 9248 -rwxr-xr-x 1 jacquesmenu staff 4734384 Feb 16 09:17 xmlversion
```

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

```
1 jacquesmenu@macmini: ~/musicformats_local_clone/build > ls -sal lib
2 total 1283720
3 0 drwxr-xr-x@ 6 jacquesmenu staff 192 Feb 16 09:17 .
4 0 drwxr-xr-x 10 jacquesmenu staff 320 Feb 16 09:15 ..
5 107368 -rwxr-xr-x 1 jacquesmenu staff 54970432 Feb 16 09:17 libmusicformats.0.9.60.
   dylib
6 0 lrwxr-xr-x 1 jacquesmenu staff 28 Feb 16 09:17 libmusicformats.0.dylib
   -> libmusicformats.0.9.60.dylib
7 1176352 -rw-r--r-- 1 jacquesmenu staff 592564120 Feb 16 09:16 libmusicformats.a
8 0 lrwxr-xr-x 1 jacquesmenu staff 23 Feb 16 09:17 libmusicformats.dylib ->
   libmusicformats.0.dylib
```


14.4 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.

14.4.1 User specific installation on Mac OS™

This is done with `make localinstall`:

```
1 jacquesmenu@macmini: ~/musicformats_local_clone/build > make localinstall
2 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.

14.4.2 Global installation on Mac OS™

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

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

14.5 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++17 compiler. Use Xcode on Mac OS™ and 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++17 at least. More recent versions are fine too.

Once in the local repository clone, just execute:

```
1 cd build
2
3 cmake --build <builddir> --target install
```

Part V

Options and help (OAH)

Chapter 15

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 "whaaaaah!" 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 17.2 [Querying about options by name], page 63,.

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 it 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 input 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 16

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 [MusicFormatsAPIGuide](#) 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 software, such as `-h` (help), and `-o` (output file name):

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

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

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

and

```
1 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`:

```
1 jacquesmenu@macmini > xml2ly -query h
2 --- Help for prefix "h" ---
3     '-h=abc,xyz' is equivalent to '-habc, -hxyz'
4
5 --- Help for atom "h" in subgroup "Options and help"
6     -help, -h
7         Display xml2ly's full help.
```

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

```

1 jacquesmenu@macmini > xml2ly -va
2 Command line version of musicxml2lilypond converter v0.9.52 (November 29, 2021)
3
4 Representations versions:
5   MXSR
6     v0.9.5 (October 6, 2021)
7   MSR
8     v0.9.52 (November 27, 2021)
9   LPSR
10    v0.9.5 (October 6, 2021)
11
12 Passes versions:
13   mxsr2msr
14     v0.9.51 (November 27, 2021)
15   msr2msr
16     v0.9.51 (November 15, 2021)
17   msr2lpsr
18     v0.9.5 (October 6, 2021)
19   lpsr2lilypond
20     v0.9.52 (December 16, 2021)
21 What xml2ly does:
22
23   This multi-pass converter basically performs 5 passes:
24     Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
25              and converts it to a MusicXML tree;
26     Pass 2a: converts that MusicXML tree into
27              a first Music Score Representation (MSR) skeleton;
28     Pass 2b: populates the first MSR skeleton from the MusicXML tree
29              to get a full MSR;
30     Pass 3:  converts the first MSR into a second MSR to apply options
31     Pass 4:  converts the second MSR into a
32              LilyPond Score Representation (LPSR);
33     Pass 5:  converts the LPSR to LilyPond code
34              and writes it to standard output.
35
36   Other passes are performed according to the options, such as
37   displaying views of the internal data or printing a summary of the score.
38
39   The activity log and warning/error messages go to standard error.

```

16.1 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`:

```

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

```

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

```

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

```

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

```

1 jacquesmenu@macmini > xml2ly -query msr-pitches-language
2 --- Help for atom "msr-pitches-language" in subgroup "Notes"
3     -msr-pitches-language, -mplang LANGUAGE
4         Use LANGUAGE to display note pitches in the MSR logs and text views.
5         The 13 MSR pitches languages available are:
6         arabic, catalan, deutsch, english, espanol, francais,
7         italiano, nederland, norsk, portugues, suomi, svenska and vlaams.
8         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 17.2 [Querying about options by name], page 63,:

```

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

```

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

16.2 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:

```

1 jacquesmenu@macmini > xml2ly -query insider
2 --- Help for atom "insider" in subgroup "Options and help"
3     -insider, -ins
4         Use the 'insider' mode for the options and help,
5         in which the options are grouped as they are used internally by MusicFormats.
6         In the 'regular' default mode, they are grouped by user-oriented topics,
7         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:

```

1 jacquesmenu@macmini > xml2ly -query ignore-ornaments
2 --- Help for atom "ignore-ornaments" in subgroup "Ornaments"
3   -ignore-ornaments, -oorns
4     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:

```

1 jacquesmenu@macmini > xml2ly -query ignore-ornaments -insider
2 --- Help for atom "ignore-ornaments" in subgroup "Notes" of group "mxsr2msr" ---
3   -ignore-ornaments, -oorns
4     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.

16.3 Early options

A particular case of options is the *early options*, which are taken into account prior to the options being actually analyzed. This is 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 19 [Including options and arguments from a file], page 73, about this feature.

The early option values can be displayed with option `-display-early-options-values` and `-deov`:

```

1 jacquesmenu@macmini: ~ > xml2ly -quiet -display-early-options-values -trace-passes
2
3 %-----
4   Handle the options and arguments from argc/argv
5 %-----
6   Early options values:
7     EarlyInsiderOption      : true
8     EarlyQuietOption        : false
9     TraceEarlyOptions       : false
10    EarlyOahVerboseMode     : false
11    EarlyTracingOah         : false
12    EarlyTracingOahDetails  : false
13    EarlyTraceComponents    : false
14    EarlyTracePasses        : true
15    EarlyIncludeFileNamesList : none
16
17 Input file name or '-' for standard input expected

```

Chapter 17

Options and help introspection

17.1 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`:

```

1 jacquesmenu@macmini: ~ > xml2ly -help-midi
2 --- Help for subgroup "MIDI" in group "MIDI group" ---
3 MIDI group (-help-midi-group, -hmidi-group):
4 -----
5 MIDI (-help-midi, -hmidi):
6   -no-midi
7       Generate the '\midi' block as a comment instead of active code.
8   -midi-tempo, -mdtempo MIDI_TEMPO_SPEC
9       Generate a '\tempo' command in the \midi block.
10      MIDI_TEMPO_SPEC can be:
11      'DURATION = PER_SECOND'
12      or
13      "DURATION = PER_SECOND" .
14      DURATION is a string such as '8.', and PER_SECOND is an integer.
15      The single or double quotes are used to allow spaces around the '=' sign,
16      otherwise they can be dispensed with.
17      Using double quotes allows for shell variables substitutions, as in:
18      PER_SECOND=66
19      xml2ly -midiTempo "8. ${PER_SECOND}" .
20      The default is '8 = 180'.

```

17.2 Querying about options by name

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

```

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

```

```

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

```

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


```

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

```

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

```

1 jacquesmenu@macmini > xml2ly -name-help
2 --- Help for atom "name-help" in subgroup "Options and help"
3     -name-help, -nh OPTION_NAME
4         Print help about OPTION_NAME.
5         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...

17.3 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:

```

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

```

```

1 jacquesmenu@macmini > xml2ly -find output-file-name
2 2 occurrences of string "output-file-name" have been found:
3     1:
4         -output-file-name, -o FILENAME
5         Write output to file FILENAME instead of standard output.
6     2:
7         -auto-output-file-name, -aofn
8         This option can only be used when writing to a file.
9             Writethe output to a file in the current working directory.
10            The file name is derived from that of the input file:
11            any suffix after the '.' is replaced by one suited for the output
        format,
12            or such a suffix is adde if no '.' is present.

```

17.4 Displaying help about options usage

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

```

1 jacquesmenu@macmini > xml2ly -help-options-usage
2 xml2ly options usage:
3     In xml2ly, '-' as an argument, represents standard input.
4
5     Most options have a short and a long name for commodity.
6     The long name may be empty if the short name is explicit enough.
7
8     The options are organized in a group-subgroup-atom hierarchy.
9     Help can be obtained for groups or subgroups at will,
10    as well as for any option with the '-name-help, -nh' option.
11
12    A subgroup can be showm as a header only, in which case its description is printed

```

```

13 only when the corresponding short or long names are used.
14
15 Both '-' and '--' can be used to introduce options,
16 even though the help facility only shows them with '-'.
17
18 There some prefixes to allow for shortcuts,
19 such as '-t=voices,meas' for '-tvoices, -tmeas'.
20
21 The options can be placed in any order,
22 provided the values immediately follow the atoms that need them.
23
24 Using options that attempt to create files, such as '-o, -output-file-name',
25 leads to an error if the environment is read-only access,
26 as is the case of https://libmusicxml.gnome.fr .

```

17.5 Displaying a help summary

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

```

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

```

Chapter 18

Options examples

18.1 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:

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

18.2 Options simple values

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

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

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

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

```

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

```

```

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

```

18.3 Options more complex values

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

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

```

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

```

18.4 More complex options

The boolean options in MusicFormats can be combined:

```

1 jacquesmenu@macmini: ~ > xml2ly -query cubase
2 --- Help for atom "cubase" in subgroup "Input"
3 -cubase:
4     Useful settings for MusicXML data exported from Cubase.
5     This combined option is equivalent to:
6     -ignore-redundant-clefs, -irclefs:
7         Ignore clefs that are the same as the current one.
8     -ignore-redundant-keys, -irkeys:
9         Ignore keys that are the same as the current one.
10    -ignore-redundant-times, -irtimes:
11        Ignore times that are the same as the current one.
12 jacquesmenu@macmini: ~ >

```

Options can also share a common prefix:

```

1 jacquesmenu@macmini: ~ > xml2ly -query trace-when-handling-musicxml-data -insider
2 --- Help for atom "trace-when-handling-musicxml-data" in subgroup "MXSR Trace" of group "
  Mxsr" ---
3 -t<SHORT_NAME>, -trace--<LONG_NAME>
4     Trace SHORT_NAME/LONG_NAME when handling MusicXML data.
5 The 2 known SHORT_NAMES are:
6     enc and divs.
7 The 2 known LONG_NAMES are:
8     encoding and divisions.

```

OAH offers *macro options*, such as:

```

26 --- Help for atom "auto-utf8d" in subgroup "Files"
27 -auto-utf8d, -au8d:
28     To ease the production of braille files.
29     This macro option is equivalent to:
30     -auto-output-file-name, -aofn:
31         This option can only be used when writing to a file.
32         Writethe output to a file in the current working directory.
33         The file name is derived from that of the input file:
34         any suffix after the '.' is replaced by one suited for the output format,
35         or such a suffix is adde if no '.' is present.
36 -use-encoding-in-file-name, -ueifn:
37     Append a description of the encoding used
38     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:

```

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

```

18.5 Displaying the options values

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

```

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

```

Executing this command:

```

1 jacquesmenu@macmini > xml2ly -global-staff-size 30 -display-cpu-usage -display-options-
  values
2 The options values for xml2ly are:
3 Informations group (-help-informations-group, -hinfos-group), 1 atom chosen:
4 -----
5 Informations (-help-informations, -hinfos), 1 atom chosen:
6 fDisplayCPUUsage : true, has been set by user
7
8 Options and help group (-help-oah-group, -hoah-group), 1 atom chosen:
9 -----
10 Options and help (-help-oah, -hoah), 1 atom chosen:
11 fDisplayOptionsValues : true, has been set by user
12
13 Layout group (-help-layout-group, -hlayout-group), 1 atom chosen:
14 -----
15 Layout (-help-layout, -hlayout), 1 atom chosen:
16 fGlobalStaffSize : 30, has been set by user
17
18 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-values-all`: displays the whole set of options with their values, and whether they have been set by the user:

```

1 jacquesmenu@macmini > xml2ly -global-staff-size 30 -display-cpu-usage -display-options-
  values-all
2
3 All the options values for xml2ly are:
4
5 OAH Trace (-help-trace, -ht):
6 -----
7 Other (-help-trace-other, -hto):
8 fTraceComponents : false
9 fTracePasses : false
10 fTraceGeometry : false
11 fTraceIdentification : false
12 fTraceForTests : false
13
14 ... ..
15
16 Informations group (-help-informations-group, -hinfos-group):
17 -----
18 Informations (-help-informations, -hinfos):
19 fDisplayCPUUsage : true, has been set by user
20
21 Files group (-help-files-group, -hfiles-group):
22 -----
23 Files (-help-files, -hfiles):
24 fOutputFileName :
25 fAutoOutputFileName : false
26
27 Options and help group (-help-oah-group, -hoah-group):
28 -----
29 Options and help (-help-oah, -hoah):
30 insider : fOptionHasBeenSelected: false
31 fOahVerboseMode : false
32 fReverseNamesDisplayOrder : false
33 fDisplayOptionsValues : true, has been set by user
34
35 ... ..
36
37 Staves group (-help-staves-group, -hstaves-group):
38 -----
39 Staves (-help-staves, -hstaves):
40 fCreateVoicesStaffRelativeNumbers : false
41 fLedgeLinesRGBColor : [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   fNoRaggedBottom       : false
74   fNoRaggedLast         : false
75   fNoRaggedLastBottom   : false
76   fNoRaggedRight        : 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",
97   midiTempoPerSecond = 180, line 0]
98
99 Input file name or '-' for standard input expected

```

18.6 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:


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

For example, consider xml2ly:

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

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

```
1 jacquesmenu@macmini > xml2ly -find display-lpsr
2 3 occurrences of string "display-lpsr" have been found:
3     1:
4       -display-lpsr, -dlpsr
5       Write the contents of the LPSR data with a summary of its MSR component to standard
6       error.
7     2:
8       -display-lpsr-full, -dlpsrfull
9       Write the contents of the LPSR data with its full MSR component to standard error.
10    3:
11    -display-lpsr-short, -dlpsrshort
12    Write the contents of the LPSR data, short version, to standard error.
```

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

Chapter 19

Including options and arguments from a file

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

```
1 jacquesmenu@macmini > xml2ly -query include
2 --- Help for atom "include" in subgroup "Options and help"
3     -include, -inc FILENAME
4         Include the options and arguments contained in FILENAME.
5         FILENAME is a string and should be a path to a text file.
6         Such a file is expected to hold at most one option or argument per line.
7         A '#' starts a comment that spans to the end of the line.
8         Comments and empty lines are ignored and can be used at will.
9         '-include, -inc' options may be used to include other files,
10        up to a maximum level of 10.
11        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 standard input.

19.1 An options and arguments file example

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

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/
  AnacrusisOptionsAndArguments.txt
2 # some options
3
4 # output file
5 -auto-output-file-name
6
7 # contents
8 -title "Anacrusis created with '-include' option"
9 -subtitle "Just for the fun"
10
11 # layout
12 -global-staff-size 30
13
14 # non-musical
15 -cpu
16
17 # the MusicXML file
18
19 basic/Anacrusis.xml
```

Including this file with `xml2ly` gives:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -include basic/
  AnacrusisOptionsAndArguments.txt
2 Timing information:
3
4 Activity      Description                                     Kind      CPU (sec)
5 -----
6
7             Handle the options and arguments from argc/argv      mandatory  0.03038
8 Pass 1        Create an MXSR reading a MusicXML file      mandatory  0.00353
9 Pass 2a       Create an MSR skeleton from the MXSR        mandatory  0.00071
10 Pass 2b      Populate the MSR skeleton from MusicXML data mandatory  0.00139
11 Pass 3       Convert the first MSR into a second MSR     mandatory  0.00037
12 Pass 4       Convert the second MSR into an LPSR        mandatory  0.00039
13 Pass 5       Convert the LPSR score to LilyPond code     mandatory  0.00088
14
15 Total (sec)  Mandatory  Optional
16 -----
17 0.03766      0.03766    0.00000
18
19 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > ls -sal Anacrusis.ly
20 8 -rw-r--r--@ 1 jacquesmenu  staff   1553 Feb  9 09:44 Anacrusis.ly

```

The resulting score is:

Anacrusis created with '-include' option

Just for the fun



19.2 Options values and arguments in included files

As shown in section 11.5 [Quoting, variables and aliases], page 40, 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:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/basic > cat
  QuotingInIncludedOptionsFiles.txt
2 # contents
3 -title This year's title
4 -subtitle 'Last year's quoted multi-word subtitle'
5 -subsubtitle "Double quoted multi-word subsubtitle"
6
7 # display
8 -display-options-values
9
10 # LilyPond
11 -lilypond-generation-infos
12
13 # output
14 -auto-output-file-name

```

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

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles/basic > xml2ly -include
  QuotingInIncludedOptionsFiles.txt HelloWorld.xml
2 The options values for xml2ly are:
3 Files group (-help-files-group, -hfiles-group), 1 atom chosen:
4 -----
5 Files (-help-files, -hfiles), 1 atom chosen:
6 fAutoOutputFileName : true
7
8 Options and help group (-help-oah-group, -hoah-group), 1 atom chosen:
9 -----
10 Options and help (-help-oah, -hoah), 1 atom chosen:
11 fDisplayOptionsValues : true
12
13 Header group (-help-header-group, -hheader-group), 3 atoms chosen:
14 -----
15 Header (-help-header, -hheader), 3 atoms chosen:
16 fTitle : This year's title
17 fSubTitle : Last year's quoted multi-word subtitle
18 fSubSubTitle : Double quoted multi-word subsubtitle
19
20 Output group (-help-output-group, -houtput-group), 1 atom chosen:
21 -----
22 Output (-help-output, -houtput), 1 atom chosen:
23 fXml2lyInfos : true

```

19.3 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:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/
  HelloWorldOptionsAndArguments_1.txt
2 # output file
3 -auto-output-file-name
4
5 # contents
6 -title 'My title'
7 -subtitle " Nice subtitle"
8 -subsubtitle "Subsubtitle from HelloWorldOptionsAndArguments_1.txt"
9
10 # layout
11 -global-staff-size 30
12
13 # non-musical
14 -display-cpu-usage
15
16 # the MusicXML file
17 basic/HelloWorld.xml
18
19 # nested include
20 -include basic/HelloWorldOptionsAndArguments_2.txt

```

The included `basic/HelloWorldOptionsAndArguments_2.txt` file contains:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/
  HelloWorldOptionsAndArguments_2.txt
2 # non-musical
3 -subsubtitle "Subsubtitle from HelloWorldOptionsAndArguments_2.txt"
4
5 # cycle detection check
6 # -include HelloWorldOptionsAndArguments_1.txt

```

Including `basic/HelloWorldOptionsAndArguments_1.txt`, we get:

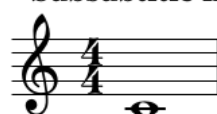
```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -include basic/
  HelloWorldOptionsAndArguments_1.txt
2 Timing information:
3
4 Activity      Description                                     Kind      CPU (sec)
5 -----
6
7              Handle the options and arguments from argc/argv      mandatory  0.02962
8 Pass 1        Create an MXSR reading a MusicXML file      mandatory  0.00362
9 Pass 2a       Create an MSR skeleton from the MXSR        mandatory  0.00185
10 Pass 2b      Populate the MSR skeleton from MusicXML data mandatory  0.00288
11 Pass 3       Convert the first MSR into a second MSR     mandatory  0.00092
12 Pass 4       Convert the second MSR into an LPSR        mandatory  0.00090
13 Pass 5       Convert the LPSR score to LilyPond code    mandatory  0.00143
14
15 Total (sec)  Mandatory  Optional
16 -----
17 0.04122     0.04122   0.00000

```

The resulting score is:

My title
Nice subtitle
Subsubtitle from HelloWorldOptionsAndArguments_2.txt



19.4 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:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > cat basic/
  HelloWorldOptionsAndArguments_2.txt
2 # non-musical
3 -subsubtitle "Subsubtitle from HelloWorldOptionsAndArguments_2.txt"
4
5 # cycle detection check
6 -includeinc basic/HelloWorldOptionsAndArguments_1.txt

```

Now we get:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/musicxmlfiles > xml2ly -include basic/
  HelloWorldOptionsAndArguments_1.txt
2           Including file [basic/HelloWorldOptionsAndArguments_1.txt]: more
  than 10 include levels, quitting
3           The include file names stack contains 10 elements:
4             1: [basic/HelloWorldOptionsAndArguments_2.txt]
5             2: [basic/HelloWorldOptionsAndArguments_1.txt]
6             3: [basic/HelloWorldOptionsAndArguments_2.txt]
7             4: [basic/HelloWorldOptionsAndArguments_1.txt]
8             5: [basic/HelloWorldOptionsAndArguments_2.txt]
9             6: [basic/HelloWorldOptionsAndArguments_1.txt]
10            7: [basic/HelloWorldOptionsAndArguments_2.txt]
11            8: [basic/HelloWorldOptionsAndArguments_1.txt]
12            9: [basic/HelloWorldOptionsAndArguments_2.txt]
13           10: [basic/HelloWorldOptionsAndArguments_1.txt]

```

19.5 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`, `-no-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`:

```

1 jacquesmenu@macmini > cat Bethena_SaxTenor_OptionsAndArguments.txt
2 # argument
3   Bethena_SaxTenor.xml
4
5 # output
6   -auto-output-file-name
7
8 # informations
9   -lilypond-generation-infos
10
11 # presentation
12   -global-staff-size 23
13   -page-count 3
14   -no-ragged-last-bottom
15
16 # words conversions
17   -convert-musicxml-words-to-msr-tempo "Valse tempo (Thema)"
18   -convert-musicxml-words-to-msr-tempo "A tempo (Valse cantabile)"
19   -convert-musicxml-words-to-msr-tempo "A tempo"
20   -convert-musicxml-words-to-msr-tempo "a tempo"
21   -convert-musicxml-words-to-msr-tempo "Finale"
22   -convert-musicxml-words-to-msr-tempo "Andante"
23   -convert-musicxml-words-to-msr-tempo "Tempo I"
24
25 # tweaking
26   -repeat-brackets
27
28   -all-wedges-below
29

```

30 | `-ignore-slurs`

Chapter 20

Non-musical options

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

20.1 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:

```

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

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

```

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

```

20.2 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.

```

```

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

```

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

```

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

```

Chapter 21

Trace options

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

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

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

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

```
1 menu@macbookprojm > xml2ly -help=trace
2
3 --- Help for group "Trace" ---
4
5 Trace (-ht, -help-trace) (hidden by default)
6   There are trace options transversal to the successive passes,
7   showing what's going on in the various translation activities.
8   They're provided as a help for the maintenance of MusicFormats,
9   as well as for the curious.
10  The options in this group can be quite verbose, use them with small input data!
11  All of them imply '-trace-passes, -tpasses'.
12  -----
13  Options handling trace          (-htoh, -help-trace-options-handling):
14    -toah, -trace-oah
15      Write a trace of options and help handling to standard error.
16      This option should best appear first.
17    -toahd, -trace-oah-details
18      Write a trace of options and help handling with more details to standard error.
19      This option should best appear first.
20  Score to voices                (-htstv, -help-trace-score-to-voices):
21    -t<SHORT_NAME>, -trace<LONG_NAME>
22      Trace SHORT_NAME/LONG_NAME in score to voices.
23      The 9 known SHORT_NAMES are:
24        score, pgroups, pgroupsd, parts, staves, st, schanges, voices and voicesd.
25      The 9 known LONG_NAMES are:
26        -score, -part-groups, -part-groups-details, -parts, -staves.
27  ... ..
```

As can be seen, there are event options to trace the handling of options and help by xml2ly.

The source code contains many instances of trace code, such as:

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

Chapter 21. Trace options

Building `xm121y` 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 22

Warnings and errors (WAE)

Part VII

Multiple languages support

Chapter 23

Multiple languages support

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

For example, `xm12ly` offers several languages options:

```

1 jacquesmenu@macmini > xm12ly -find language
2 6 occurrences of string "language" have been found:
3   1:
4     -msr-pitches-language, -mplang LANGUAGE
5     Use LANGUAGE to display note pitches in the MSR logs and text views.
6         The 13 MSR pitches languages available are:
7         arabic, catalan, deutsch, english, espanol, francais,
8         italiano, nederland, norsk, portugues, suomi, svenska and vlaams.
9         The default is 'kQTPNederland'.
10  2:
11     -lpsr-pitches-language, -lppl LANGUAGE
12     Use LANGUAGE to display note pitches in the LPSR logs and views,
13     as well as in the generated LilyPond code.
14     The 13 LPSR pitches languages available are:
15     arabic, catalan, deutsch, english, espanol, francais,
16     italiano, nederland, norsk, portugues, suomi, svenska and vlaams.
17     The default is 'kQTPNederland'.
18  3:
19     -lpsr-chords-language, -lpcl LANGUAGE
20     Use LANGUAGE to display chord names, their root and bass notes,
21     in the LPSR logs and views and the generated LilyPond code.
22     The 5 LPSR chords pitches languages available are:
23     french, german, ignatzek, italian and semiGerman.
24     'ignatzek' is Ignatzek's jazz-like, english naming used by LilyPond by
25     default.
26     The default is 'kChordsIgnatzek'.
27  4:
28     -show-all-harmonies-contents, -sacc PITCH
29     Write all harmonies contents for the given diatonic (semitones) PITCH,
30     supplied in the current language to standard output.
31  5:
32     -show-harmony-details, -scd HARMONY_SPEC
33     Write the details of the harmony for the given diatonic (semitones) pitch
34     in the current language and the given harmony to standard output.
35     HARMONY_SPEC can be:
36     'ROOT_DIATONIC_PITCH HARMONY_NAME'
37     or
38     "ROOT_DIATONIC_PITCH = HARMONY_NAME"
39     Using double quotes allows for shell variables substitutions, as in:
40     HARMONY="maj7"
41     xm12ly -show-harmony-details "bes ${HARMONY}"
42  6:
43     -show-harmony-analysis, -sca HARMONY_SPEC

```



```
43     Write an analysis of the harmony for the given diatonic (semitones) pitch
44         in the current language and the given harmony to standard output.
45     HARMONY_SPEC can be:
46     'ROOT_DIATONIC_PITCH HARMONY_NAME INVERSION'
47     or
48     "ROOT_DIATONIC_PITCH = HARMONY_NAME INVERSION"
49     Using double quotes allows for shell variables substitutions, as in:
50     HARMONY="maj7"
51     INVERSION=2
52     xml2ly -show-harmony-analysis "bes ${HARMONY} ${INVERSION}"
```

Part VIII

The MusicFormats Scripting Language (MFSL)

Chapter 24

MFSL (MusicFormats Scripting Language)

MFSL is meant for launching MusicFormats tools easily, gathering options for them and providing selection criteria to do so. The syntax and semantics are very simple, for use by musicians.

A file containing MFSL elements is called a *script*, with name extension `.mfs1` typically, though other extensions can be used at will.

The main features of MFSL are:

- the options are written the same as in OAH, such as `-global-staff-size 25.5`;
- the numbers and strings in options values are written the same as in OAH too – ‘a single quotes string’, “a double quotes string”;
- a small number of names are reserved keywords:
 - `tool`, to specify the MusicFormats tool to be run;
 - `input`, to indicate the input source when the tool is launched, a file name or – for standard input;
 - `choice`, `default` and `case`, to select options or others depending on the needs;
 - `select` and `all`, to indicate whether the tool should be launched once or multiple times;
 - comments can be used in scripts: they can be span from a `#` to the end of the line, or span multiple lines, starting and ending with `###`.

The names in a choice specification are so-called *labels*, here “tutti”, “soprano”, “alto”, “tenor” and “bass”:

```
1 choice SCORE :
2   tutti | soprano | alto | tenor | bass ,
3
4   default: tutti
5 ;
```

An *interpreter* named `mfs1` is provided by MusicFormats that can run MFSL scripts as is done by the usual shells and languages such as Python or Ruby:

```
1 jacquesmenu@macmini > mfs1 -about
2 What mfs1 does:
3
4   This interpreter reads text input containing
5   a tool name, an input source name, keywords and options ,
6   and launches the specified tool
7   with these options applied to the input source name.
8
9   The input can come from a file or from standard input ,
10  allowing the interpreter to be used interactively or in shell pipes.
11
12  The activity log and warning/error messages go to standard error.
```

This chapter first presents MFSL with example scripts, and then the MFSL language in a more formal way.

24.1 A MusicXML example

We use the `src/interpreters/mfsl/Hymn121.xml` MusicXML file in this chapter to show more and more useful uses of the MFSL interpreter.

The score used as a starting point is:

121. ANGELS FROM THE REALMS OF GLORY Flemish

Allegro moderato

From that:

- the original PDF file has been scanned with Photo Score Ultimate™ to obtain `src/interpreters/mfsl/Hymn121.xml`
- a glitch has been introduced in the last measure of part P4 in the form of a `<lyric/>` element:

```
1 <lyric number="1">
2   <syllabic>begin</syllabic>
3   <text>glitch</text>
4 </lyric>
```

24.2 A first, minimal MFSL script example

The `src/interpreters/mfsl/minimal.mfsl` script illustrates two basic features of MFSL:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > cat minimal.mfsl
2 #!/Users/jacquesmenu/musicformats-git-dev/build/bin/mfsl
3
4 # This is a comment (for human readers), from '#' to the end of line
5
6 # The first line above contains a so-called shebang,
7 # a special comment indicating the access path to the MFSL interpreter
8 # used to process the remainder of this file
9
10 ###
11 And here is a multi-line comment.
12
```

```

13 This minimal MFSL script contains only the mandatory elements:
14 - a tool specification, telling which MusicFormats tool to use,
15   here xml2ly;
16
17 - an input specification, specifying the input source for this tool,
18   here file name "Hymn121.xml".
19
20 It can be a file name or '-' to specify standard input.
21
22 Typing './minimal.mfsl' in a terminal causes xml2ly to be launched,
23 in a way equivalent to the direct command:
24     xml2ly Hymn121.xml
25
26 Useful scripts contain options in complement to that,
27 since MFSL is intended to gather options to be used by the tool.
28 ###
29
30 # -----
31 # the MusicFormats tool to be used
32 # -----
33
34 tool : xml2ly
35
36
37 # -----
38 # the input file to be processed by the tool
39 # -----
40
41 input : "Hymn121.xml"
42
43 # double quote can be used also to delimitate strings:
44 # input : 'Hymn121.xml'
45
46 # this would be OK too, since there are no spaces in the file name
47 # input : Hymn121.xml

```

24.3 Launching an MFSL script

This first line of an MFSL script is the so-called *shebang*. It contains the path to the MFSL interpreter, allowing running such a script by its name from the terminal, provided the script file is made executable.

This can be done in the operating system file system GUI, or using the `chmod` command or equivalent in a terminal:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > chmod +x minimal.mfsl
2
3 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > ls -sal minimal.mfsl
4 -rwxr-xr-x@ 1 jacquesmenu  staff  1488 Mar 26 18:20 minimal.mfsl

```

The MFSL interpreter has the `-no-launch`, `-nol` option to display the essential information found in the script and quit without actually running the tool:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > mfsl -query no-launch
2 --- Help for atom "no-launch" in subgroup "MFSL"
3 -no-launch, -nol
4     Analyze the MFSL script, but don't launch the tool actually.
5     This is useful to check the options gathered by the MFSL interpreter,
6     and what command(s) would be launched.
7     This option implies the '-display-tool-and-input, -ttai' option.
8 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles >

```

The `-display-tool-and-input`, `-dtai` option itself has this effect:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > mfsl -query display-tool-and
  -input
2 --- Help for atom "display-tool-and-input" in subgroup "MFSL"
3   -display-tool-and-input, -dtai
4     Write MFSL tool and input analysis activity to standard output.
```

When running the `src/interpreters/mfsl/minimal.mfsl` script from the command line with option `-no-launch`, `-nol`, we get:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > ./minimal.mfsl -no-launch
2 ==> The known tools names are: ["msdlconverter", "xml2brl", "xml2gmn", "xml2ly" and "
  xml2xml"]
3 ==> tool: xml2ly
4 ==> input: Hymn121.xml
5 ==> Launching xml2ly with the argument and option gathered from ./minimal.mfsl
6 ==> The command to be executed is:
7   xml2ly Hymn121.xml
8 ==> The command above is *NOT* executed
```

Launching it without this options gives:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > ./minimal.mfsl > Minimal.ly
2
3 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > ls -sal Minimal.ly
4 24 -rw-r--r--  1 jacquesmenu  staff  11772 Mar 28 09:44 Minimal.ly
```

Figure 24.1 [The score created with `minimal.mfsl`], page 94, contains the score obtained with LilyPond from `src/interpreters/mfsl/Minimal.ly`.

24.4 Interactive use of the MFSL interpreter

As is traditional when using interpreters from the command line, the MFSL interpreter `mfsl` can be launched without any file name or with a `'-'` denoting standard input. If that is done in a terminal, a `Ctrl-d` (control key held when typing the `d`) should be typed after the input, to indicate the end of it, such as:

```
1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > mfsl
2 tool : xml2ly
3 input : "HelloWorld.xml"
4
5 -auto-output-file-name
6 -display-cpu-usage
7   ==> Ctrl-d typed here to indicate the end of input
8 Timing information:
9
10 Activity      Description                                     Kind      CPU (sec)
11 -----
12
13 Pass 1        Handle the options and arguments from argc/argv      mandatory  0.03659
14 Pass 2a       Create an MXSR reading a MusicXML file               mandatory  0.00336
15 Pass 2b       Create an MSR skeleton from the MXSR                 mandatory  0.00157
16 Pass 3        Populate the MSR skeleton from MusicXML data         mandatory  0.00199
17 Pass 4        Convert the first MSR into a second MSR              mandatory  0.00059
18 Pass 5        Convert the second MSR into an LPSR                  mandatory  0.00065
19 Pass 5        Convert the LPSR score to LilyPond code              mandatory  0.00111
20
21 Total (sec)   Mandatory   Optional
22 -----
23 0.04586      0.04586    0.00000
24
25 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > ls -sal HelloWorld.ly
26 8 -rw-r--r--@ 1 jacquesmenu  staff  1221 Mar 28 07:48 HelloWorld.ly
```

Figure 24.1: The score created with `minimal.mfsl`

121. ANGELS FROM THE REALMS OF GLORY

Flemish

Score for **121. ANGELS FROM THE REALMS OF GLORY** (Flemish), created with `minimal.mfsl`. The score is divided into four systems of staves.

System 1 (Measures 1-5): Four voices (Voice 1 to Voice 4) and piano accompaniment. Dynamics include *mp* (mezzo-piano).

System 2 (Measures 6-10): Continuation of the score. Dynamics include *f* (forte).

System 3 (Measures 11-15): Continuation of the score. Dynamics include *f* (forte).

System 4 (Measures 16-20): Continuation of the score. Dynamics include *ff* (fortissimo). The score ends with a *glitch* annotation.

24.5 The structure of MFSL scripts

MFSL is about collecting options to launch a MusicFormats tool using some specified input.

Thus, an MFSL script starts by:

- a mandatory `tool` specification: it tells which MusicFormats tool to use, here `xml2ly`;
- a mandatory `input` specification: it tells the input source to apply the tool to, such as `Hymn121.xml` above.

After that, there can be:

- any number of MusicFormats options;
- any number of `choice` specifications identified by so-called *labels*, to provide options selection criteria;
- any number of `case` statements stating the options to be used depending on the selected choice labels;
- optionally, a `select` or a `all` statement.

The details are presented in the forthcoming sections.

24.6 Options to the MFSL interpreter

The MFSL interpreter has its own options, which can be displayed by option `-help`, `-h`.

The specific options are placed in two groups. The first one, seen with option `-help-user`, `-hu`, is for regular users:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > mfsl -help-user
2 --- Help for subgroup "MFSL" in group "MFSL group" ---
3 MFSL group (-help-mfsl-user-group, -hmfsl-user-group):
4 -----
5 MFSL (-help-user, -hu):
6   -display-tokens, -dtoks
7       Write a trace of the MFSL tokens to standard error.
8   -display-tool-and-input, -dtai
9       Write MFSL tool and input analysis activity to standard output.
10  -display-options, -dopts
11      Write MFSL options analysis activity to standard output.
12  -select, -sel CHOICE:LABEL
13      Select LABEL for choice CHOICE.
14      The tool will be run once using the corresponding options block(s).
15  -all CHOICE
16      Select each label for choice CHOICE in turn.
17      The tool will be run as many times, using the corresponding options block(s).
18  -no-launch, -nol
19      Analyze the MFSL script, but don't launch the tool actually.
20      This is useful to check the options collected by the MFSL interpreter,
21      and what command(s) would be launched.
22      This option implies the '-display-tool-and-input, -ttai' option.
```

The second group, displayed by option `-help-maintainer`, `-hm`, is for the curious and the maintainers of MusicFormats:


```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > mfsl -help-maintainer
2 --- Help for subgroup "MFSL" in group "MFSL group" ---
3 MFSL group (-help-mfsl-maintenance-group, -hmfsl--maintainance-group):
4 -----
5 MFSL (-help-maintainer, -hm):
6   -trace-scanning, -tscan
7     Write a trace of the MFSL scanning by Flex-generated code to standard error.
8   -trace-parsing, -tparse
9     Write a trace of the MFSL parsing by Bison-generated code to standard error.
10  -trace-choices, -tchoices
11    Write MFSL choice analysis activity to standard output.
12  -trace-choice-statements, -tChoice
13    Write MFSL choice statements handling activity to standard output.
14  -trace-case-statements, -tcases
15    Write MFSL case statements handling activity to standard output.
16  -trace-options-blocks, -toblocks
17    Write MFSL options blocks analysis activity to standard output.

```

24.7 Options in an MFSL script body

Let us now use options to produce a bass guitar score from the same `src/interpreters/mfsl/Hymn121.xml` file, which contains four parts named P1 to P4.

This is done by:

- filtering the MusicXML data to keep only part P4;
- using options to obtain the desired result.

For example, in `src/interpreters/mfsl/Hymn121ForBassGuitarPart.mfsl`, the `<lyric/>` element is removed with option `-ignore-musicxml-lyrics`, `-imlyrics`:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > cat Hymn121ForBassGuitarPart
2 .mfsl
3
4 #####
5 This MFSL script contain s:
6   - a 'tool' specification, telling which MusicFormats tool
7     to use, here xml2ly;
8
9   - an 'input' specification, the input source to apply
10     the latter to, here file name "Hymn121.xml";
11
12   - any number of MusicFormats options,
13     which can easily be commented/uncommented at will.
14 #####
15
16
17 # -----
18 # the MusicFormats tool to be used
19 # -----
20
21 tool : xml2ly
22
23
24 # -----
25 # the input file to be handled by the tool
26 # -----
27
28 input : "Hymn121.xml"

```

```

29
30
31 # -----
32 # the options block
33 # -----
34
35 # output
36 -output-file-name "Hymn121_BassGuitar.ly"
37
38 # header
39 -title "Hymn 121 - Angels From The Realms Of Glory"
40 -subtitle "Part 4 - Bass guitar"
41
42 -keep-musicxml-part-id P4
43
44 -ignore-musicxml-lyrics
45   # there's a glitch at the last measure...
46
47 -ragged-bottom off
48
49 -ambitus
50
51 # dynamics
52 -all-wedges-below
53
54 # staff
55 #   -modern-tab
56 #   -tab-full-notation
57
58 # display
59 #   -display-options-values
60
61 # LilyPond
62 -lilypond-generation-infos

```

Running this script with:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > ./Hymn121ForBassGuitarPart.
  mfs1
2
3 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > ls -sal Hymn121_BassGuitar.
  ly
4 8 -rw-r--r--@ 1 jacquesmenu  staff  3952 Mar 27 08:55 Hymn121_BassGuitar.ly

```

produces this score:

Hymn 121 - Angels From The Realms Of Glory
Part 4 - Bass guitar

Flemish

The image shows a musical score for a bass guitar part. It is titled 'Hymn 121 - Angels From The Realms Of Glory' and 'Part 4 - Bass guitar'. The score is written for 'Voice 4' (bass) and includes dynamics like *mp*, *f*, and *ff*. The notation is in 4/4 time and features various musical symbols like notes, rests, and wedges. The score is labeled 'Flemish' in the top right corner.

24.8 Using choices in an MFSL script

Let us go one step further, using `src/interpreters/mfsl/Hymn121.xml` again, to obtain various scores for a vocal SATB (soprano, alto, tenor, bass) choir.

This example shows the use of a **choice** specification and a **case** statement, to indicate the various possible score choices and what options to use for them, and **select** or **every** statements or options to do the selection.

The actual selection works this way:

- there can be a **select** or **every** statement, only one of those, at the end of the script;
- if one of them is present in the script, it is used to decide which options to use for the tool;
- this behaviour can be overridden by the use of **-select**, **-sel** or **-every**, respectively;
- if none of **select** or **every** is present, the default label for the choice is implicitly used.

The `src/interpreters/mfsl/Hymn121ForVocalSATBQuartet.mfsl` script contains:

```

1 jacquesmenu@macmini: ~/musicformats-git-dev/files/mfslfiles > cat
   Hymn121ForVocalSATBQuartet.mfsl
2 #!//Users/jacquesmenu/musicformats-git-dev/build/bin/mfsl
3
4 ###
5 This MFSL script contains:
6 - a tool specification, telling which MusicFormats tool
7   to use, here xml2ly;
8
9 - an input specification, the input source to apply the latter to,
10  here file name "Hymn121.xml";
11
12 - a number of MusicFormats options;
13
14 - a SCORE choice specification containing 5 label,
15   each telling which options to use
16   to get a score generated with this script and xml2ly/LilyPond;
17
18 - a 'select' specification to choose one such SCORE label;
19 ###
20
21 # -----
22 # the MusicFormats tool to be used
23 # -----
24
25 tool : xml2ly
26
27
28 # -----
29 # the input file to be handled by the tool
30 # -----
31
32 input : "Hymn121.xml"
33
34
35 # -----
36 # some common options
37 # -----
38
39 # header
40 -title "Hymn 121 - Angels From The Realms Of Glory"
41
42 # display
43 # -display-options-values
44
45 # LilyPond
46 -lilypond-generation-infos
47
48

```

```

49 # dynamics
50 -all-wedges-below
51
52
53 # -----
54 # the labels for the SCORE choice
55 # -----
56
57 # choose names that match the intent,
58 # uppercase to emphasize a choice name is a matter of taste
59
60 choice SCORE :
61     tutti | soprano | alto | tenor | bass,
62
63     default: tutti
64 ;
65
66
67 # -----
68 # tell which options are to be used depending on SCORE labels
69 # -----
70
71 case SCORE :
72     tutti:
73         -output-file-name "Hymn121_SATB_tutti.ly"
74
75         -global-staff-size 17.675
76         -subtitle "Tutti"
77     ;
78
79     soprano:
80         -output-file-name "Hymn121_SATB_soprano.ly"
81
82         -markup { "B" \hspace #-0.375 \raise #1.5 {\flat} " instruments" }'
83         -keep-musicxml-part-id P1
84         -msr-rename-part "P1:soprano"
85
86     # -display-lpsr-short
87     ;
88
89     alto:
90         -output-file-name "Hymn121_SATB_alto.ly"
91
92         -subtitle "Alto"
93         -keep-musicxml-part-id P2
94     ;
95
96     tenor:
97         -output-file-name "Hymn121_SATB_tenor.ly"
98
99         -subtitle "Tenor"
100         -keep-musicxml-part-id P3
101
102         -global-staff-size 23 # an easier-to-read score is needed
103     ;
104
105     bass:
106         -output-file-name "Hymn121_SATB_bass.ly"
107
108         -subtitle "Bass"
109         -keep-musicxml-part-id P4
110
111         -ignore-musicxml-lyrics # there's a glitch at the last measure
112     ;
113 ;
114
115

```

```

116 # -----
117 # choosing which score(s) to generate
118 # -----
119
120 ###
121 Comment/uncomment the statements below to choose
122 either a particular SCORE or every one of them to be generated.
123 Only one of 'select' and 'every' can be used.
124
125 The 'select' statement chooses one particular score, overriding
126 the default "tutti" label setting in the 'choice' statement.
127
128 This can also be done with an option to the MFSL interpreter such as:
129     -select SCORE:tenor
130
131 The 'every' statement causes all possible scores for the given choice
132 to be generated.
133
134 This can also be done with this option to the MFSL interpreter:
135     -every SCORE
136 ###
137
138 # select SCORE : tutti ; # superfluous actually
139 # select SCORE : soprano ;
140 # select SCORE : alto ;
141 # select SCORE : tenor ;
142 # select SCORE : bass ;
143
144 # every SCORE ;

```

24.9 Running the tool for one choice label

`src/interpreters/mfsl/Hymn121ForHarmonyBand.mfsl`

24.10 Running the tool for every choice label

24.11 MFSL language details

24.12 Error handling in MFSL scripts

24.13 Behind the scenes...

The MFSL interpreter is organized as two passes:

1. a first pass analyses the script body and build an internal representation, which is displayed by the `-trace-choices`, `-tchoices` option;
2. the second pass uses the informations thus collected to launch the tool as many times as needed, depending on `select` and `all` usage in the script body and the options to MFSL interpreter.

Part IX

xml2ly

Chapter 25

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.

25.1 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 developer, 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.

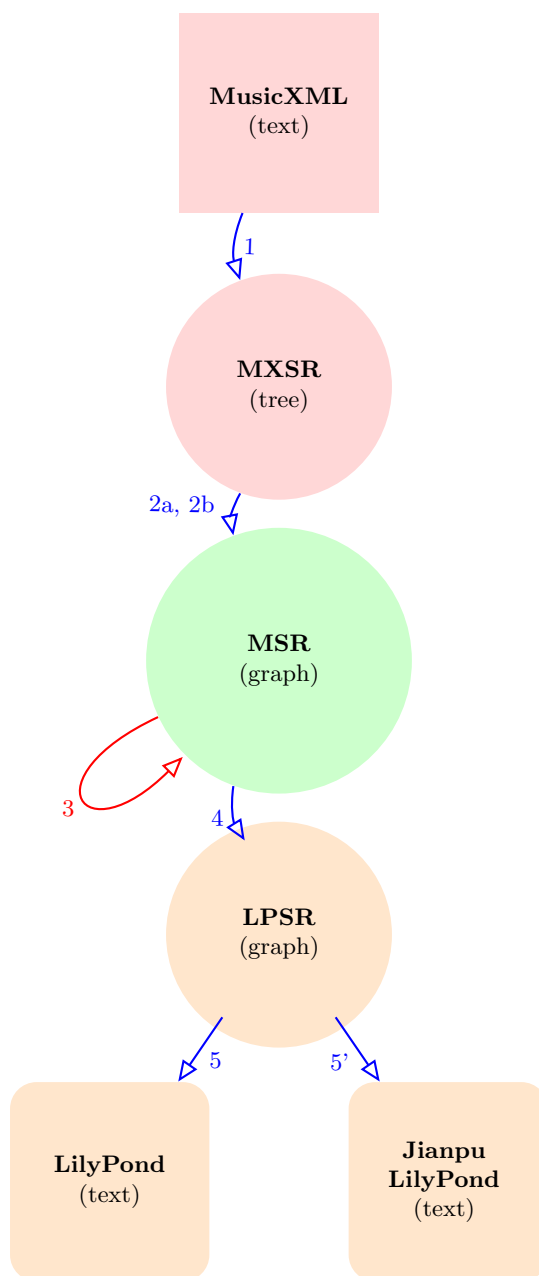
Speed was not an explicit goal, but as it turns out, `xml2ly` is not bad in this respect.

25.2 What xml2ly does

`xml2ly` performs the 5 steps from MusicXML to LilyPond to translate the former into the latter, as shown in figure [25.1](#) [`xmlToLyArchitecture`], page [103](#).. Converting from MXSR to MSR is done in two sub-phases for implementation reasons.

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

Figure 25.1: xml2ly architecture




```

1 jacquesmenu@macmini > xml2ly -about
2 What xml2ly does:
3
4 This multi-pass converter basically performs 5 passes:
5   Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
6             and converts it to a MusicXML tree;
7   Pass 2a: converts that MusicXML tree into
8             a first Music Score Representation (MSR) skeleton;
9   Pass 2b: populates the first MSR skeleton from the MusicXML tree
10            to get a full MSR;
11   Pass 3:  converts the first MSR into a second MSR to apply options
12   Pass 4:  converts the second MSR into a
13            LilyPond Score Representation (LPSR);
14   Pass 5:  converts the LPSR to LilyPond code
15            and writes it to standard output.
16
17 Other passes are performed according to the options, such as
18 displaying views of the internal data or printing a summary of the score.
19
20 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:

```

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

```

25.3 Useful options to xml2ly

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

```

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

```

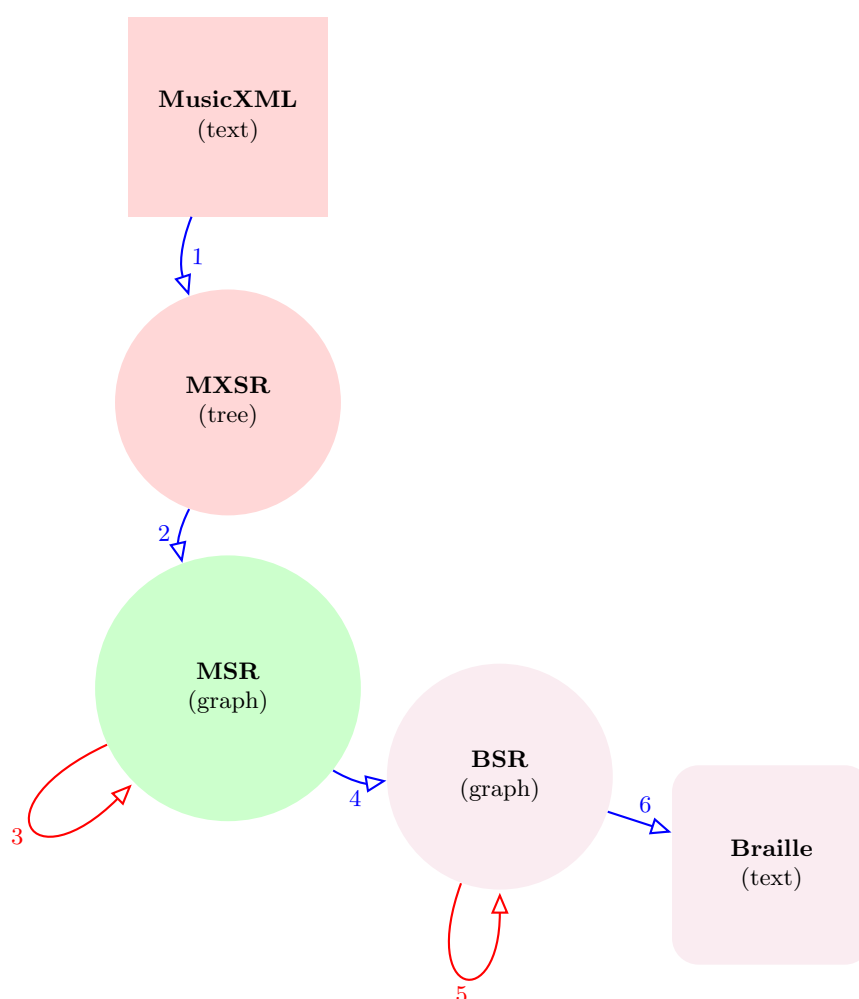
Part X

xml2brl

Chapter 26

xml2brl

Figure 26.1: xml2brl architecture



26.1 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 translator 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 it 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 maintenance guide.

26.2 What xml2brl does

xml2brl performs the 5 steps from MusicXML to LilyPond to translate the former into the latter, as shown in figure 25.1 [xmlToLyArchitecture], page 103,. Converting from MXSR to MSR is done in two sub-phases for implementation reasons.

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

```

1 jacquesmenu@macmini > xml2brl -about
2 What xml2brl does:
3
4     This multi-pass converter basically performs 6 passes:
5         Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
6                 and converts it to a MusicXML tree;
7         Pass 2a: converts that MusicXML tree into
8                 a first Music Score Representation (MSR) skeleton;
9         Pass 2b: populates the MSR skeleton from the MusicXML tree
10                to get a full MSR;
11         Pass 3:  converts the first MSR into a second MSR, to apply options
12         Pass 4:  converts the second MSR into
13                 a first Braille Score Representation (BSR)
14                 containing one Braille page per MusicXML page;
15         Pass 5:  converts the first BSR into a second BSR
16                 with as many Braille pages as needed
17                 to fit the line and page lengths;
18         Pass 6:  converts the BSR to Braille text
19                 and writes it to standard output.
20
21     In this preliminary version, pass 3 merely clones the MSR it receives.
22
23     Other passes are performed according to the options, such as
24     displaying views of the internal data or printing a summary of the score.
25
26     The activity log and warning/error messages go to standard error.

```

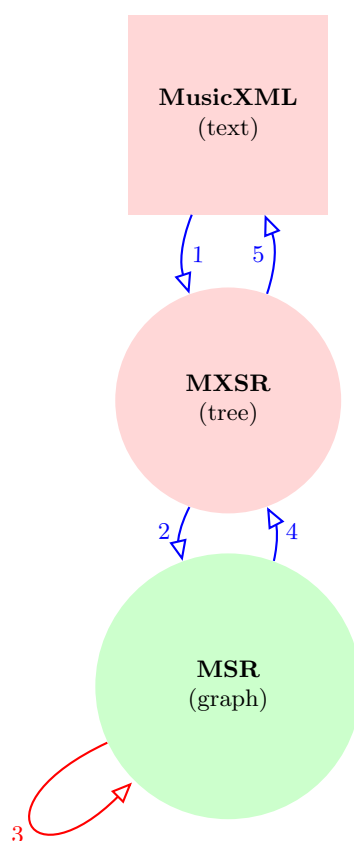
Part XI

xml2xml

Chapter 27

xml2xml

Figure 27.1: xml2xml architecture



27.1 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.

27.2 What xml2xml does

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

in figure 25.1 [xmlToLyArchitecture], page 103,. Converting from MXSR to MSR is done in two sub-phases for implementation reasons.

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

```
1 jacquesmenu@macmini > xml2xml -about
2 What xml2xml does:
3
4 This multi-pass converter basically performs 6 passes:
5     Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
6              and converts it to a MusicXML tree;
7     Pass 2a: converts that MusicXML tree into
8              a first Music Score Representation (MSR) skeleton;
9     Pass 2b: populates the MSR skeleton from the MusicXML tree
10             to get a full MSR;
11     Pass 3:  converts the first MSR into a second MSR, to apply options;
12     Pass 4:  converts the second MSR into a second MusicXML tree;
13     Pass 5:  converts the second MusicXML tree to MusicXML code
14             and writes it to standard output.
15
16 Other passes are performed according to the options, such as
17 displaying views of the internal data or printing a summary of the score.
18
19 The activity log and warning/error messages go to standard error.
```

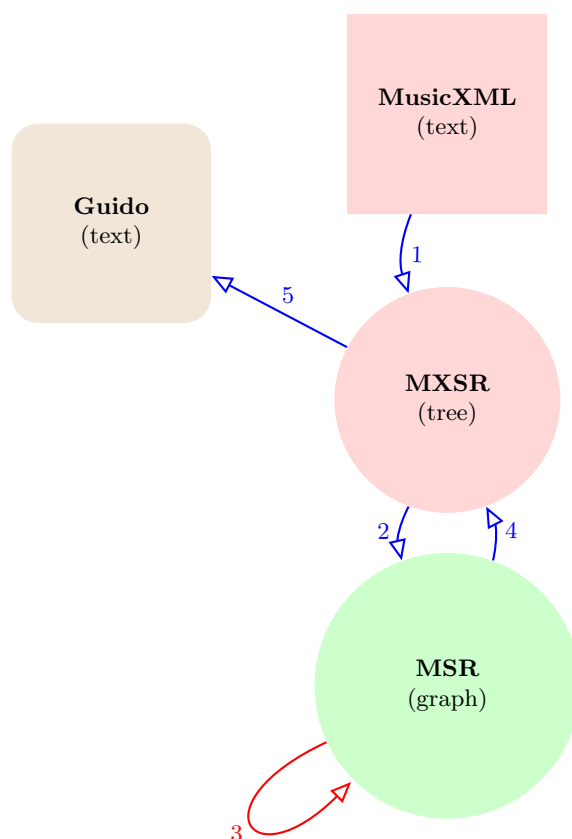
Part XII

xml2gmn

Chapter 28

xml2gmn

Figure 28.1: xml2gmn architecture



28.1 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.

28.2 What xml2gmn does

xml2gmn performs the 5 steps from MusicXML to LilyPond to translate the former into the latter, as shown in figure 25.1 [xmlToLyArchitecture], page 103,. Converting from MXSR to MSR is done in two sub-phases for implementation reasons.

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

```
1 jacquesmenu@macmini > xml2xml -about
2 What xml2xml does:
3
4     This multi-pass converter basically performs 6 passes:
5     Pass 1:  reads the contents of MusicXMLFile or stdin ('-')
6              and converts it to a MusicXML tree;
7     Pass 2a: converts that MusicXML tree into
8              a first Music Score Representation (MSR) skeleton;
9     Pass 2b: populates the MSR skeleton from the MusicXML tree
10             to get a full MSR;
11     Pass 3:  converts the first MSR into a second MSR, to apply options;
12     Pass 4:  converts the second MSR into a second MusicXML tree;
13     Pass 5:  converts the second MusicXML tree to MusicXML code
14             and writes it to standard output.
15
16     Other passes are performed according to the options, such as
17     displaying views of the internal data or printing a summary of the score.
18
19     The activity log and warning/error messages go to standard error.
```

Part XIII

Indexes

Files index

Symbols

*.xml	22
Anacrusis.ly	17
Anacrusis_From_xml2xml.ly	21
Anacrusis_From_xml2xml.xml	20
Hymn121.xml	95
MinimalScore_Jianpu.ly	18
MusicFormatsBashDefinitions.bash	42
MusicFormatsForMacOS.zip	48
MusicFormatsForUbuntu	51, 52
MusicFormatsVersionNumber.txt	48
build/bin	55
error.txt	14
files	25
lilypondstuff/jianpu	27
msdlfiles	25
musicformats-git-dev	25
musicxmlfiles	25
output.ly	14

B

build	
CMakeLists.txt	54
Makefile	54

C

clisamples	
LilyPondIssue34.cpp	30

Mikrokosmos3Wandering.cpp	30
MusicAndHarmonies.cpp	29

F

files	
musicxmlfiles	22
musicxmlfiles/clefs	22

L

libmusicxml/samples	
RandomChords.cpp	29
RandomMusic.cpp	29
libmusicxml/src/files	
musicxmlfiles	23

S

src/clisamples	
LilyPondIssue34.cpp	30
Mikrokosmos3Wandering.cpp	30
MusicAndHarmonies.cpp	29
src/interpreters/mfslinterpreter/mfsl	
Hymn121.xml	91, 96, 97
Hymn121ForBassGuitarPart.mfsl	96
Hymn121ForHarmonyBand.mfsl	100
Hymn121ForVocalSATBQuartet.mfsl	98
Minimal.ly	93
minimal.mfsl	91, 93

Options

Symbols

--help-options-usage, -hou	64	-help-user, -hu	95
-about, -a	31	-ignore-musicxml-lyrics, -imlyrics	96
-all-wedges-below	77	-ignore-ornaments, -oorns	61
-ambitus	59	-ignore-slurs	77
-avoid-msr2msr, -am2m	104	-include	76
-braille	32, 33	-include, -inc	73, 75
-contact, -c	60	-insider, -ins	61, 62, 75
-convert-musicxml-words-to-msr-tempo	77	-jianpu	31, 33, 59
-cpu	79	-lilypond	33
-cubase	59	-mf-version, -mfv	36
-custos	59	-musicxml	33
-display-early-options-values	62	-name-help, -nh	61, 63, 64
-display-options-values, -dov	69	-no-launch, -nol	92, 93
-display-options-values-all, -dova	70	-no-ragged-last-bottom	77
-display-tool-and-input, -dtai	93	-o	59
-every	98	-page-count	77
-find	64, 71	-query	15, 58, 63
-global-staff-size	77	-quiet, -q	60
-guido	33	-repeat-brackets	77
-h	59	-select, -sel	98
-help, -h	95	-title	15
-help-maintainer, -hm	95	-trace-choices, -tchoices	100
-help-midi, -hmidi	63	-vac	60
-help-summary, -hs	65	-version, -v	36
-help-trace	82	-version-full, -vf	36

MusicXML index

Symbols

inversion </>80
lyric </>91, 96
note </>27, 28

part-list </>27, 28
time </>27, 28
words </>77

Main index

Symbols

*.xml	22	-musicxml	33
<inversion />	80	-name-help, -nh	61, 63, 64
<lyric />	91, 96	-no-launch, -nol	92, 93
<note />	27, 28	-no-ragged-last-bottom	77
<part-list />	27, 28	-o	59
<time />	27, 28	-page-count	77
<words />	77	-print	45
-	59, 73, 90, 93	-query	15, 58, 63
--	59	-quiet, -q	60
--help-options-usage, -hou	64	-repeat-brackets	77
-about, -a	31	-select, -sel	98
-all-wedges-below	77	-title	15
-ambitus	59	-trace-choices, -tchoices	100
-avoid-msr2msr, -am2m	104	-trace-voices, -trace-notes	59
-braille	32, 33	-trace=voices,notes	59
-contact, -c	60	-v, -a, -c	60
-convert-musicxml-words-to-msr-tempo	77	-vac	60
-cpu	79	-version, -v	36
-cubase	59	-version-full, -vf	36
-custos	59	40
-display-early-options-values	62	40
-display-options-values, -dov	69	.mfs1	90
-display-options-values-all, -dova	70	/	40
-display-tool-and-input, -dtai	93	/Applications	49
-every	98	2>	14
-exec	45	;	40
-find	64, 71	>	14
-global-staff-size	77	Anacrusis.ly	17
-global-staff-size 25.5	90	Anacrusis_From_xml2xml.ly	21
-guido	33	Anacrusis_From_xml2xml.xml	20
-h	59	C:	40
-h=rests,notes	59	C:\ Program Files	52
-help, -h	95	Ctrl-d	93
-help-maintainer, -hm	95	Extra	80
-help-midi, -hmidi	63	Hymn121.xml	95
-help-summary, -hs	65	LIBRARY_PATH	51
-help-trace	82	LilyPondIssue34	27
-help-user, -hu	95	Makefile	22, 23
-hrests, -hnotes	59	Mikrokosmos3Wandering	27, 30
-ignore-musicxml-lyrics, -imlyrics	96	MinimalScore_Jianpu.ly	18
-ignore-ornaments, -oorns	61	MusicFormatsBashDefinitions.bash	42
-ignore-slurs	77	MusicFormatsForMacOS.zip	48
-include	76	MusicFormatsForUbuntu	51, 52
-include, -inc	73, 75	MusicFormatsVersionNumber.txt	48
-insider, -ins	61, 62, 75	P1	96
-jianpu	31, 33, 59	P4	91, 96
-lilypond	33	PATH	51, 52, 56
-mf-version, -mfv	36	SetMusicFormatsVersionNumber.bash	36
		Stanza	44

#	90	before	75
###	90	Braille	26, 27, 29--33, 106
\$(ls *Stanza*.xml)	44	BSR	26, 29
\	40, 43, 45	builtin	39
{}	45		
all	90, 95, 100	C	
bin	49, 56	C++17	1, 54--56
build/bin	55	cache	79
case	90, 95, 98	Catalina	49
cd	22, 38	choice	90, 95, 98
chmod	92	Chord	34
choice	90, 95, 98	clone	53
cmake	1, 54, 56	clustered	60
default	90	command line	1, 8, 27, 38, 47, 55, 56, 58, 59, 62, 74, 75, 93
dev	25	compiler writing	11
echo	38, 41	contracted	59
error.txt	14	converter	26, 27, 33, 61, 73, 102, 112
every	98	current working directory	38--40
false	66		
files	25	D	
find	22, 44, 45	default	25, 53
for	44	directories	40
input	90, 95	distribution	47
lilypondstuff/jianpu	27	DMG	48
ls	43, 44	Dominique Fober	7, 19, 54, 102
make	23, 54	double quotes	40--42, 74
make install	56	DTD	7
make localinstall	56		
man	39	E	
man find	45	early option	62, 75
mfs1	90, 93	early options	62
msdlfiles	25	environment	
musicformats-git-dev	25	variable	51, 52
musicformats_local_clone	53	ex-nihilo	27
musicxmlfiles	25	executed	58
mxm12msr	77		
name-help	64	F	
o*	19	format	26
output.ly	14	fresco	43
pwd	38	frozen	25, 107
select	90, 95, 98, 100	function	59
sh	38, 43	functions	42
time	79		
title	15	G	
tool	90, 95	Gatekeeper	49, 50
true	66	General button	50
v.	25	generator	27, 33, 61
v0.9.60	48	git	53
v0.9.61	36	GitHub	48
zsh	38	GNU	59
checkVersions ()	42	GraceNotesGroup	34
musicxmlfile2lilypond ()	59	GUI	38, 50, 92
		Guido	19, 27, 30, 32, 112
A			
Allow Anyway	50	H	
API	1, 8, 27, 29, 59	High Sierra	49
atoms	58	home directory	39
automate	15	Hymn121.xml	91, 96, 97
B		Hymn121ForBassGuitarPart.mfs1	96
Bash	38, 58	Hymn121ForHarmonyBand.mfs1	100

Hymn121ForVocalSATBQuartet.mfsl	98	PartGroup	34
I		PartGroupElement	34
input redirection	42	pass	26, 61, 77, 100
insider	61	passes	11
interpreter	90	path	40
J		PDF	47
Jianpu numeric notation	18	Photo Score Ultimate™.....	91
L		phrasé	77
label	98	pipe	42
labels	90, 95	prefix	59
libmusicxml2 ..	7, 19, 26, 29, 31, 33, 54, 112	prompt	38, 58
libraries	47	Q	
LilyPond	7, 11, 12, 14, 15, 17, 19, 21, 26--30, 32--34, 43, 44, 87, 102, 106	quarantine	49
LilyPond code generated	102	quotes	40, 41, 74
lilypond-Jianpu	27, 28	R	
LilyPondIssue34.cpp	30	RandomChords.cpp	29
Linux	22, 47, 54	RandomMusic.cpp	29
LSPR	11, 26, 28, 30, 72	reads	58
M		regular	61
Mac OS™.....	22, 47--49, 54--56, 79	repository	48, 49
MacPorts	54, 56	representation	26, 61
macro options	69	reuse	11
master	25	S	
master branch	53	script	42
MFSL	90--92, 95	Security & Privacy	50
Mikrokosmos3Wandering.cpp	30	Segment	34
Minimal.ly	93	select	90
minimal.mfsl	91, 93	semantic	36
Monterey	49	service	8, 33, 42, 58, 64, 66, 67
MSDL	25, 28, 29, 32	services	33
MSR	11, 26, 28--30, 34, 106, 112	several times	61
multi-pass	19, 26	shebang	92
MusicAndHarmonies.cpp	29	shell	38, 43, 58
MusicFormats ...	1, 8, 11, 12, 15, 17, 19, 22, 25--29, 31--34, 36, 38, 42, 44, 47--50, 53--56, 58, 60--62, 64, 68, 71--73, 76, 79, 87, 90, 95, 107	single step	26
MusicXML	7, 11, 12, 19, 20, 22, 25--34, 43, 44, 77, 80, 87, 91, 96, 102, 106, 107, 109, 112, 113	space	40
musicxmlfiles	22, 23	Staff	34
musicxmlfiles/clefs	22	standard error	13, 14
MXSR	11, 19, 26, 28, 29	standard input	28, 73, 90, 93
N		standard output	13, 14, 28, 29, 32
nested	34	swing	77
Note	34	Syllable	34
O		System Preferences	50
OAHA	8, 15, 26, 27, 58, 59, 63, 69	T	
Open	50	terminal	38
operating system ..	43, 47, 49, 54--56, 79, 92	tool	90
output redirection	42	Tuplet	34
P		U	
Part	34	Ubuntu	48
		Unix	40, 54, 56
		V	
		value	41
		variable	41
		version	53
		view	62
		Voice	34

Main index

VoiceElement [34](#)

W

Web [1](#), [8](#)

well-formed [58](#)

whole [47](#)

Windows™ [40](#), [47](#), [48](#), [56](#)

X

Xcode [54](#), [56](#)

Z

Z shell [38](#)

Zip [48](#)